XACML-Based Web Service Policy Constraint Language (WS-PolicyConstraints)

Working Draft 05, 27 June 2005

Authors:

Anne Anderson (anne.anderson@sun.com)

Balasubramanian Devaraj (bala.devaraj@sun.com)

Abstract:

This specification describes a generic language – *WS-PolicyConstraints* - for expressing one component of a web services policy: the policy constraints on individual policy variables, or “vocabulary items”. Examples of policy constraints include specifying an acceptable value for price or encryption mechanism, a range of acceptable values for bandwidth or time of day, or a set of acceptable values for IP address or security token types. *WS-PolicyConstraints* is suitable for standardization, as it can be used by any domain vocabulary. The vocabulary items themselves may be arbitrary XML schema elements, SAML Attributes, or XACML Attributes, and are defined by other standards groups, consortia, enterprises, etc. outside the scope of this specification.

In addition to specifying the language for expressing constraints, this specification also describes how to use *WS-PolicyConstraints* to verify that a specific set of vocabulary items conforms to a set of constraints, and to compute the intersection of constraints placed on the same vocabulary items in two different policies. An intersection might be needed to determine whether a consumer and a provider, each with their own policies, have any mutually compatible sets of policy constraints.

WS-PolicyConstraints is designed to be used with another “policy sets” language that specifies acceptable sets of constrained vocabulary items. By using *WS-PolicyConstraints* together with a policy sets language, policy processors are provided with all the information required to verify a set of actual values against any policy, or to match or intersect any two policies, without needing any vocabulary-specific code modules or configuration information. This means policy processors do not need to be modified in order to handle policies that use new vocabulary schemas or updates to existing schemas, even if those vocabulary schemas are themselves proprietary.

Status:

This version of the specification is a working draft.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

Table of Contents

1 Introduction (non-normative) 4
1.1 Policy architecture 5
1.2 Policies 5
1.3 Policy example 6
1.4 Policy vocabularies 6
1.5 Policy statements 7
1.6 Specification of policy constraints 8
1.7 Intersection of policy constraints and policies 9
1.8 Policy merging and overwriting 10
1.9 Translation of vocabulary specifications 11
1.10 Use models 11
1.11 Benefits of standardizing policy constraints 13
1.12 Related work 13
1.13 Terminology 14
1.14 Notation 15
2 Specification of constraints (normative) 17
2.1 Vocabulary item references 17
2.2 Literal values 18
2.3 Constraints 18
2.4 Constraint examples (non-normative) 19
3 Validation of vocabulary instances (normative) 21
3.1 Verification of a constraint 21
4 Intersection of policies (normative) 22
4.1 Incompatible constraints 22
4.2 Intersection of constraints 22
5 New constraint functions (normative) 23
5.1 Presence/absence functions 23
5.2 [namespace:]limit-scope 23
5.3 [namespace:]ipAddress-match 24
5.4 [namespace:]string-getKrb5SName 24
5.5 [namespace:]hexBinary-getCertExtensionValue 24
5.6 [namespace:]string-to-uri 24
6 Supported constraint functions and their intersections (normative) 26
6.1 Intersection with semantic hierarchies 27
7 Preferences (normative) 29
8 References 30
8.1 Normative references 30
8.2 Non-normative references 30
Example of policy normalization (non-normative) 32
Acknowledgments 34
Revision History 35
Notices 36

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

77

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1 Introduction *(non-normative)*

This specification describes the syntax and semantics of a language – *WS-PolicyConstraints* - for specifying, verifying, intersecting, combining, and applying **constraints** on the discrete elements, or **vocabulary items**, included in a **policy.** *WS-PolicyConstraints* is designed particularly for web services, but is also suitable for other types of **policies.** *WS-PolicyConstraints* is based on the OASIS Standard *eXtensible Access Control Markup Language* [XACML] and draws particularly from the *XACML profile for Web-services* [WSPL].

A small example will help illustrate the functions of *WS-PolicyConstraints*. Consider a web service that uses a particular schema such as *WS-Security* [WSS] to specify message security headers. One element in this header might be a specification of the length of the encryption key that is used. This "encryption key length" is one **vocabulary item** in the **vocabulary** (the schema) used in the security header to describe message security characteristics. The service may require that encryption key lengths must be greater than or equal to some minimum length considered secure, but less than or equal to some maximum length determined by limits on lengths the service's encryption software is designed to support. *WS-PolicyConstraints* can be used by the service to express the range of key length values that the service will accept. It can be used by a potential client to express one key length that the client wants to use, or a range of key lengths that the client is able to use. Given the service's expression of the range of values that the service will accept and a particular message security header, *WS-PolicyConstraints* can be used to verify that the key length specified in the header satisfies the requirements of the service. Given the service's expression of the key lengths that it will accept, and a potential client's expression of the key lengths it is able or willing to use, *WS-PolicyConstraints* can be used by a service broker to determine that the service and the client have a mutually acceptable range of key length values. A service deployer can use *WS-PolicyConstraints* to describe the range of key length values that the service is to support in a particular deployment (out of a possibly larger range that the service software is capable of supporting).

Specifying **constraints** on **vocabulary items** is only one component of a web service **policy.** Web services **policy** can be described using four functional layers:

1. **Vocabulary specification layer:** this layer specifies the syntax and semantics of discrete elements, or **vocabulary items**, to be used in some particular type of **policy.** Examples of such specifications are *WS-Security*[WSS] and *WS-Reliability*[WSRM]. The *Web Ontology Language* [OWL] attempts to provide a generic language for expressing semantics at this level suitable for use with any **vocabulary.**

2. **Policy constraints specification layer**: this layer specifies sets of values, or constraints, for individual **vocabulary items** in a **policy**. Examples of such specifications are *WS-PolicyConstraints* and, in theory, the *Web Ontology Language* [OWL].

3. **Policy sets specification layer:** this layer specifies which sets of **constrained vocabulary items** are acceptable for a **policy;** i.e. It expresses various interdependent combinations of **constrained vocabulary items** that are acceptable. Examples of such specifications are *WS-Policy* [WSPOLICY] and a proposed extension called *Compositors* [COMPOSITORS] to the Features and Properties section of the *Web Services Description Language (WSDL)* Version 2.0 [WSDL].

4. **Bindings specification layer:** this layer specifies which policies are bound to which aspects of a service. Examples of such specifications are *WS-PolicyAttachments* [WSPA] and the Features and Properties section of WSDL 2.0 [WSDL].

Satisfactory solutions are already in use for the **vocabulary** layer, and new ones appear regularly. Several solutions have been proposed for the **policy sets** and **bindings** layers. To date, however, **policy** designers have either left the expression and interpretation of **policy constraints** to custom software for supporting each type of **vocabulary**, or have proposed use of a semantic web language such as OWL [OWL]. No specific OWL functions have been developed for expressing the types of **constraints** required for web service **policies**, however, and **policy** implementers have not chosen to

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

use semantic web approaches to specifying **policies.** *WS-PolicyConstraints* is a proposed alternative that specifies a rich, but policy-targeted set of functions that can be used with existing **vocabulary** specifications or can be used to express new **vocabularies** in a representation-independent way.

The concepts and use model for *WS-PolicyConstraints* are described in detail in the following sections of this Introduction.

1.1 Policy architecture

The **policy** layers mentioned above, with the addition of **vocabulary translation** and **service metadata** layers**,** are diagrammed in Table 1. The **vocabulary translation** layer is needed in the case where the **policy constraints** do not accept the native syntax used by the **vocabulary**. The **service metadata** layer is where **policy** information is made available to other clients or services.

<table>
<tr><th colspan="4">Layer Specifications</th><th>Layer Descriptions</th></tr>
<tr><td>WS-Security</td><td>WS-Reliability</td><td>...</td><td>...</td><td>Vocabulary Specification: Defines vocabulary items used to characterize a service or aspect of a service. These are the policy variables.</td></tr>
<tr><td colspan="2">XSLT</td><td>XPath</td><td>[pass-through]</td><td>Vocabulary Translation: Translates existing vocabulary syntax to the syntax used by the Policy Constraints Layer if necessary</td></tr>
<tr><td colspan="2" rowspan="2">OWL</td><td colspan="2"></td><td>Vocabulary Semantics: Defines domain-specific semantics of the vocabulary</td></tr>
<tr><td colspan="2">WS-PolicyConstraints</td><td>Policy Constraints: Defines constraints (generic semantics) on individual vocabulary items and how to compute the intersection of such constraints</td></tr>
<tr><td>WSDL Compositors</td><td colspan="3">WS-Policy</td><td>Policy Sets: Defines acceptable sets of policy constraints and how to compute intersection and composition of such sets; constraints themselves are opaque to this layer.</td></tr>
<tr><td>WSDL F&P</td><td colspan="3">WS-PolicyAttachments</td><td>Policy Bindings: Binds policies to service descriptions</td></tr>
<tr><td>WSDL</td><td>ebXML Reg/Rep</td><td>SOAP</td><td>WS-MetadataExchange</td><td>Service Metadata: Makes policies available to clients or other services</td></tr>
</table>

Table 1Policy architecture diagram

1.2 Policies

A **policy** is a description of behavior, particularly of desired or intended behavior. We are familiar with **policies** that govern corporate behavior, such as equal employment **policies** or privacy **policies**, but computer applications also have behaviors, and these can also be described using **polices.** One example familiar to most of us is an "access control **policy":** a description of how an application should behave with respect to allowing access to resources. But this is only one example from a broad range of computer application behaviors that could be described using a **policy.** Such **policies** describe the **constraints** that determine the application's behavior. Some application **constraints** are determined by inherent capabilities; for example, an application either has the code required to support a given message encryption mechanism or not. Other application **constraints** are determined by decisions that constrain how the inherent capabilities are to be used; for example, the deployer of a service application may have enabled only one of two built-in encryption mechanisms. Any **policy** that describes the behavior of a computer application must take into account both the decisions and the inherent capabilities that constrain the application's behavior.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

152 In the web services arena, **policies** may be associated with a entire set of web services, with a single
153 web service as a whole, or with particular parts of a service such as ports or endpoints, bindings, port
154 types or interfaces, operations, messages, or even with fractions of messages. A **policy** may be
155 associated with a generic type of service or with a specific instance of a service or one of its sub-parts. It
156 is the job of the **Policy Bindings** Layer to associate **policies** with services or parts of services. WSDL
157 [WSDL] (using Features and Properties) and *WS-PolicyAttachments* [WSPA] provide ways to associate
158 **policies** with services. WSDL [WSDL] itself, UDDI [UDDI], SOAP [SOAP], and *WS-MetadataExchange*
159 [WSMD] all provide ways in which **policies** and other service metadata are made available to service
160 clients or to other services.

161 1.3 Policy example

162 An example of a web services **policy** for message security headers follows. This example will be used
163 in the subsequent sections.

164 The security token type must always be either a KerberosV5TGT or an X509v3 certificate. If the
165 message part being transmitted is “AccountNumber”, then the 3DES-CBC encryption algorithm
166 must be used; otherwise encryption is optional but must be 3DES-CBC if used.

167 1.4 Policy vocabularies

168 A **policy** uses one or more particular **vocabularies** to specify behaviors. In the world of corporate
169 **policies,** the **vocabulary** for privacy **policies** might include **items** such as “data category”, “user
170 category”, “third party”, “consent to disclose”, “mailing list”, “record retention period”, etc. In the world of
171 web services, as used by this specification, a **vocabulary** is an independent set of technical features and
172 parameters, called **vocabulary items**, that are associated with some aspect of using a web service. For
173 example, the **vocabulary** in the policy in Section 1.3 includes **vocabulary items** such as “security token
174 type”, “message part”, and “encryption algorithm”. “KerberosV5TGT” and “X509v3” are possible values
175 for the “security token type” **vocabulary item,** “AccountNumber” is a value for the “message part”
176 **vocabulary item**, and “3DES-CBC” is a value for the **“**encryption algorithm” **vocabulary item.**

177 As further examples, a **vocabulary** associated with reliable messaging might include **items** such as
178 “time to live”, “number of retries” and “interval between retries”. A **vocabulary** associated with services
179 that offer computational resources might include **items** such as “gigaflops”, “number of processors
180 available”, and “platform architecture”. Items that might occur in many **vocabularies** include “offer
181 price”, “minimum bandwidth”, “date of service”, “time of service request”, “client security token type”, and
182 “client IP address”.

183 **Vocabularies** for web services are defined in written specifications, of which there are several
184 categories. Some specifications describe capabilities associated with almost any web service.
185 Examples of this type of specification include *WS-Reliability*[WSRM] and *WS-Security*[WSS] Such
186 specifications are usually created by cross-industry standards organizations. Other specifications
187 describe capabilities common to a particular class of services, such as services offering computational
188 resources or services offering music downloads. These types of specifications are usually created by
189 standards groups within a specific industry. Yet other specifications describe capabilities associated with
190 a particular service. These types of specifications may be created by an individual vendor or other
191 enterprise, and may be proprietary or internal to the enterprise.

192 **Vocabulary items** may be expressed in various forms. Web service specifications are usually
193 associated with an XML Schema, whose elements make up the **vocabulary. Vocabulary items** may
194 also be expressed as XACML [XACML] or *SAML* [SAML] Attributes, or may be converted to one of these
195 from some other format. A **policy** will often use **vocabulary items** from multiple **vocabularies**. As an
196 example, the **policy** may specify various reliable messaging parameters using a *WS-Reliability*
197 **vocabulary**, but may qualify these by using environmental **vocabulary items** such as time of day
198 specified using an XACML current-time Attribute**.**

199 The semantics, or meanings, of **vocabulary items** are usually expressed in written form as part of the
200 **vocabulary** specification. Semantic web languages such as OWL [OWL] and RDF [RDF] attempt to

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

provide a standard computer-readable description of the semantics of a **vocabulary,** but currently most semantic descriptions are captured only in custom code modules.

Note that, conceptually, **vocabulary items** need not be independent. For example, **item** “age” may be related to another **item** “birthdate”, “ship date” may be related to another item “order date**”**. Requiring **vocabulary items** to be independent, however, eliminates the possibility of various types of errors in **policies,** such as inconsistent circular dependencies (A > B AND B > C AND C > A). More research is needed to identify the types of dependencies that can be supported without compromising the ability to compute **policy** intersections**.**

1.5 Policy statements

A **policy statement** is a computer-readable expression used to convey a **policy** between entities. An application deployer, for example, may use a **policy statement** to convey to the application itself the **constraints** that are to be placed on the application's inherent capabilities. In another example, a service provider application may use a **policy** statement to convey the **constraints** of the application to potential service consumers. In yet another example, a service consumer client application may use a **policy statement** to convey the **constraints** that its user requires of a service to a service broker or to a service itself.

Policies are not always in the form of explicit **policy statements.** An application's **policies** may be hard-coded, or may be set via some application- or platform-specific set of deployment parameters. It should be possible, however, to capture this information in the form of a **policy statement** if needed.

In the past, web service specifications have had to define their own **policy statement** syntax. Greater functionality and interoperability, as well as smaller footprints and easier management of services, is possible if the syntax for **policy statements** is specified in a standard way. Standard **policy statements** can be useful even if not all web service components use them; for example, a client application can be written to understand standard **policy statements** exported by services the client will use even if the client's own **policies** are stored in an internal format – having one format to deal with makes the client easier to design, code, and maintain.

One layer that can be defined to help express standard **policy statements** is a way of describing the combinations of predicates, or **constraints**, on **vocabulary items** that are acceptable for a given **policy**. There are various ways of expressing this aspect of the example **policy** from Section 1.3, but they are all equivalent to the following set of sets, which are expressed in **disjunctive normal form** (an OR of ANDed simple predicates):

```
{ { “SecurityTokenType” EQUALS “KerberosV5TGT”,
   “MessagePart” EQUALS “AccountNumber”,
   “EncryptionAlgorithm” EQUALS “3DES-CBC” },

  { “SecurityTokenType” EQUALS X509v3,
   “MessagePart” EQUALS “AccountNumber”,
   “EncryptionAlgorithm” EQUALS “3DES-CBC” },

  { “SecurityTokenType” EQUALS “KerberosV5TGT”,
   “MessagePart” NOT EQUAL TO “AccountNumber”,
   “EncryptionAlgorithm” EQUALS “3DES-CBC”},

  { “SecurityTokenType” EQUALS “X509v3”,
   “MessagePart” NOT EQUAL TO “AccountNumber”,
   “EncryptionAlgorithm” EQUALS “3DES-CBC”},
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

{ “SecurityTokenType” EQUALS “KerberosV5TGT”,
“MessagePart” NOT EQUAL TO “AccountNumber”,
“EncryptionAlgorithm” NOT PRESENT },

{ “SecurityTokenType” EQUALS “X509v3”,
“MessagePart” NOT EQUAL TO “AccountNumber”,
“EncryptionAlgorithm” NOT PRESENT } }

The **policy** is satisfied if at least one of these sets of **constraints** is satisfied, so this is called the **policy sets** layer. In addition to specifying the **policy sets,** the **policy sets** layer should also provide a way to express relative preferences among the various sets. If the language allows **policies** to be expressed in a form other than **disjunctive normal form**, the language should provide an algorithm for obtaining the **policy sets** in normal form, since this is the best form for automatic matching of **policies.**

At least two proposals have been made for languages that address the **policy sets** layer: *Web Services Description Language 2.0* [WSDL] Features and Properties and *WS-Policy* [WSPOLICY] Both provide operators to create expressions equivalent to the **policy sets** above, but neither proposal currently includes the specification of preferences.

Neither proposal provides ways to represent the EQUALS operator, nor do they provide ways to represent other possible predicate operators such as GREATER THAN, or ELEMENT OF <subset>. So these proposals, and the **policy sets** layer, can be characterized as describing the various sets of **constraints** that are acceptable for a **policy,** but as not describing the **constraints** themselves.

In order to make the management of **policies** tractable, a **policy** usually will consist of multiple sets of **policy sets,** each associated with a different aspect of communication or with a different service interface. For example, there may be one **policy set** for security parameters, another for reliable messaging parameters, and another for service-specific options such as price and quantity. The **policy sets** layer must provide some mechanism for determining which aspect each **policy** applies to; this might be specified, for example, by **constraints** that indicate the **vocabularies** used by a particular **policy set**. The **policy bindings** layer will provide some mechanism for determine which interface each **policy** applies to.

1.6 Specification of policy constraints

The authors of both proposed **policy sets** languages assume that the language for expressing the individual **constraints** in a **policy** will be specified as part of each **vocabulary** specification, and therefore should not be standardized as part of the specification of **policy** itself**.** There are strong advantages to having a standard language for expressing **constraints,** however; and these advantages will be described in Section 1.10.

But first, let's describe what such a **policy constraints** language would look like. Each **constraint** in a **policy** specifies an allowable value, range of values, or set of values for a **vocabulary item** by describing a **vocabulary item**, an operator, and another **vocabulary item** or a literal value. For example, using the **policy statement** in the previous section, the individual **constraints** might be expressed as:

“SecurityTokenType” IS A MEMBER OF {“KerberosV5TGT”, “X509v3”}
“MessagePart” EQUALS “AccountNumber”
“MessagePart” NOT PRESENT
“EncryptionAlgorithm” EQUALS “3DES-CBC”
“EncryptionAlgorithm” NOT PRESENT

A language at the **policy constraints** layer provide ways to express these **constraints.** It should provide a rich set of operators for specifying values, ranges of values, and sets of values of various

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

types. The **policy constraints** language should also provide a way to specify which end of a range of acceptable values, or which elements in a set of acceptable values are preferred.

The **policy sets** layer will use the **constraints** specified by the **policy constraints** layer to express acceptable sets of **constraints:**

{ “SecurityTokenType” IS A MEMBER OF {“KerberosV5TGT”, “X509v3”},
“MessagePart” EQUALS “AccountNumber”,
“EncryptionAlgorithm” EQUALS “3DES-CBC” },

{ “SecurityTokenType” IS A MEMBER OF {“KerberosV5TGT”, “X509v3”},
“MessagePart” NOT EQUAL TO “AccountNumber”,
“EncryptionAlgorithm” EQUALS “3DES-CBC”},

{ “SecurityTokenType” IS A MEMBER OF {“KerberosV5TGT”, “X509v3”},
“MessagePart” NOT EQUAL TO “AccountNumber”,
“EncryptionAlgorithm” NOT PRESENT },

Notice that, by supporting operators such as “IS A MEMBER OF <set>”, the number of **constraints** required to express the **policy sets** can be reduced: rather than one **constraint** to say “SecurityTokenType” EQUALS **“**KerberosV5TGT” and another **constraint** to say “SecurityTokenType” EQUALS “X509v3”, there can be a single **constraint** saying “SecurityTokenType” IS A MEMBER OF {KerberosV5TGT, X509v3}. Using appropriate **constraint** operators is even more important in the case of **vocabulary items** that may have a large range of values, such as time periods, IP addresses, or prices: it would be prohibitive (or impossible, depending on the granularity needed) to try to express every possible time between 9am and 5pm for example.

Vocabulary items that must not be present could logically be handled either by the **policy constraints** layer or by the **policy sets** layer. In the **policy constraints** layer, this can be treated as a constraint on the **vocabulary item** itself. In the **policy sets** layer, this can be treated as a characteristic of the set. One further **constraint** type that could logically be handled at either layer is a **vocabulary item** that must be present, but may have any value (i.e. is unconstrained).

A **policy constraint** language should provide predicate operators such as EQUALS <value>, EQUALS <value set>, IS GREATER THAN <value>, IS GREATER THAN OR EQUAL TO <value>, IS LESS THAN <value>, IS LESS THAN OR EQUAL TO <value>, IS AN ELEMENT OF <value set>, and IS A SUBSET OF <value set>. Depending on the **policy sets** language used, the **policy constraint** language may also need to provide operators for MUST BE PRESENT BUT MAY HAVE ANY VALUE, and MUST NOT BE PRESENT. Since the implementation of the comparison and set operators depends on the data type of the values involved, the **policy constraint** language must provide a way to determine the data type of each **vocabulary item** or **item** value, and should be able to handle a rich set of data types.

The **policy constraint** language should also provide a way of specifying preferred **vocabulary item** values where a **constraint** allows a **vocabulary item** to have a range or set of values.

1.7 Intersection of policy constraints and policies

Use and analysis of **policies** usually requires the ability to compute the intersections of **constraints** and of **policies.** The intersection of two **policies** is a single policy that will accept every set of **vocabulary items** that would be accepted by both of the original **policies.** The intersection of two **policy constraints** is a single constraint that will accept every **vocabulary item** value that would be accepted by both of the original **constraints.** Even if no physical intersection document is constructed, users of **policies** need to have the types of information required to perform an intersection, so precise specification of these semantics is useful.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

As an example of the need for **constraint** intersections, a **policy set** may contain two **constraints** that are inconsistent – they can never both be satisfied by any **vocabulary item** value. More formally, two **constraints** are inconsistent if they place constraints on the same **vocabulary item,** but the intersection of the sets of values accepted by the two **constraints** is empty. A **policy set** that contains two or more inconsistent **constraints** is itself inconsistent, and can be eliminated from the set of **policy sets** because that **policy set** can never be satisfied (this elimination does not change the meaning of the rest of the **policy** since **policy sets** are logically connected by OR operators). If, after all inconsistent **policy sets** have been eliminated, there are no remaining **policy sets,** then the **policy** itself can never be satisfied. For any **policy** that will be evaluated multiple times, it is more efficient to remove inconsistent **policy sets** before doing any evaluation, rather than waste effort evaluating impossible-to-satisfy conditions over and over.

As an example of the need for **policy** intersections, take the case of two web service entities that wish to communicate. In order to do so, they must use a set of **vocabulary item** values that satisfies the **policies** of both entities; i.e. they must use a set of **vocabulary item** values that satisfies the intersection of both **policies.** When **vocabulary items** are independent, the acceptable **vocabulary items** will be those that satisfy all the **constraints** in one **policy set** from the first entity's **policy** and all the **constraints** in one **policy set** from the second entity's **policy.** These combinations of **policy sets** that must be satisfied are represented by the cross-product, or intersection, of the two sets of **policy sets.** Many of the **policy sets** in this intersection are likely to be inconsistent. In order to know whether the two entities have any mutually acceptable **policy**, the inconsistent **policy sets** must be eliminated. If at least one **policy set** remains, then the two **policies** are consistent, and any set of **vocabulary items** that satisfies at least one of the **policy sets** left will be acceptable. Again, eliminating the inconsistent **policy sets** requires being able to determine whether any **constraints** are inconsistent, which is the ability to determine the intersection of **constraints** on the same **vocabulary items.**

A **policy constraint** language should provide an algorithm for determining the intersection of any **constraints.** From that, the intersection of any two **policy sets** and the intersection of any two **policies** can be determined.

Some **policy** engines need to determine not only the **policy** intersection, but also to select from the intersection a specific preferred **policy set** containing preferred values for **vocabulary items**. This selection might be done by various entities: the initiator of a communication, the consumer, the provider, a third party service broker, etc. Since the preferences of a consumer may conflict with the preferences of a provider, the entity that selects the actual **vocabulary item** values to use must have some algorithm for deciding how to resolve conflicts. When the selection is done by the consumer or by the provider, those entities will presumably use their own preferred value. When the selection is done by a third party, some “fair” resolution algorithm might be implemented.

1.8 Policy merging and overwriting

In some cases, **policies** need to be merged or replaced. For example, a **policy bindings** layer language may allow one **policy** to be specified for a service as a whole, and another **policy** to be specified for some particular interface of the service. The **policy bindings** layer language is responsible for defining whether the specific interface **policy** is supposed to replace, strengthen, or weaken the overall service **policy.** Applying a replacement **policy** is straightforward, but computing strengthened or weakened **policies** is somewhat more complex. If one **policy** weakens another, then only a set of **vocabulary items** that satisfies both **policies** is acceptable. The effective joint **policy** in this case is the intersection of the two **policies**, as described in Section 1.7. If one **policy** strengthens another **policy**, then any set of **vocabulary items** that satisfies either **policy** is acceptable. The effective **policy** in this case is a **policy** consisting of the union of the **policy sets** of the two original **policies.**

A **policy** union may contain redundant **policy sets**. That is, one **policy set** may accept a subset of the **vocabulary items** accepted by another **policy set.** For efficient application or for analysis of the effective **policy**, it is useful to eliminate redundant **policy sets.** Formally, a **policy set** is redundant with respect to a second **policy set** if the intersection of the two **policy sets** is equal to the first **policy set**.

So, once again, a **policy constraint** language should provide an algorithm for determining the

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

intersection of any two **constraints.** From that, the intersection of any two **policy sets** can be determined, and the union of any two **policy sets** can be minimized.

1.9 Translation of vocabulary specifications

Sometimes a **vocabulary** specification will not be available to a **policy** processing engine. This might be the case, for example, where the **vocabulary** specification uses a proprietary XML schema.

In these cases, someone with access to the **vocabulary** schema can write a set of XSLT transforms to convert instances of the **vocabulary** specification to a non-proprietary format, such as another XML format or to a set of XACML `<Attribute>`s. The resulting format can be published as an alternative **vocabulary** specification. **Policies** can then be written against this alternative, non-proprietary **vocabulary** specification. Any instance of the proprietary **vocabulary** specification can be processed by the XSLT transforms prior to writing use with **policy** processing entities.

For example, a service using a proprietary **vocabulary** specification could perform the XSLT transforms, and then use the resulting non-proprietary form of the **vocabulary** specification in its WSDL description. **Policies** for that service can then be written against the non-proprietary form of the **vocabulary** specification. Standard **policy** processing engines can process any such **policies**.

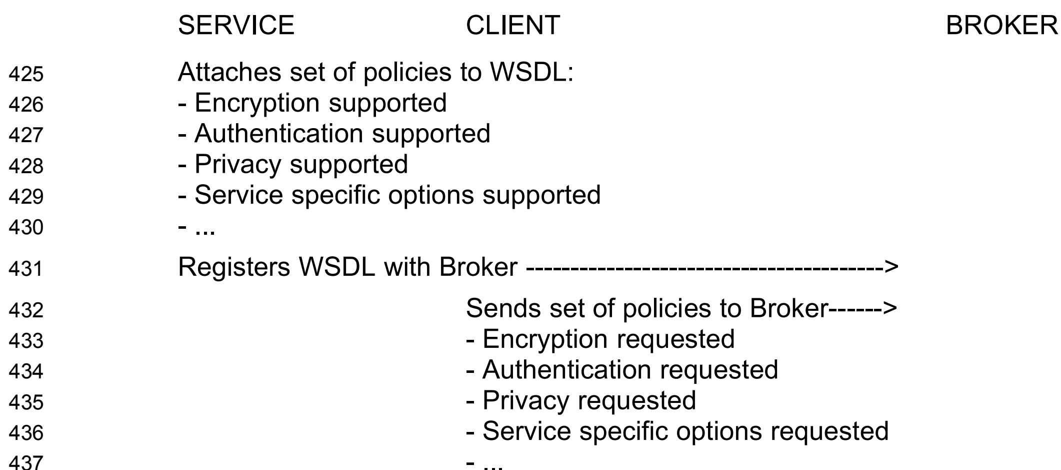
1.10 Use models

So how are these web service **policies** going to be used? This section describes several use models. This is not intended to be an exhaustive list.

Grid service broker

In a Grid environment, service descriptions written in WSDL may be registered with a Grid service broker for each instance of a given service. Each service description contains **policies** associated by that service instance with the service as a whole or with particular interfaces. A potential client sends the service broker a skeleton WSDL instance, identifying one or more service interfaces the client wants to use, along with the client's **policies**. Both the service **policies** and the client **policies** are written in a **policy sets** language that uses *WS-PolicyConstraints* to express the individual **policy constraints.** The service broker computes the intersection of the client's **policies** with each candidate service's **policies**. If the intersection is not empty, then the client's requirements are consistent with those of the service instance, so service broker sends the address of the service instance back to the client, along with the **policy** representing the intersection. The client constructs messages to the service using **vocabulary item** values that satisfy the client's preferred **policy set** in this intersected **policy**. The client sends the set of **vocabulary item** values it intends to use to the service as part of session establishment. The service uses this set in responding to the client's messages.

A possible scenario for this use case is diagrammed below.



Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

Sends policies to Broker as query--->

Intersects client policies with each servers policies until a non-empty intersection found

<------ Returns service's WSDL and policy intersection to client

<----------------- Validates policy intersection against policies. If valid, initiates communication with Server using and sending policy intersection

Validates communication and--------> policy intersection against policies. If validated, continues communication using policy intersection; otherwise, returns error

Figure x: Grid policy usage scenario

WSDL publication

An application server publishes the **policies** for a service instance it supports in a publicly accessible repository as part of the service instance's WSDL description. A client application retrieves the WSDL description from the repository, and extracts the **policy** for an interface the client plans to invoke. The client software may contain generic code modules it uses to match its internal configuration against the **policy constraints** in various **policy sets**. It uses the results to select a preferred **policy set** and preferred values for the **vocabulary items** as constrained by this **policy set.** In this scenario, the client never constructs its **policies** physically; it implements **policy constraint** matching functions internally in a way consistent with the semantics of *WS-PolicyConstraints*. The client's **policy** in this case is virtual, while the service instance's **policy** is a physical instantiation.

Service deployment configuration

A service developer describes the inherent capabilities of a service application in the form of **policies** associated with various service interfaces. A service deployment tool reads these basic **policies** and presents their options (**policy sets)** to a site deployment engineer. The deployment engineer chooses **vocabulary item** values consistent with the preferred **policy sets** for each **policy.** The deployment tool then configures the service instance to use the specified **policies.** As in the case of WSDL publication above, the service instance may not have its **policies** expressed in a physical **policy** schema instance, but might process the **policy constraints** using specific code modules.

Message verification

A service instance verifies the **vocabulary items** used in a message received from a client against the instance's **policy** for the message interface. If the **items** do not comply with the **policy,** the service rejects the message, possibly sending an error describing the failed **constraint** intersections back to the client. A client similarly may verify that the **vocabulary items** used in a response from a service are consistent with the client's **policies** or with a negotiated **policy** that the client and service have agreed to use.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1.11 Benefits of standardizing policy constraints

A web service and its clients must understand the full semantics of a **vocabulary item** in order to implement the associated functionality. For example, the encryption components of a web service and its client must know how to implement an encryption algorithm that is specified as necessary in a **policy.**

But a number of important components that use or process web service **policies** do not need to implement the semantics of **vocabulary items,** but do need to implement the semantics of **constraints** on those items. Some examples of components that need to understand only the **constraints** include:

1. **Policy** verifiers, that must verify that a given service communication conforms to at least one acceptable set of **policy constraints. Policy** verifiers may also need to find the intersection of separately specified **policies**, such as the **policy** for a particular service interface that inherits **constraints** from the **policy** for the entire service, but augments or overrides them with **constraints** of its own.
2. Deployment tools that set the options for a web service the values specified in a deployment **policy**.
3. Service brokers that match clients having given capabilities and requirements against services with **policies** that indicate they support compatible capabilities and requirements. Such brokers need to compute the intersections of **policies.**
4. Service compositors that combine several services to create a single virtual service; compositors must combine the **policies** of the component services into a single set of **policies** for presentation to potential clients of the virtual service. Compositors can simplify the **policies** presented to potential clients if they are able to eliminate subset **policies.** Compositors may also need to determine that the **policies** of services acting as consumers of other services acting as providers are compatible: this involves computing the intersection of two **policies.**
5. **Policy** query interfaces that can handle various types of requests for information about a **policy**. These interfaces may need to find the intersection of service-wide and specific interface **policies**, determine whether a specific set of **vocabulary items** provided by the querying entity satisfy a **policy,** or return the **policy sets** that meet certain criteria provided by the querying entity.
6. **Policy set** appliers that choose one **policy set** from a **policy's policy sets** and applies **vocabulary item** values that satisfy that **policy set** to a particular service communication.
7. **Policy** authoring and analysis tools.

All these functions can be performed by standard, generic code modules if **policy constraints** are specified using a standard **policy constraints** language. These modules can handle any **policy,** even if the **vocabularies** involved are new, changed, or proprietary. This is clearly a big advantage in reducing footprint size, increasing interoperability, improving correctness, reliability, and usability of web service **policies.** With a standard language, it is feasible to invest in good tools for dealing with **policies,** since those tools will be used for multiple functions and with multiple types of **policies.**

1.12 Related work

Various semantic web languages such as OWL [OWL] and its derivatives have been proposed [WSCC] as appropriate for expressing **policy constraints,** since they can theoretically express the semantics of both the **vocabulary items** and of the **constraints** on those **items.** Two problems recommend against this approach. First, none of the current semantic web languages support arithmetic comparison operators. Since this is a major component of **policy constraints,** this is a serious deficiency. But, one could argue, let's add such operators to an existing semantic web language rather than invent a new one. This brings us to the second problem, which is that semantic web languages require specification of far more, and more complex semantics, than a web service **policy** requires. Developing the appropriate taxonomies and ontologies requires someone skilled in semantic web concepts and languages, and such skills are currently not readily available to web services developers. In the future, if

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

529 an appropriate semantic web profile is developed for expressing web service **policies,** it should be easy
530 to translate between any standard **policy constraints** language and such a profile, since they both are
531 expressing generic **constraints.** This *WS-PolicyConstraints* specification does suggest linking to an
532 *Resource Description Framework* [RDF] specification where it is appropriate to consider hierarchical
533 categories or category equivalences in a **policy.** This allows a **policy** processor to match **vocabulary**
534 **items** from **vocabularies** that may express the same **item** at different levels of generalization, or simply
535 by using different names. Using an appropriate RDF taxonomy, for example, it is possible to determine
536 that a "James Bond Movie" **item** satisfies a request for an "Action Movie" **item,** and matches and
537 intersections can then be computed.

538 Besides semantic web approaches, a second potential candidate for handling the functions of a **policy**
539 **constraints** language is the *Object Constraint Language* (OCL) [. This language has many of the types
540 of predicate functions needed in a **policy constraint** language. The arguments against using OCL,
541 however, are that it is not an XML-based language, which is important for web services, and that it is not
542 designed to deal with intersection of predicate functions. In addition, OCL has not been approved by any
543 standards organization and has achieved little commercial acceptance to date.

544 A third candidate is the *XACML profile for Web-services,* or *Web Services Policy Language* [WSPL].
545 WSPL handles the tasks of both the **policy sets** layer and of the **policy constraints** layer, and is
546 specifically designed for web services. WSPL, while it was developed in the OASIS XACML TC
547 standards group, was not accepted for progression toward a standard by that group due to charter
548 conflicts. WSPL also conflicts with both of the dominant candidates for a **policy sets** layer language,
549 *WS-Policy* [WSPOLICY] and WSDL [WSDL]with Compositors [COMPOSITORS], making it politically
550 difficult to progress. This *WS-PolicyConstraints* specification is an attempt to take the predicate
551 language used by WSPL and separate it from the **policy sets** components so that it can be used with
552 other **policy sets** languages. The cost of this separation is that the resulting language is no longer a
553 subset of the OASIS Standard *eXtensible Access Control Markup Language* [XACML], and thus is not
554 as easily used as a vehicle for XACML policies. This cost may be alleviated by translators between
555 XACML and whatever language becomes the approved **policy sets** language. Such translators should
556 be simple to develop, since the proposed **policy sets** languages are basically Boolean operator
557 specifications that map easily to XACML Boolean operators. Not all XACML policies can be expressed
558 in *WS-PolicyConstraints* since XACML supports operators and predicates for which the intersection can
559 not readily be computed. This is also true of WSPL, so there is no additional loss there.

560 A fourth candidate is the *ebXML Collaboration-Protocol Profile and Agreement Specification* [CPP/A].
561 This specification includes one schema (*CPP*) for describing policies regarding a wide range of
562 "collaboration protocols", including data-transport, reliable messaging, signaling, security, packaging, and
563 specific message exchanges, and another schema (*CPA*) that describes an agreed-upon set of
564 collaboration protocol parameters. There is a non-normative appendix that suggests ways in which two
565 *CPP*s could be intersected to produce a *CPA*. *CPP/A* has no general **policy set** or **policy constraint**
566 mechanisms: all of the functionality uses **vocabulary items** that have implicit semantics. The
567 intersection process must implement these implicit semantics, and even then *CPP/A* suggests that many
568 aspects of *CPA* generation may require manual negotiation. Thus *CPP/A* is unable to express
569 **constraints** on arbitrary new **vocabularies,** and depends on users accepting the *CPP/A* **vocabulary.** It
570 does not provide preference, "must not be present", or explicit choice mechanisms. This is an example
571 of an ad hoc **policy** language that has neither generality nor built-in intersection functionality.

572 To summarize, it does not appear that there is any existing language that satisfies the requirements of a
573 **policy constraints** layer. This specification attempts to satisfy those requirements, while building as
574 much as possible on existing standards such as XACML.

575 1.13 Terminology

576 **Attribute** – The identity, data type, and value for a single **vocabulary item.**

577 **Bag –** An unordered collection of values that MAY contain duplicates.

578 **Coincident constraints –** Two **constraints** are **coincident** if they contain references to at least one
579 common **vocabulary item.**

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

580 **Constraint** – See **Policy constraint.**

581 **Disjunctive normal form –** A standard way of organizing a logical expression as sets of predicates,
582 where the predicates in a set are related by AND, while the sets themselves are related by OR; such as
583 "{a AND b AND c} OR {b AND d}". **disjunctive normal form** is useful in determining whether an
584 expression is satisfied by a particular set of terms because each set can be evaluated separately; if any
585 set is satisfied, then the entire expression is satisfied. Any Boolean combination of independent[?]
586 predicates can be expressed in **disjunctive normal form.**

587 **Equivalent policies**: Two **policies** are equivalent if the set of **vocabulary instances** validated by the
588 first **policy** is equal to the set of **policies** validated by the second **policy.**

589 **Policy** – A set of rules that describe some aspect of the behavior of an entity. More concretely, and as
590 used in this specification, it is an expression describing all the acceptable sets of constraints on items in
591 a given vocabulary.

592 **Policy constraint -** A reference to one or more **vocabulary items** along with a specification of particular
593 values allowed for those **items**. These may also be called the predicates in the **policy**, or statements of
594 individual constraints and capabilities in the **policy**.

595 **Policy set** – A set of simple **constraints** (predicates) on **vocabulary items** such that any set of
596 **vocabulary items** that satisfies this set also satisfies the entire **policy.** If a **policy** is expressed in
597 **disjunctive normal form**, then each ANDed set of simple predicates is a **policy set.** A **policy** is the
598 union of all its **policy sets.**

599 **Policy statement –** A concrete representation of a policy.

600 **Valid:** A **vocabulary instance** is **valid** against, or with respect to, a given **policy** if at least one **policy**
601 **set** of the **policy** is satisfied by the **vocabulary instance.**

602 **Vocabulary** – The identities, formats, and semantics of an independent set of technical features and
603 parameters associated with use of a web service. A **vocabulary** may be defined as a set of **Attributes**
604 or as an XML schema. One might think of the **vocabulary** as the set of variables used in stating a
605 **policy.**

606 **Vocabulary instance –** A specific instance of a **vocabulary.** A set of **Attributes** with their values or a
607 schema instance that provides values for various **vocabulary items** in a **vocabulary.**

608 **Vocabulary item** - One independent technical feature or parameter in a **vocabulary.** One might think of
609 this as a variable that can be used in a policy.

610 1.14 Notation

611 The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
612 RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF
613 RFC 2119 [RFC2119].

> 614 "they MUST only be used where it is actually required for interoperation or to limit behavior which
> 615 has potential for causing harm (e.g., limiting retransmissions)"

616 These keywords are thus capitalized when used to unambiguously specify requirements over protocol
617 and application features and behavior that affect the interoperability and security of implementations.
618 When these words are not capitalized, they are meant in their natural-language sense.

619 Conventional XML namespace prefixes are used throughout the listings in this specification to stand for
620 their respective namespaces as follows, whether or not a namespace declaration is present in the
621 example:

622 The prefix `xacml:` stands for the XACML policy namespace.

623 The prefix `xs:` stands for the W3C XML Schema namespace **[XS]**.

624 The prefix `xf:` stands for the *XQuery 1.0 and XPath 2.0 Function and Operators* specification
625 namespace **[XF].**

626 In examples, the following XML Internal Entities are assumed to have been defined in the policy

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

627 specification:

```
628 <!ENTITY data "http://www.w3.org/2001/XMLSchema#">
629 <!ENTITY function "urn:oasis:names:tc:xacml:1.0:function:">
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

2 Specification of constraints (normative)

This section specifies how to express **constraints** on **vocabulary items.**

A **constraint** SHALL be expressed as an XACML[XACML] predicate (`<Apply>` element) over one or more **vocabulary items** or literal values contained in a **vocabulary instance**. The XACML predicate SHALL conform to the restrictions specified in the following sections. A non-normative copy of each relevant section of the XACML Policy Schema is reproduced for the convenience of the reader. The normative syntax and semantics of these XACML schema elements SHALL be that contained in the *XACML 2.0 Core Specification*[XACML].

2.1 Vocabulary item references

A **vocabulary item** used in a **constraint** SHALL be referenced using either an XACML `<AttributeSelector>` or an XACML `<AttributeDesignator>` instance.

```
<xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>

<xs:complexType name="ExpressionType" abstract="true"/>

<xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"
substitutionGroup="xacml:Expression"/>
<xs:complexType name="AttributeSelectorType">
        <xs:complexContent>
                <xs:extension base="xacml:ExpressionType">
                        <xs:attribute name="RequestContextPath" type="xs:string"
use="required"/>
                        <xs:attribute name="DataType" type="xs:anyURI"
use="required"/>
                        <xs:attribute name="MustBePresent" type="xs:boolean"
use="optional" default="false"/>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:element name="ResourceAttributeDesignator" type="xacml:AttributeDesignatorType"
substitutionGroup="xacml:Expression"/>

<xs:element name="ActionAttributeDesignator" type="xacml:AttributeDesignatorType"
substitutionGroup="xacml:Expression"/>

<xs:element name="EnvironmentAttributeDesignator"
type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression"/>

<xs:complexType name="AttributeDesignatorType">
        <xs:complexContent>
                <xs:extension base="xacml:ExpressionType">
                        <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
                        <xs:attribute name="DataType" type="xs:anyURI"
use="required"/>
                        <xs:attribute name="Issuer" type="xs:string"
use="optional"/>
                        <xs:attribute name="MustBePresent" type="xs:boolean"
use="optional" default="false"/>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:element name="SubjectAttributeDesignator"
type="xacml:SubjectAttributeDesignatorType" substitutionGroup="xacml:Expression"/>

<xs:complexType name="SubjectAttributeDesignatorType">
        <xs:complexContent>
                <xs:extension base="xacml:AttributeDesignatorType">
                        <xs:attribute name="SubjectCategory" type="xs:anyURI"
use="optional" default="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject"/>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

Vocabulary item reference examples (non-normative):

1. An XACML `<AttributeSelector>` instance:

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

```
698  <AttributeSelector
699          RequestContextPath="//Signature/SignatureMethod@Algorithm/fn:base-
700  uri()"
701          DataType="&type;anyURI"/>
```

702 This <AttributeSelector> references a **vocabulary item** named `Algorithm` that is an
703 XML attribute in an *XML Digital Signature* schema instance. The `RequestContextPath` in the
704 <AttributeSelector> is an XML attribute that contains an XPath expression that selects the
705 `Algorithm` value as a URI. Since there could conceivably be more than one instance of any
706 given **vocabulary item** in a **vocabulary instance,** the set of all values that are selected by the
707 `RequestContextPath` are returned as a **bag,** which is an unordered collection of values that
708 MAY contain duplicates. If the **vocabulary instance** contains no **items** that match the XPath
709 selection criteria, then the **bag** returned will be empty.

710 2. An XACML <ResourceAttributeDesignator> instance:

```
711  <ResourceAttributeDesignator
712          AttributeId="data-rate"
713          DataType="&type;integer"/>
```

714 This <ResourceAttributeDesignator> references a **vocabulary item** named `data-rate`
715 that is expressed as an XACML Resource **Attribute** in a **vocabulary instance.** The identifier for
716 the **Attribute** is "`data-rate`" (such identifiers are actually URIs or URLs with specific
717 namespaces), and the data type is "`integer`". Again, since there could conceivably be more
718 than one instance of any given **vocabulary item** in a **vocabulary instance,** the set of all values
719 in all Resource **Attributes** that match this identity and data type will be returned as a **bag** of
720 values. If the **vocabulary instance** contains no **Attributes** that match both of the
721 <AttributeDesignator> XML attributes, then the **bag** returned will be empty.

722 Any XACML **Attribute** MAY have an "`Issuer`" XML attribute that identifies the issuer of the
723 Attribute; there is no default value for "`Issuer`". XACML Subject **Attributes** MAY also have a
724 "`SubjectCategory`" XML attribute that explicitly identifies the type of subject (user, machine,
725 application, receiver, etc.); the default "`SubjectCategory`" value is "`&category;access-`
726 `user`", meaning the human user on whose behalf the access is being made.

727 2.2 Literal values

728 A literal value used in a **constraint** SHALL be expressed using an XACML <AttributeValue>
729 instance.

```
730  <xs:element name="AttributeValue" type="xacml:AttributeValueType"
731  substitutionGroup="xacml:Expression"/>
732
733  <xs:complexType name="AttributeValueType" mixed="true">
734          <xs:complexContent mixed="true">
735                  <xs:extension base="xacml:ExpressionType">
736                          <xs:sequence>
737                                  <xs:any namespace="##any" processContents="lax"
738  minOccurs="0" maxOccurs="unbounded"/>
739                          </xs:sequence>
740                          <xs:attribute name="DataType" type="xs:anyURI"
741  use="required"/>
742                          <xs:anyAttribute namespace="##any" processContents="lax"/>
743                  </xs:extension>
744          </xs:complexContent>
745  </xs:complexType>
```

746 2.3 Constraints

747 A **policy constraint** SHALL be expressed using an XACML <Apply> instance, with the addition of an
748 optional `Preference` XML attribute.

```
749  <xs:element name="Apply" type="xacml:ApplyType"
750  substitutionGroup="xacml:Expression"/>
751
752  <xs:complexType name="ApplyType">
753          <xs:complexContent>
754                  <xs:extension base="xacml:ExpressionType">
755                          <xs:sequence>
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

```
                              <xs:element ref="xacml:Expression" minOccurs="0"
maxOccurs="unbounded"/>
                       </xs:sequence>
                       <xs:attribute name="FunctionId" type="xs:anyURI"
use="required"/>
                       <xs:attribute name="Preference" type="xs:string"
use="optional"/>
               </xs:extension>
       </xs:complexContent>
</xs:complexType>
```

Exactly one **vocabulary item** SHALL be referenced in a single **constraint.**

> TBD: this can probably be relaxed under certain restrictions. Problems occur with circular references (A > B, B > C, C > A), and with intersection of references where one vocabulary item is constrained by one or more literal values (A > B, B = 10).

The values for the “`Preference`” attribute are “`greater`” or “`lesser`”. A value of “`greater`” means that greater or later (for time or date types) values are preferred. A value of “`lesser`” means that lesser or earlier values are preferred. This attribute is required when the “`FunctionId`” attribute specifies a function that matches a range of values. This attribute is ignored otherwise. See Section [] for more information about the use of the “`Preference`” attribute.

`<Apply>` instances SHALL NOT be nested except for the purpose of converting a bag to a single value where needed by the **constraint** function or for purposes of limiting the scope of a set of **constraints**. That is to say, an `Expression` used in a **constraint** predicate SHALL be one of the following:

1. An `<AttributeValue>`,
2. An `<AttributeSelector>`,
3. One of the `<AttributeDesignator>` types, or
4. An `<Apply>` element having a `FunctionId` value corresponding to one of the XACML `<type>-one-and-only` functions and a single `Expression` element that is an `<AttributeSelector>` or one of the `<AttributeDesignator>` types.
5. An `<Apply>` element with a `FunctionId` value equal to `"[namespace:]limit-scope"`. These elements may be arbitrarily nested. All **constraints** specified under a `"[namespace:]limit-scope"` function SHALL be considered as if they were individual **constraints** under a single AND operator at the **policy set** level.

Except as specified in the preceding list, the `FunctionId` attribute in an `<Apply>` element SHALL be one of those listed in Table 2 of Section 5.

2.4 Constraint examples (non-normative)

The **constraint**

> “There must be a single `data-rate` **vocabulary item** that must be greater than or equal to 64000”

(where `data-rate` has been defined in the **vocabulary** as an XACML Resource Attribute of type integer), is expressed in *WS-PolicyConstraints* as follows:

```
<Apply FunctionId="&function;integer-greater-than-or-equal">
       <Apply FunctionId="&function;integer-one-and-only">
               <ResourceAttributeDesignator
                       AttributeId="data-rate"
                       DataType="&type;integer"/>
       </Apply>
       <AttributeValue DataType="&type;integer">64000</AttributeValue>
</Apply>
```

The **constraint**

> “The one XML Digital Signature present must use the RSA-SHA1 signature method.”

(where the Digital Signature vocabulary is that defined in *XML Digital Signature*[XMLDSig]), is expressed in *WS-PolicyConstraints* as follows:

```
<Apply FunctionId="function;anyURI-equal">
       <Apply FunctionId="&function;anyURI-one-and-only">
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

```
                <AttributeSelector
                        RequestContextPath=
"//Signature/SignatureMethod@Algorithm/fn:base-uri()"
                        DataType="&type;anyURI"/>
        </Apply>
        <AttributeValue
DataType="&type;anyURI">http://www.w3.org/2000/09/xmldsig#rsa-sha1</AttributeValue>
</Apply>
```

The **constraint**

“At least one of the credit card types accepted must be VISA.”

(where “credit-card-type” is defined in the **vocabulary** as an XACML Resource Attribute with type “string”), is expressed in *WS-PolicyConstraints* as follows:

```
<Apply FunctionId="&function;string-is-in">
        <AttributeValue DataType="&type;string">VISA</AttributeValue>
        <ResourceAttributeDesignator
                AttributeId="credit-card-type"
                DataType="&type;string"/>
</Apply>
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

3 Validation of vocabulary instances (normative)

This section specifies how to **validate** a given **vocabulary instance** against a given **policy** that is in Distributed Normal Form as **policy sets.** It is the task of the Policy Sets Layer to specify how to normalize a given **policy.** [A non-normative example of **policy** normalization is included in Appendix A.]

If all **constraints** in at least one **policy set** in a **policy** have been successfully verified against a given **vocabulary instance**, and if any additional requirements from the **policy sets** layer have been satisfied, then that **vocabulary instance** has been "**validated**" against that **policy.** If the **policy sets** layer handles the specification of **vocabulary items** that are unconstrained or that must not be present, then those requirements must be satisfied as part of the validation of a **vocabulary instance** against a **policy set.**

3.1 Verification of a constraint

A specific **constraint** is verified by dereferencing all **vocabulary item** references in the **constraint** against the given **vocabulary instance**, and evaluating the **constraint** function(s) using the semantics specified in the XACML 2.0 Core Specification[XACML]. If the result is Boolean "`True`", then the **constraint** has been successfully verified against the **vocabulary instance.** If the result is anything other than Boolean "`True`" (e.g. A result of "`False`" or a function error or inability to dereference the **item** reference), then the **constraint** has failed to verify against the **vocabulary instance** and the **policy set** containing the **constraint** has failed to validate.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

4 Intersection of policies (normative)

This section describes how to determine the intersection of two **policies.**

The intersection of two **policies** is a single **policy** that will validate exactly the set of **vocabulary instances** that is the intersection of the sets of **vocabulary instances** validated by the two **policies**. This intersection is computed by taking the cross product of the **policy sets** in the two **policies**.

For example, if the **policy sets** in Policy 1 are:

{A, B, C}, {E, F}

And the **policy sets** in Policy 2 are:

{G, H, I}, {J, K}

Then the cross product is:

{A, B, C} + {G, H, I}, {A, B, C} + {J, K}, {E, F} + {G, H, I}, {E, F} + {J, K}

or

{A, B, C, G, H, I}, {A, B, C, J, K}, {E, F, G, H, I}, {E, F, J, K}

This cross product MAY be simplified by eliminating **policy sets** containing incompatible **constraints.**

4.1 Incompatible constraints

Two **constraints** that reference the same **vocabulary item** are **coincident.** If the intersection of two **coincident constraints** is the empty set, then the **constraints** are incompatible. Any **policy set** that contains incompatible **constraints** MAY be eliminated. The resulting **policy** is **equivalent** to the **policy** prior to the elimination of incompatible **constraints** since the incompatible **constraints** could never return a result of Boolean "True" against any **vocabulary instance.**

4.2 Intersection of constraints

The intersection of two **coincident constraints** is either the empty set, the two **constraints,** or a single **constraint.** The result of taking the intersection of two **constraints,** and the FunctionId and <AttributeValue> of any single resulting **constraint** are specified in Table 2 in Section 5. If the intersection of the two **constraints** is the empty set, then the **constraints** are incompatible**.**

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

5 New constraint functions (normative)

This section defines several new functions that MAY be used in specifying **policy constraints.** Any compliant *WS-PolicyConstraints* processor SHALL implement these functions.

5.1 Presence/absence functions

Where a **policy** requires that certain **vocabulary items** must not be present, or must be present but may have any value, and if the **policy sets** language lacks a mechanism for indicating these types of requirements, then a **policy set** MAY be augmented with an instance of one of the following functions as appropriate for purposes of computing intersections.

[namespace:]must-be-present

This function SHALL take one `<AttributeDescriptor>` or an `<AttributeValue>` with data type "&xml;string" as its argument and SHALL return a Boolean result. The `<AttributeValue>` string SHALL be interpreted as an XPath expression. The result SHALL be "true" if the bag containing the values of the specified `<AttributeDescriptor>` or the nodeset representing the XPath expression contains at least one value; the result SHALL be "false" otherwise.

This function is equivalent to applying the `"bag-size"` function to the `<AttributeDescriptor>` and comparing the result using the `"&function;integer-greater-than-or-equal"` function to the integer value "1".

[namespace:]must-not-be-present

This function SHALL take one `<AttributeDescriptor>` or an `<AttributeValue>` with data type "&xml;string" as its argument and SHALL return a Boolean result. The `<AttributeValue>` string SHALL be interpreted as an XPath expression. The result SHALL be "true" if the bag containing the values of the specified `<AttributeDescriptor>` or the nodeset representing the XPath expression contains no values (i.e. is an empty bag or set); the result SHALL be "false" otherwise.

This function is equivalent to applying the `"bag-size"` function to the `<AttributeDescriptor>` and comparing the result using the `"function:integer-equal"` function to the integer value "0".

5.2 [namespace:]limit-scope

This function is used to group **constraints** that must all be satisfied by a single element in an XML schema instance.

This function SHALL take two or more parameters, where the first parameter is and `<AttributeValue>` with data type "&datatype;string", and the remaining parameters are **policy constraints**. It SHALL return a Boolean result. The `<AttributeValue>` used as the first parameters SHALL be interpreted as an XPath expression. The result SHALL be "true" if the **constraints** are all "true" when applied to at least one single **node** in the nodeset selected by the `<AttributeValue>`'s XPath expression. That is, all **constraints** must be satisfied by the same node, although there may be multiple such **nodes** where all **constraints** are satisfied.

Instances of the "limit-scope" function SHALL be treated for purposes of **policy** intersection as if all the **constraints** were individual **constraints** specified separately without the "limit-scope" function. This function may be nested arbitrarily deeply, but all **constraints** are treated as if they were specified at the top-most **policy set constraint** level.

Non-normative example:

```
<Apply FunctionId="[namespace:]limit-scope">
    <AttributeValue DataType="&xml;string">/security/key-
info</AttributeValue>
    <Apply FunctionId="&function;integer-equal">
```


Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

```
            <AttributeSelector ElementId="/key-length"
    DataType="&xml;integer">
            <AttributeValue DataType="&xml;integer">96</AttributeValue>
        </Apply>
        <Apply FunctionId="&function;string-equal">
            <AttributeSelector ElementId="/algorithm"
    DataType="&xml;string">
            <AttributeValue DataType="&xml;string">DES-CBC</AttributeValue>
        </Apply>
    </Apply>
```

This function returns "`true`" only if there is at least one "`/security/key-info`" element that has a "`key-length`" child with value `96` and an "`algorithm`" child with value "`DES-CBC`". [XPath syntax might treat parent "limit-scope" as new root, or we might require evaluator to follow only paths that lead into same element.]

This function specifies two **policy constraints**. For intersection purposes, these **constraints** are considered as if they were enclosed with an AND operator at the **policy sets** level.

5.3 [namespace:]ipAddress-match

This function is used to specify a range of IP address values, optionally qualified by port-ranges.

This function SHALL take two parameters, where the first parameter that is an <AttributeValue> of type "&xml;string" and the second parameter is an <AttributeSelector> or <AttributeDescriptor" with datatype "&datatype;ipAddress".

This function SHALL return "True" if the IP addresses and ports (if included) specified by the <AttributeSelector> or <AttributeDescriptor> are a subset of those specified by the <AttributeValue>.

[Note: this function is used rather than the XACML ipAddress-regexp-match function because it is very difficult to specify ranges of IP addresses and port using regular expressions, and because taking the intersection of two such regular expressions may not be well-defined.]

5.4 [namespace:]string-getKrb5SName

This function is used to obtain the service's "sname" from a Kerberos ticket.

This function SHALL take one parameter, which SHALL be an <AttributeSelector> or <AttributeDescriptor> with datatype "&xml;base64Binary".

This function SHALL return a value with datatype "&xml;string". This value SHALL be the "sname" of the service in the Kerberos ticket specified by the <AttributeSelector> or <AttributeDescriptor>. If this value does not contain a valid Kerberos ticket, then the result SHALL be "Indeterminate" (error).

5.5 [namespace:]hexBinary-getCertExtensionValue

This function is used to obtain the value of an extension in an X.509 certificate.

This function SHALL take three parameters, where the first parameter is an <AttributeValue> of type "&xml;string", the second parameter is another <AttributeValue> of type "&xml;string", and the third parameter is an <AttributeSelector> or <AttributeDescriptor> of type "&xml;base64Binary".

This function SHALL return the value contained in the X.509 certificate extension from the X.509 certificate specified by the third parameter having the OID specified in the first parameter and the criticality specified in the second parameter. The acceptable values of the second parameter are "Critical", "NotCritical", and "CriticalOrNot". The third parameter must contain an ASN.1 encoded X.509 certificate. If any of the parameters is invalid, then the result SHALL be "Indeterminate" (error).

5.6 [namespace:]string-to-uri

This function takes as input a value of type "&xml;string", and produces as output a value of type "&xml;anyURI". The input parameter SHALL be a string that can be interpreted as a URI. If the input

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

962 value is not a valid URI, the result of this function SHALL be “Indeterminate” (error).

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

6 Supported constraint functions and their intersections (normative)

The following table lists the XACML functions that are supported for use in **policy constraints**, and describes how to compute the intersection of any two **constraints.**

If two **constraints** in the same **policy set** constrain the same **vocabulary item**, then they are said to be **coincident.** If two **constraints** in the same **policy set** are not **coincident,** then their intersection is the two original **constraints.**

If two **coincident constraints** are not compatible according to the "Compatibility test" column of Table 2 below, then the two **constraints** are incompatible and the **policy set** containing them MUST be discarded.

If two **coincident constraints** are compatible according to the "Compatibility test" column, then their intersection is the **constraint** specified in the "Replacement constraint" column of Table 2.

Table 2 is to be interpreted according to the following key.

Columns one, two and four contain shorthand versions of an XACML `<Apply>` element. The portion before the open parenthesis (e.g. "type-equal" in the first row) represents the `<Apply>` element's `FunctionId` attribute value. The "type-" portion represents any of the type-specific parts of the standard XACML function identifiers.

Alphabetic symbols (e.g. "a" in the first row) represent XACML `<AttributeDesignator>`, `<AttributeSelector>` or `<AttributeValue>` elements. Where a **constraint** `FunctionId` takes a single value rather than a bag for an argument where an `<AttributeDesignator>` or `<AttributeSelector>` is used, the `<AttributeDesignator>` or `<AttributeSelector>` SHALL be enclosed in an inner `<Apply>` element having as its `FunctionId` the appropriate "type`-one-and-only`" function.

Where "Keep both constraints" appears in the "Replacement constraint" column, there is no single replacement `<Apply>` element: the **predicates** are compatible, but not combinable. In these cases, the original `<Apply>` elements MUST NOT be modified by this step in the procedure.

any-constraint(a) is "true" if there is any **constraint** in the **policy set** that applies to **vocabulary item** a. no-constant(a) is "true" if there is no **constraint** in the **policy set** that applies to **vocabulary item** a.

∩ means set intersection.

⊆ means "is a proper subset of".

	First constraint	Second constraint	Compatibility test	Replacement constraint
1	type-equal(a,b)	type-equal(a,c)	b == c	type-equal(a,b)
2	type-equal(a,b)	type-greater-than(a,c)	b > c	type-equal(a,b)
3	type-equal(a,b)	type-greater-than-or-equal(a,c)	b ≥ c	type-equal(a,b)
4	type-equal(a,b)	type-less-than(a,c)	b < c	type-equal(a,b)
5	type-equal(a,b)	type-less-than-or-equal (a,c)	b ≤ c	type-equal(a,b)
6	type-greater-than (a,b)	type-greater-than(a,c)		type-greater-than(a,max (b,c))

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

7	type-greater-than (a,b)	type-greater-than-or-equal(a,c)		Where b ≥ c	type-greater-than (a,b)
8				Where b < c	type-greater-than-or-equal (a,c)
9	type-greater-than-or-equal(a,b)	type-greater-than-or-equal(a,c)		type-greater-than-or-equal (a,max(b,c))	
10	type-less-than (a,b)	type-less-than(a,c)		type-less-than(a,min(b,c))	
11	type-less-than (a,b)	type-less-than-or-equal (a,c)		Where b > c	type-less-than-or-equal(a,c)
12				Where b ≤ c	type-less-than (a,b)
13	type-less-than-or-equal(a,b)	type-less-than-or-equal (a,c)		type-less-than-or-equal (a,min(b,c))	
14	type-greater-than (a,b)	type-less-than(a,c)	b < c	Keep both constraints	
15	type-greater-than (a,b)	type-less-than-or-equal (a,c)	b < c	Keep both constraints	
16	type-greater-than-or-equal(a,b)	type-less-than(a,c)	b < c	Keep both constraints	
17	type-greater-than-or-equal(a,b)	type--less-than-or-equal (a,c)	b < c	Keep both constraints	
18	set-equals(a,b)	set-equals(a,c)	b == c	set-equals(a,b)	
19	set-equals(a,b)	subset(a,c)	b ⊆ c	set-equals(a,b)	
20	subset(a,b)	subset(a,c)	∩ (b,c) ≠ 0	subset (a,∩ (b,c))	
21	must-be-present (a)	Any constraint other than must-not-be-present(a)		The second constraint	
22	must-not-be-present(a)	must-not-be-present(a)		must-not-be-present(a)	
23	any-constraint(a)	no constraint(a)		any-constraint(a)	
24	time-in-range (a,b,c)	time-equal(a,d)	b ≤ d ≤ c	time-equal(a, d)	
25	time-in-range (a,b,c)	time-greater-than-or-equal(a,d)	b ≤ d ≤ c	time-in-range(a, d, c)	
26	time-in-range (a,b,c)	time-less-than-or-equal (a,d)	b ≤ d ≤ c	time-in-range(a, b, d)	
27	type-regexp-match (a, b)	type-equal(c, b)	∩ (a,c) ≠ 0	type-equal(c, b)	
28	type-regexp-match (a, b)	Type-regexp-match(c, b)	∩ (a,c) ≠ 0	type-regexp-match(∩ (a,c), b)	

Table 2 - Intersection of coincident constraints

6.1 Intersection with semantic hierarchies

The values used in a **constraint** may be specified to be associated with a semantic hierarchy, either by the **policy sets** layer or by the **vocabulary specification** layer.

The intersection of two **coincident** equality **constraints** using values that are not equal, but are in the same semantic hierarchy SHALL be an equality **constraint** in which the **vocabulary item** is the same and the literal value is the value that has the union of the semantics of the two input values (i.e. the descendant, or most specific, value). If the two values are specified as having equivalent semantics, then the intersection SHALL be either one of the two input **constraints.** If the values are not in the same semantic hierarchy, then the two **constraints** are not compatible.

Non-normative example: if the first **constraint** is “string-equal(Movie type, “Action movie”)' and

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1007 the second **constraint** is 'string-equal(Movie type, “James Bond movie”)', and “James Bond
1008 movie” inherits all the semantics of ”Action movie”, then the intersection is the **constraint** 'string-
1009 equal(Movie type, “James Bond movie”)'.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

7 Preferences (normative)

Unless specified otherwise in the **policy sets layer,** a **policy** issuer's preferences SHALL be considered to be first in order of **policy set** preference, and then in order of value preference for each **vocabulary item** in a **policy set.**

Unless **policy set preferences** are specified using another mechanism in the **policy sets** layer, then **policy sets** SHALL be considered preferred in the order specified when the **policy** is in Disjunctive Normal Form. If the **policy** is not specified in Disjunctive Normal Form, then a DNF generation algorithm preserving preference SHALL generate sets using nodes to the left under any operator having “OR” semantics before using nodes to the right.

> Non-normative note: If **policy sets** are generated using an algorithm that produces results equivalent to that in Appendix A, then those **policy sets** will be generated in order of preference.
>
> Non-normative example: If the **policy** is equivalent to the following (where A, B, C, and D are individual **constraints** on **vocabulary items):**
>
> AND(OR(A,B), OR(C,D))
>
> then the **policy sets** in preference order are:
>
> {A, C}, {B, C}, {A, D}, {B, D}

Unless **constraint** preferences are specified using another mechanism in the **policy sets** layer, **constraints** SHALL be considered preferred in the order that they are specified in a **policy set** that has been generated in **policy set** preference order. That is, if two **constraints** reference the same **vocabulary item**, then the **constraint** listed earlier in the **policy set** is preferred over a **constraint** listed later in the same **policy set**.

Some **constraints** express a set or range of **vocabulary item** values. In such a case, unless **vocabulary item** value preferences are specified using another mechanism in the **policy sets** or **vocabulary item** definition layers, these preferences SHALL be specified as follows. For a set, values SHALL be considered preferred in the order they are specified in the set of values. For **constraints** that specify a range of **vocabulary item** values, the “`Preference`” XML attribute in the “`Apply`” element SHALL be used to indicate whether greater or lesser values in the range are preferred.

Non-normative examples:

In the **constraint**

```
<Apply FunctionID=”time-in-range” Preference=”greater”>
      <AttributeSelector RequestContextPath=”.../transaction-
time/text()”>
      <AttributeValue>9am</AttributeValue>
      <AttributeValue>5pm</AttributeValue>
</Apply>
```

the preferred values are the ones closer to 5pm, since those time values (in 24-hour time) are greater than values closer to 9am.

In the **constraint**

```
<Apply FunctionId=”time-is-in”>
      <AttributeSelector RequestContextPath=”.../transaction-
time/text()”>
      <Apply FunctionId=”time-bag”>
            <AttributeValue>5pm</AttributeValue>
            <AttributeValue>8pm</AttributeValue>
            <AttributeValue>7am</AttributeValue>
      </Apply>
</Apply>
```

the most preferred value is 5pm, followed by 8pm, followed by 7am, since the values are specified in this order in the **constraint.**

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

8 References

8.1 Normative references

[RFC2119] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

[XACML] T. Moses, ed., eXtensible Access Control Markup Language (XACML), OASIS Access Control (XACML) TC, Committee Draft 04, 6 December 2004, http://www.oasis-open.org/committees/download.php/10414/access_control-xacml-2.0-core-cd-04.zip.

8.2 Non-normative references

[Barth04] A. Barth, et al., Conflict and combination in privacy policy languages, Workshop on Privacy in the Electronic Society, 8 November 2004, http://portal.acm.org/citation.cfm?id=1029195&coll=portal&dl=ACM

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

[CPP/A] Collaboration-Protocol Profile and Agreement Specification Version 2.0, OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, 23 September 2002, http://www.oasis-open.org/committees/ebxml-cppa/documents/ebCPP-2.0.pdf

[OCL] Response to the UML 2.0 OCL RfP (ad/2000-09-03), Revised Submission, Version 1.6, 6 January 2003, OMG Document ad/2003-01-07, http://www.omg.org/docs/ad/03-01-07.pdf.

[OWL] W3C, Web Ontology Language,

[RDF] W3C, Resource Description Framework (RDF),

[SAML] S. Cantor, et al., eds., Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Security Services TC, OASIS Committee Draft 02, 24 September 2004, http://www.oasis-open.org/committees/download.php/9455/sstc-saml-core-2.0-cd-02.pdf.

[SOAP] M. Gudgin, et al., SOAP Version 1.2 Part 1: Messaging Framework,

[UDDI] UDDI Version 2 Specifications, OASIS UDDI Specification TC, OASIS Standard, July 2002, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2.

[WSCC] W3C, W3C Workshop on Constraints and Capabilities for Web Services, 12-13 October 2004, http://www.w3.org/2004/09/ws-cc-program.html.

[WSDL] E. Christensen, et al., Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001, http://www.w3.org/TR/wsdl.

[WSMD] WS-MetadataExchange, http://www-106.ibm.com/developerworks/library/specification/ws-mex/

[WSRM] K. Iwasa, et al., eds., WS-Reliability v1.1, OASIS Web Service Reliable Messaging TC, OASIS Standard, 15 November 2004, http://www.oasis-open.org/committees/download.php/9330/WS-Reliability-CD1.086.zip.

[WSPOLICY] S. Bajaj, et al., Web Services Policy Framework (WS-Policy), September 2004, http://www-106.ibm.com/developerworks/library/specification/ws-polfram/

[WSPA] WS-PolicyAttachments

[WSPL] T. Moses, ed., XACML profile for Web-services, OASIS Access Control (XACML) TC, Working Draft 04, 29 September 2004, http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1106 **[WSS]** T. Nadalin, et al., eds., Web Services Security: SOAP Message Security 1.0
1107 (WS-Security 2004), OASIS Web Services Security TC, OASIS Standard
1108 200401, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-
1109 wss-soap-message-security-1.0.pdf.
1110 **[XMLDSig]** W3C, XML Digital Signature,
1111

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

A. Example of policy normalization (non-normative)

[Does this section belong? WS-Policy makes a big deal of normalization. Having this section means any Boolean-equivalent **policy sets** language can be handled, and makes it clear that WS-Policy is not special. This section also allows the algorithm to take into account **policy set** level handling of must be present/absent versus **policy constraint** level specification.] In any case, this section is half-baked and needs to be corrected. Reference to a standard algorithm would be very nice. Otherwise, reference the algorithm used in "The Simple Digital Library Interoperability Protocol (SDLIP-Core)" http://www-diglib.stanford.edu/~testbed/doc2/SDLIP/javadoc2/sdlip/helpers/QueryUtil.html or javax.wbem.query.QueryExp class, described at http://wbemservices.sourceforge.net/WBEMSDKDG_html/p25.html.http://wbemservices.sourceforge.net/WBEMSDKDG_html/p25.html contains a method, "canonizeDOC", that could be used to convert an arbitrary tree containing Boolean predicates into Disjunctive Normal Form.

Any **policy** that is expressed in the form of a Boolean combination of independent[?] predicates can be converted to **disjunctive normal form (DNF)** via standard algorithms. For many **policy** operations that require matching or intersecting **policy sets,** it is not necessary to convert the entire **policy** to **DNF**: **policy sets** can be constructed and tested incrementally, terminating the process when a suitable **policy set** is found.

For convenience, an example of such an algorithm is provided here, where the **policy** is assumed to be in the form of a tree with root node `<root>`, where each node in the tree is <AND>, <OR>, <NOT>, or a simple predicate. A <NOT> node may have only a single child node.

In this example, required presence or absence of vocabulary items in a particular **policy set** is handled via [???].

1. Initialize:
   PolicySets={}
   Combinations={<root>}
2. Select first element node C in Combinations.
3. If C contains no non-leaf nodes, then C is a valid **policy set.** Move C from Combinations to PolicySets.
4. If Combinations is empty, terminate. PolicySets contains all **policy sets** in the original **policy.**
5. Choose first non-leaf child N of C.
6. If N is <AND>, replace C with a copy of C where N has been replaced by all children of N. If N is <OR>, make a copy of C for each child of N. In each copy, replace C with one of the children of N. If doing a breadth-first construction, replace C with the copy containing the first child of C, and append remaining copies to the end of Combinations. If doing a depth-first construction, replace C with all copies. If N is <NOT>, if the child of N is a simple predicate, replace N with the negation of the child of N. If the child of N is <AND>, then for each child c[i] of AND, replace the child with <NOT> c[i]. Replace N with these modified children. If the child of N is <OR>, then...
7. Go to step 2.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

B. Future work

This section describes further work that might be considered for inclusion in this specification.

1. THEORY:
    1. Dependent vocabulary items: definition currently says “independent”. Under what conditions can we handle AttrA >= AttrB types of constraints? E.g. A >= B ∩ A >= 3 is both constraints. A >= B ∩ A = B is A = B. Have to worry about A > B, B > C, C > A types of circular (and inconsistent) dependencies.
2. FUNCTIONS/OPERATORS:
    1. Intersectable XPath expressions: although it is easy to determine, when applied to a specific schema instance, whether two different XPath expressions select the same set of nodes, it is not possible to determine this in general for any schema instance. A subset of XPath is needed for which intersections of two XPath expressions can be determined with respect to any valid schema instance. Proposal: prohibit XPath query operators and element order specifiers [x]. All expressions must be absolute rather than relative (in the case of the limit-scope function, they can be relative to the node(s) selected by the limit-scope function.
    2. Add more XACML standard functions to the table? How about high-order bag functions?
    3. Add reference to an algorithm for computing intersection of two regular expression (intersection of two FSAs): Janusz A. Brzozowski, “Derivatives of Regular Expressions”, Journal of the ACM, V. 11, Issue 4, Oct. 1964, pp. 481-494, http://portal.acm.org/citation.cfm?id=321249. Algorithms in Java at: http://dmoz.org/Computers/Programming/Languages/Regular_Expressions/Java/
    4. Add support for arithmetic operators? e.g. Birthdate + 21 years <= CurrentDate?
    5. Day of Week match function? NO. This can be handled via sets of days fairly easily.
    6. Specify support for certain additional nested functions, such as “string-to-uri” surrounding “string-concatenate” functions used to create a URI from a reference and a URI fragment.
3. INTERSECTION ISSUES
    1. Intersection where one policy constrains an item, but another policy being intersected with it does not: WS-Policy says “If the vocabulary of one policy includes an assertion type that is not in the vocabulary of another policy, then the behavior associated with that assertion type is prohibited in the intersection of those policies. Is this adequate? I would think “item is not constrained” is a more logical interpretation of missing assertion types, and explicit use of “must not be present” function represents “assertion type is prohibited”. For example, if one policy says something must be encrypted, and another policy does not constrain encryption, then certainly the intersection is not to prohibit encryption: either the policies are incompatible, or the combined policy must require encryption. Needs to be worked out together with “must be present”/”must not be present” semantics.
    2. Way to indicate that intersection should include all instances of some element that occurs in both policies. Example from CPP/A is that all comments from both client and server CPPs should be included in the CPA.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

C. Acknowledgments

The authors would like to acknowledge the contributions of the OASIS Access Control (XACML) Technical Committee, and of the editor and contributors to the XACML profile for Web-services[WSPL] on which this specification is based.

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1192 # D. Revision History

Rev	Date	By Whom	What
01	15 Dec 2004	Anne Anderson	Initial version based on scaling back Web services profile of XACML.
02	21 Dec 2004	Anne Anderson	Intro improved; “must be present”/not added
03	7 Jan 2005	Anne Anderson	Add preferences, semantic hierarchies, more supported functions, more complete TBD section.
04	20 Jan 2005	Anne Anderson	Corrected semantic hierarchy handling, added full non-normative WS-Security example in an Appendix. Added diagram for Grid usage scenario. Changed syntax of must-be-present and must-not-be-present functions to take an XPath string rather than an <AttributeSelector> (since <AttributeSelector> returning non-leaf nodeset is undefined). Added limit-scope function for applying multiple constraints to a single XML node and its children. Added ipAddress-match function for matching IP addresses, subnets, and port ranges.
05	27 Jun 2005	Anne Anderson	Minor edits; changed legal notice. Added functions string-getKrb5SName, hexBinary-getCertExtensionValue, and string-to-uri. Removed WS-Security example, since that is now a separate document.

1193

Copyright © Sun Microsystems, Inc. 2004-2005. All Rights Reserved.

1194 # E. Notices

1195 Copyright © 2004-2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054,
1196 U.S.A. All rights reserved.

1197 Sun, Sun Microsystems, the Sun logo and Java are trademarks or registered trademarks of Sun
1198 Microsystems, Inc. in the U.S. and other countries.

1199 THIS DOCUMENT IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS,
1200 REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF
1201 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE
1202 DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY
1203 INVALID.