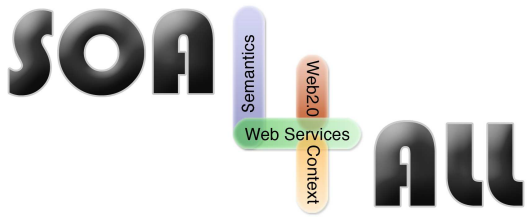

j

Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D3.4.5. Semantic Annotations For WS-Policy (SA-Policy)

Activity N:	A2 Core R&D
Work Package:	WP3 Service Annotation and Reasoning
Due Date:	M24
Submission Date:	01/02/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	EBM WS
Revision:	1.0
Author(s):	Nicolas Boissel-Dallier – EBM
Reviewer(s):	Yosu Gorroñogoitia – ATOS

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	15/01/2010	Initial content	Nicolas Boissel-Dallier (EBM)
0.5	01/02/2010	Draft	Nicolas Boissel-Dallier (EBM)
0.6	15/02/2010	First review	Yosu Gorroñogoitia (ATOS)
0.9	26/02/2010	Incorporating reviewer comments	Nicolas Boissel-Dallier (EBM)
1.0	26/2/2010	Minor typo corrections	Barry Bishop (UIBK)
1.1	28/02/2010	Final format check	Julia Wells (ATOS)

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
IRI	Internationalized Resource Identifier
OWL	Ontology Web Language
QName	Qualified Name
RDF	Resource Description Framework
SAWSDL	Semantic Annotations for Web Service Description Language
WP	Work Package
WSDL	Web Service Description Language

Executive summary

Web Services Policy is a machine-readable language for representing the capabilities and requirements of a Web service (Security, QoS…). Policies are about expressing behavioural qualities, so they can range dramatically in size and in the nature of the policy content. Additionally, the flexibility and extensibility built in to the WS-Policy language allows its few elements and attributes to be combined into a variety of complex designs.

In order to make web service selection easier, it could be useful to add semantic concepts to policy descriptions. This deliverable presents an extension of WS-Policy specification, which allows users to add semantic annotations. This new specification, called SA-Policy enables semantic reasoning on policies in order to compare customer requirements and provider capabilities, particularly in complex cases.

1. Introduction

1.1 Introductory explanation of the deliverable

SOA4All aims at realizing a world where billions of parties are exposing and consuming services via advanced Web technology: the main objective of the project is to provide a comprehensive framework that integrates complementary and evolutionary technical advances (i.e., SOA, context management, Web principles, Web 2.0 and semantic technologies) into a coherent and domain-independent service delivery platform.

In order to facilitate selection of most relevant services, this project proposes some mechanisms based on semantic technology. This deliverable is a part of it and deals with possible semantic contributions in Web Services Policy. It proposes to define semantic annotation mechanism in policies and develops an example to explain possible uses of this new extension.

1.2 Purpose and Scope

1.2.1 Purpose

This deliverable defines a set of extensions of the WS-Policy Framework (Semantic Annotations for Policy – SA-Policy) in order to support semantic descriptions of nonfunctional properties of services as well as business rules and user profiles on top of syntactical Web service standards. These extensions will constitute a candidate for standardization as part of WP11 efforts.

1.2.2 Audience

This document is intended as a reference of the SA-Policy extension. In turn its main audience are users who want to add semantic descriptions of nonfunctional properties of services and require a precise specification. Even more so it targets technical staff building tools (i.e. semantic annotation handlers, matchmaking engines…).

Inside the consortium, this mainly applies to partners involved in technical work packages within Activity cluster A1 – Fundamentals Integration Activities (for annotation tools), A2 – Core R&D Activities (for SA-Policy uses) and A3 – Use Case Activities. For outside parties beyond the consortium it can serve as an introduction to SA-Policy.

1.3 Structure of the document

The remainder of this deliverable is structured as follows: Section 2 presents an overview of WS-Policy (goals, mechanisms, uses...). Section 3 defines a new semantic annotation mechanism based on SAWSDL annotation and applied to Web Service Policies. It describes new attributes and interactions with other specifications (WS-Policy, WS-Policy Attachment...). Section 4 introduces an example of SA-Policy using basic policy ontology coupled with user additions in order to make web service selections.

2. WS-Policy overview

2.1 Policy structure

2.1.1 Policy definition

Web Services Policy is a machine-readable language for representing the capabilities and requirements of a Web service (Security, QoS…). Web Services Policy offers mechanisms to represent consistent combinations of capabilities and requirements, to determine the compatibility of policies, to name and reference policies and to associate policies with Web service metadata constructs such as service, endpoint and operation.

The WS-Policy vocabulary is relatively simple. It contains only four main elements - `Policy`, `All`, `ExactlyOne` and `PolicyReference` - and two attributes - `wsp:Optional` and `wsp:Ignorable`. However, policies introduce unique structural considerations that differ from the simple technical interface focus of WSDL and XML Schema. Because policies are about expressing behavioural qualities, they can range dramatically in size and in the nature of the policy content. Additionally, the flexibility and extensibility built in to the WS-Policy language allows its few elements and attributes to be combined into a variety of complex designs.

There is some basic terminology:

- The formal term for a policy is *policy expression*.
- A policy expression can be comprised of one or more elements that express specific policy requirements or properties. Each of these is called a *policy assertion*.
- In order to group policy assertions, we use a set of features from the WS-Policy language known as *policy operators*.
- Policy expressions can optionally be isolated into a separate document, referred to as a *WS-Policy definition*.

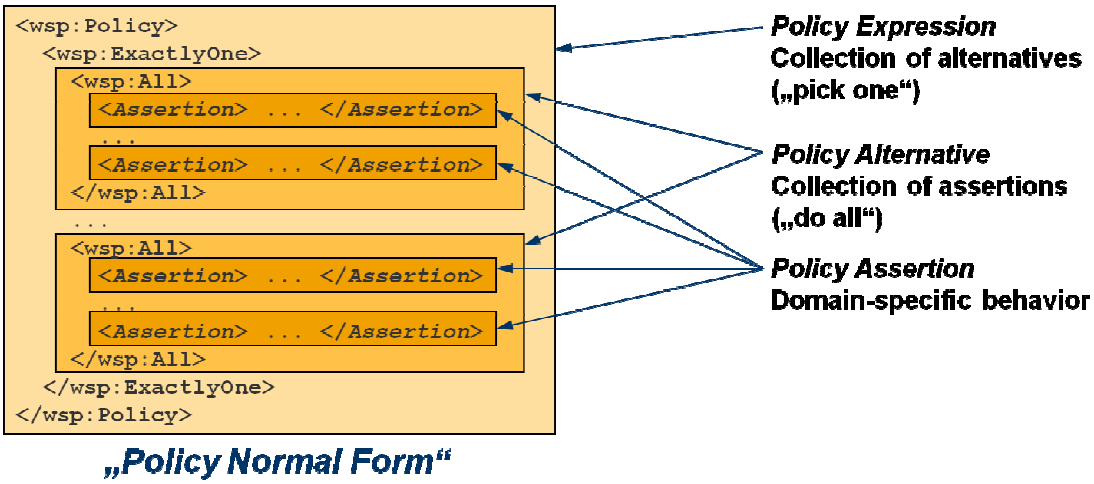

Figure 1: WS-Policy Information Model [1]

2.1.2 A small example

Let us suppose we want to build a client application for a travel agency. This application has to propose to customers to reserve both flight and hotel. We found available Web Services, which meet our needs, but they require the use of addressing headers for messaging. Just the WSDL description is not sufficient for us to enable the interaction between our client and these Web services. WSDL constructs do not indicate requirements such as the use of addressing.

Providers have the option to convey requirements, such as the use of addressing, through word-of-mouth and documentation (as they always have). To interact successfully with this service, we may have to read any related documentation or look at sample SOAP messages and infer such requirements or behaviours.

Here is an example of an expected SOAP message:

```
<soap:Envelope>
     <soap:Header>
          <wsa:To>http://example.soa4all.eu/HotelService</wsa:To>
          <wsa:Action>http://example.soa4all.eu/reserveHotel</wsa:Action>
     </soap:Header>
     <soap:Body>...</soap:Body>
</soap:Envelope>
```

Code 1: Simple SOAP message

This message uses message-addressing headers. The `wsa:To` and `wsa:Action` header blocks identify respectively the destination and the operation.

Policy makes it possible for providers to represent such capabilities and requirements in a machine-readable form. For example, hotel reservation's provider may augment the service WSDL description with a policy that requires the use of addressing. Then, we can use a policy-aware client that understands this policy and engages addressing automatically.

The example below illustrates a policy expression that requires the use of addressing.

```
<wsp:Policy>
     <wsaw:UsingAddressing/>
</wsp:Policy>
```

Code 2: Simple Policy Example

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

The `wsaw:UsingAddressing` element is a policy assertion. This assertion identifies the use of Web Services Addressing information headers. A policy-aware client can recognize this policy assertion, engage addressing automatically, and use headers such as `wsa:To` and `wsa:Action` in SOAP Envelopes.

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

2.2 Policy assertions

A policy assertion is a piece of service metadata, and it identifies a domain (such as messaging, security, reliability and transaction) specific behaviour that is a requirement.

As you can see in the previous example, Policy tag contains an assertion, which is apart from the WS-Policy Specification. A policy assertion is a piece of service metadata, and it identifies a domain (such as messaging, security, reliability and transaction) specific behaviour that is a requirement. WS-Policy just allows users to bind Web Services with policies assertions defined in other specifications. Policy assertions are defined by several WS-* protocol specifications and applications:

- WS-Addressing Metadata
- WS-Addressing WSDL Binding
- WS-Atomic Transaction
- WS-Business Activity
- WS-Message Transmission Optimization Mechanism (MTOM) Serialisation Policy
- WS-Security Policy
- WS-Reliable Messaging Policy
- ...

Each specification defines new assertions, its significations as Policy uses and possible interactions/interweaving of assertions.

2.3 Assertion composition

2.3.1 Policy operators

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

2.3.1.1 The `All` Element

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

```
<wsp:All>
     <wsaw:UsingAddressing/>
     <wsrmp:RMAssertion/>
</wsp:All>
```

Code 3: Policy Composition – `All` Element

In this case, the provider indicates the service uses of WS-Addressing and WS-Reliability Messaging Protocol (without more details). Involvements brought by the two assertions are explained in its specifications. The `All` element can be seen as the Boolean operator AND.

2.3.1.2 The `ExactlyOne` Element

This element groups a set of policy assertions from which only one can be used. Using the `wsp:ExactlyOne` element introduces the concept of policy alternatives as part of a policy expression. Each child element within this construct is considered as distinct alternative in the overall policy expression.

```
<wsp:ExactlyOne>
     <wsaw:UsingAddressing/>
     <wsrmp:RMAssertion/>
</wsp: ExactlyOne>
```

Code 4: Policy Composition – `ExactlyOne` Element

In this case, the provider indicates you can use this service with WS-Addressing or WS-Reliability Messaging Protocol (but not with both). The `wsp:ExactlyOne` element can be seen as the Boolean operator XOR (Exclusive OR).

2.3.1.3 The `optional` Attribute

The WS-Policy language provides a special Boolean attribute named `wsp:optional` that allows you to indicate that a policy assertion is not mandatory. The following example shows this attribute used in conjunction with our previous `wsaw:UsingAddressing` assertion:

```
<wsp:Policy>
     <wsaw:UsingAddressing wsp:optional="true"/>
</wsp:Policy>
```

Code 5: Policy Composition – `optional` Attribute

You can express the same policy logic in a more verbose manner using the previously described operator elements. The `wsp:All` element is essentially the same as specifying two distinct alternatives in a policy expression, as follows:

```
<wsp:Policy>
     <wsp:ExactlyOne>
          <wsp:All>
               <wsaw:UsingAddressing/>
          </wsp:All>
          <wsp:All/>
     </wsp:ExactlyOne>
</wsp:Policy>
```

Code 6: Equivalence of the `optional` Attribute

In this example, we have a `wsp:ExactlyOne` construct that houses two `wsp:All` elements. The second `wsp:All` element is highlighted. Unlike the first, which establishes a construct with one assertion, this second `wsp:All` element is empty.

2.3.2 Example

There is an example of Security Policy use:

```
<wsp:Policy>
     <sp:TransportBinding>
          <wsp:All>
               <sp:AlgorithmSuite>
                    <wsp:Policy>
                         <wsp:ExactlyOne>
                              <sp:Basic256Rsa15/>
                              <sp:TripleDesRsa15/>
                         </wsp:ExactlyOne>
                    </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:TransportToken>
                    <wsp:Policy>
                         <sp:HttpsToken/>
                    </wsp:Policy>
               </sp:TransportToken>
          </wsp:All>
     </sp:TransportBinding>
</wsp:Policy>
```

Code 7: Security Policy example [2]

In order to understand this policy, we have to refer to [3]. In this case, we notice the web service supports HTTPS protocol in order to exchange messages. It also precise user can encrypts messages using two algorithms: Basic256Rsa15 or TripleDesRsa15 (both define in [2]).

There is more than one way to express the same policy. We also could switch Policy operators around in order to split transport binding policy in two smaller within each one possible algorithm.

2.4 Policy attachment

WS-Policy defines mechanisms for associating policies with various XML web service entities (WSDL part, Schema...). In one hand, it defines attributes that allow users to identify policies (using a Qualified Name). In other hand, it makes available some policy reference mechanisms which enables to bind XML Elements and identified Policies.

2.4.1 Policy Identification

WS-Policy specification proposes three attributes within `wsp:Policy` tag for identification:

- `Name`: Identify the policy with an absolute URI, independent of the document.
- `wsu:Id` or `xml:id`: Identify the policy thanks to an ID, unique in the document. Document target namespace must be added to form the QName.

The following example illustrates both methods (considering `wsu:Id` and `xml:id` are quite similar):

```
<!-- Identification by Name -->
```

```
<wsp:Policy Name="http://ewample.soa4all.eu/policy/ExamplePolicy">
     ...
</wsp:Policy>

<!-- Identification by Id -->
<wsp:Policy wsu:Id="ExamplePolicy">
     ...
</wsp:Policy>
```

Code 8: Policy identification

2.4.2 Policy reference

In order to reuse Policy Expressions and attach them to other XML Elements (XSD Schemas, WSDL...), WS-Policy specification defines two way to refer policies.

2.4.2.1 Internal Policy reference

The first mechanism enables to embed policy reference directly in the XML codes. The specification defines an attribute (`wsp:PolicyURIs`) and a tag (`wsp:PolicyReference`), both used to add URI(s), which refers to identified policies.

There are examples of XML elements using embedded Policy Reference:

```
<anyElement wsp:PolicyURIs="http://example.soa4all.eu/policies#ExamplePolicy
                            http://example.soa4all.eu/policies#ExamplePolicy2..."/>
```

Code 9: Policy attachment with `wsp:PolicyURIs` attribute

```
<anyElement>
     <wsp:PolicyReference URI="http://www.example.com/policies#ExamplePolicy"/>
     <wsp:PolicyReference URI="http://www.example.com/policies#ExamplePolicy2"/>
<anyElement/>
```

Code 10: Policy attachment with `wsp:PolicyReference` tag

Both possibilities are similar. Users can choose anyone according to limitations of targeted XML element. For example, the `wsdl:operation` tag (in WSDL 1.1) don't accept new attributes so we have to use `wsp:PolicyReference` tag to link policies.

Policy references also can be used in other policies expressions in order to re-use and combine them:

```
<wsp:Policy>
    <wsp:ExactlyOne>
        <wsp:PolicyReference URI="http://www.example.com/policies#ExamplePolicy"/>
        <wsp:PolicyReference URI="http://www.example.com/policies#ExamplePolicy2"/>
    </wsp:ExactlyOne>
</wsp:Policy>
```

Code 11: Example using `wsp:PolicyReference`

2.4.2.2 External Policy reference

The second reference mechanism takes the opposite point of view and enables to attach list

of targeted elements with the policy description. The `wsp:PolicyAttachement` tag embeds classical Policy Expressions and a new tag (named wsp:AppliesTo) which contains references to external documents:

```
<wsp:PolicyAttachment>
     <wsp:AppliesTo>
          <!—- External reference to a specific XML Element -->
          <anyElement/>

          <!-- External reference to a domain URI -->
          <wsp:URI>http://example.soa4all.eu/definedURI</wsp:URI>

          <!-- External reference to an endpoint address -->
          <wsa:EnpointReference>
               <wsa:Address>http://example.soa4all.eu/serviceEndpoint</wsa:Address>
          </wsa:EndpointReference>
          ...
     </wsp:AppliesTo>
     <wsp:Policy>
          ...
     </wsp:Policy>
</wsp:PolicyAttachment>
```

Code 12: External Policy attachment

The `wsp:AppliesTo` tag can refer to:

- A specific XML Element,
- A domain URI (using `wsp:URI` tag) or
- A deployed service endpoint using WS-Addressing `wsa:EndpointReference` tag.

3. The specification of the deliverable

As we saw in the previous section, there is more than one way to express policies and it can be hard to understand and compare them. Even if the WS-Policy is simple, there are a lot of possible assertions and combinations. In order to make web service selection easier, it could be useful to add semantic concepts to policy descriptions. This section defines extension attributes for WS-Policy definition language that allows reference to semantic models.

3.1 SA-Policy Model Reference

In order to facilitate adoption and use of this new extension, we try to define our extension attribute staying close to SAWSDL annotation mechanism. SA-Policy Model reference can be used in every WS-Policy tag including assertions.

The SA-Policy modelReference own the same XML Schema as SAWSDL modelReference with a different namespace in order to use them at the same time:

```
<xs:attribute name="modelReference" type="listOfAnyURI"/>

<xs:simpleType name="listOfAnyURI">
     <xs:list itemType="xs:anyURI"/>
</xs:simpleType>
```

Code 13: SA-Policy `sawsp:modelReference` XML Schema

The value of `sawsp:modelReference` is a set of zero or more URIs, separated with whitespaces, that identify semantic concepts. Each URI is a pointer to a concept in a semantic model and is intended to provide semantic information about the Policy component being annotated. This specification does not define logical relationships between multiple URIs in a same `modelReference` attribute.

Identified concepts could be written in any semantic representation languages such as RDF [4], RDF-S [5], OWL [6] or WSML [7] (e.g. embedded in wsl:nonFunctionalParameter of WSMO-Lite description). It only requires that the semantic concepts defined in it be identifiable via URI references.

This document does not describe how to use these semantic annotations. It only provides an outline and hints in section 4.

3.2 Annotating Policy description

There are many Policy assertions and future specifications could define new ones in order to express new requirements or capabilities. Thanks to these specifications, each assertion is clearly defined and users can refer to the assertion definition to understand it. Doubts about policy matching are mainly brought by policy expressions (groups of assertions) and by different ways to express one combination of policies.

The WS-Policy mechanism combines assertions surrounding them by only three different tags, which are in the middle of our target. So, the main use of `sawsp:modelReference` concerns these three tags:

The schema outline for the `sawsp:modelReference` attribute in the compact form is as follows:

```
<wsp:Policy (sawsp:modelReference=”xs:anyURI*”)?…>
     ( <wsp:Policy (sawsp:modelReference=”xs:anyURI*”)?…>…</wsp:Policy> |
     <wsp:ExactlyOne (sawsp:modelReference=”xs:anyURI*”)?…>…</wsp:ExactlyOne> |
     <wsp:All (sawsp:modelReference=”xs:anyURI*”)?…>…</wsp:All> )*
</wsp:Policy>
```

Code 14: Compact schema of semantic annotation for WS-Policy

The following describes the three attributes defined in the schema outline above:

`/wsp:Policy/@sawsp:modelReference`

`/wsp:All/@sawsp:modelReference`

`/wsp:ExactlyOne/@sawsp:modelReference`

This attribute allows users to reference a semantic concept using a set of zero or more URIs separated with whitespaces.

The following example illustrates the use of semantic annotations on Policies (Code 7):

```
<wsp:Policy
sawsp:modelReference=”http://example.soa4all.eu/onto.owl#MyTransportBindingPolicy”>
     <sp:TransportBinding>
          <wsp:All
sawsp:modelReference=”http://example.soa4all.eu/onto.owl#MyHttpsAlgorithmsPolicy”>
               <sp:AlgorithmSuite>
                    <wsp:Policy>
                         <wsp:ExactlyOne
sawsp:modelReference=”http://example.soa4all.eu/onto.owl#MyAlgorithmChoicePolicy”>
                              <sp:Basic256Rsa15/>
                              <sp:TripleDesRsa15/>
                         </wsp:ExactlyOne>
                    </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:TransportToken>
                    <wsp:Policy>
                         <sp:HttpsToken/>
                    </wsp:Policy>
               </sp:TransportToken>
          </wsp:All>
     </sp:TransportBinding>
</wsp:Policy>
```

Code 15: Semantic annotation in Policy expression using `sawsp:modelReference`

If the policy description cannot be annotated, the `sawsp:modelReference` can be extended to Policy reference mechanisms as following:

`/wsp:PolicyReference/@sawsp:modelReference`

`/{any}/@sawsp:modelReference`

In the last case, the sawsp:modelReference attribute must be used at the same time. This case is problematic because of multiple possible URIs in both attributes. Avoid it if possible:

```
<XMLElement wsp:PolicyURIs=http://example.soa4all.eu/policies#ExamplePolicy
sawsp:modelReference=”http://example.soa4all.eu/onto.owl#MyPolicy”/>
```

Code 16: Semantic annotation in Policy reference

3.3 Embedding semantic models in policies

The URIs used in the `sawsp:modelReference` attribute typically refer to concepts in a semantic model that is external to the Policy description. However, the URIs can also refer to elements within the Policy description if semantic information is included in the document via the Policy extension element as shown in the following example:

```
<wsp:Policy>
     <rdf:RDF xml:base="http://example.soa4all.eu/onto">
          <owl:Class rdf:ID="MyClass"/>
     </rdf:RDF>
     <wsp:Policy sawsp:modelReference="http://example.soa4all.eu/onto#MyClass"/>
     ...
</wsp:Policy>
```

Code 17: Embedding Semantic Models in Policy

WS-Policy already allows extension elements within wsp:Policy element so SA-Policy does not define an additional container. This example illustrates the adding of OWL/RDF-S description but any semantic language could be added in a new tag within `wsp:Policy`.

4. Illustration

Let us look at an example in order to illustrate semantic annotation contribution in web service selection/ranking.

4.1 Example presentation

We still want to build a travel agency client application and we try to find web services, which meet our needs. Thanks to WSMO-Lite descriptions, the matching engine selects potential web services that meet all our functional needs. Then, we could recognise WS-Policy in order to find the fittest service. Unfortunately, no available service proposes exactly our policy requirement so we have to reason on it.

In a classic case (without semantic annotation), we only could check if provider's WS-Policy capabilities perfectly match with consumer requirements: assertions have to be equals. Thanks to semantic annotations and semantic tools (reasoners, matchmakers...), it becomes possible to make service ranking according to expressed relations between assertions or other properties like user preferences. This example focuses on both cited ranking properties and presents a possible ontology enabling representation of WS-Policy assertions and expressions.

Here is the list of selected services:

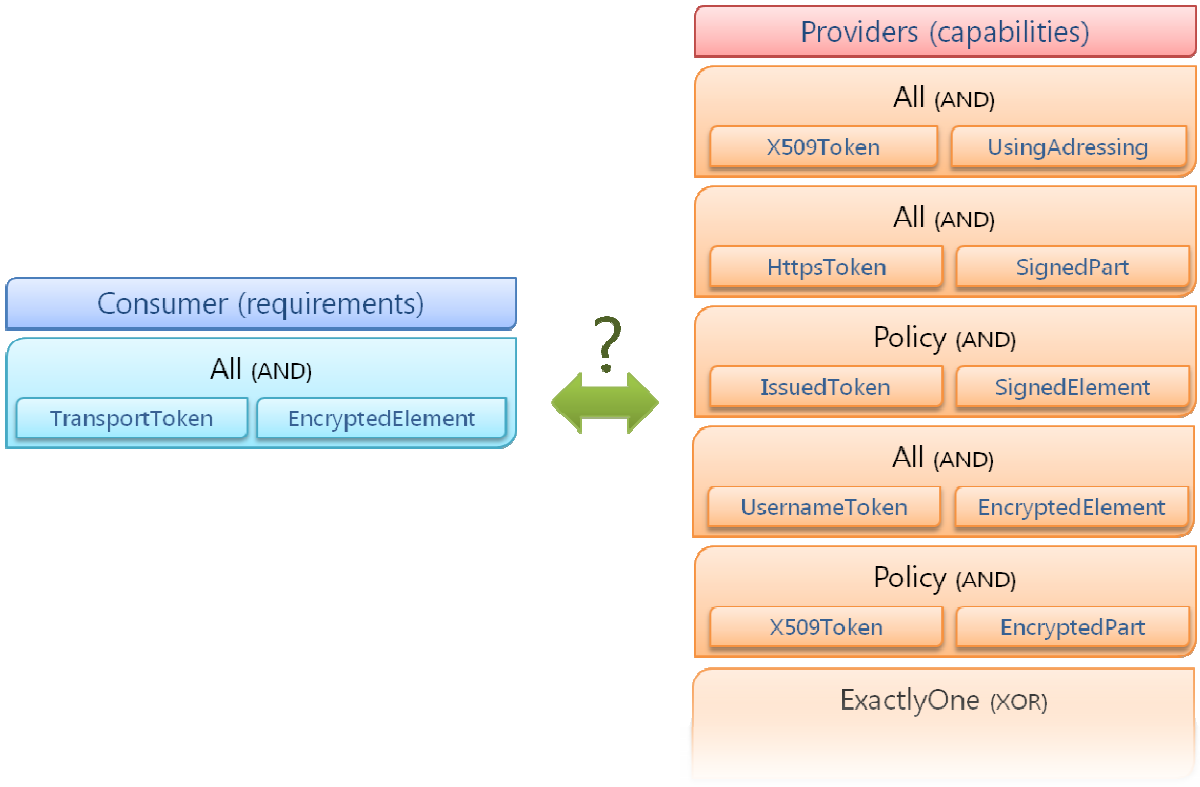

Figure 2: Example of WS-Policy requirements / capabilities

4.2 Policy ontology

In order to annotate them with semantic concepts and reason on it, we have to create usable policy ontology. It has to define possible assertions and allow users to represent his policy expressions and preferences. The following ontology is not exhaustive but clearly shows potential of semantic annotations in WS-Policy.

4.2.1 Ontology structure

Semantic concepts are divided in two parts:

- **PolicyAssertion** contains definitions of assertions. Extended concepts represent categories of assertions (according to specifications) and instances represent assertions.
- **PolicyExpression** contains a free space to describe semantically our Policy expressions in order to annotate them in our service description. This part can be directly included in WSMO-Lite description (as nonFunctionalParameter).

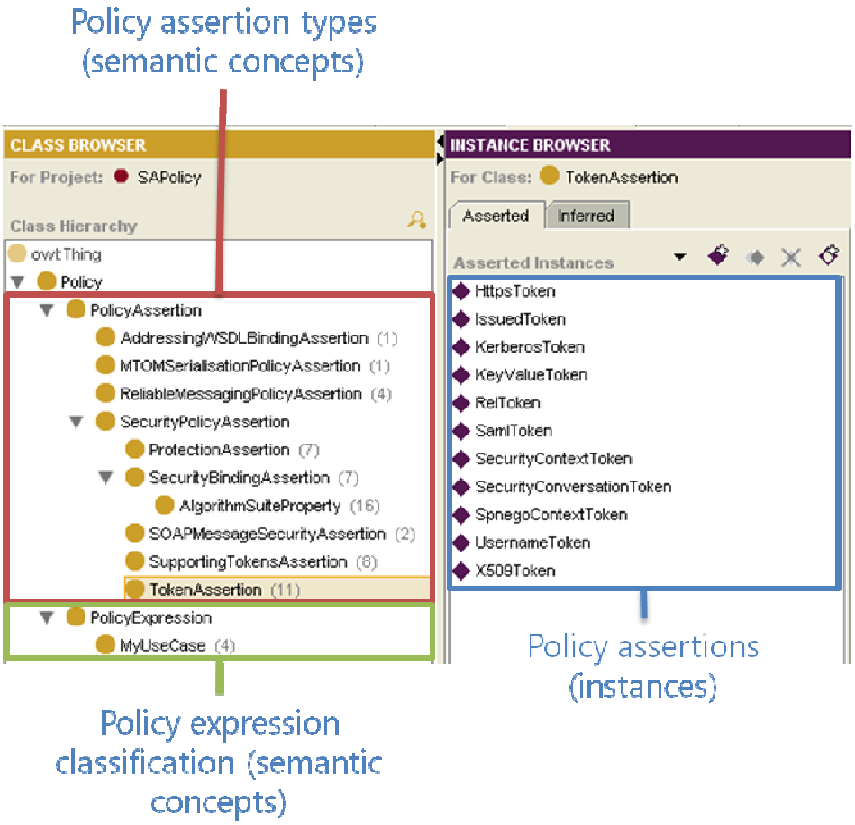

Figure 3: Semantic concepts of our Policy ontology

In this example, the ontology focused on some Security Policy assertions but the principle can be extended to all linked specifications.

On top of that, three types of properties were defined:

- **assertionRelation** contains properties that organize assertions according to specifications:
  - *hasProperty:* indicates a parent/child relation between two assertions.
- **expressionRelation** hold properties which enable to express Policy expressions in PolicyExpression part or in wsl:nonFunctionalParameter in WSMO-Lite:
  - *all* and *exactlyOne:* represent WS-Policy groups defined in the specification. They can be applied to PolicyExpression instances and point at any assertion/expression.
- **userPreference** contains properties that allow users to express preferences:
  - *haveToUseProperty* and *cannotUseProperty*: respectively indicate obligation and the forbidding to use a specific property within the specified assertion/expression.
  - *preferedTo*: represents user preferences between two assertions/expressions. See Figure 5 for example.

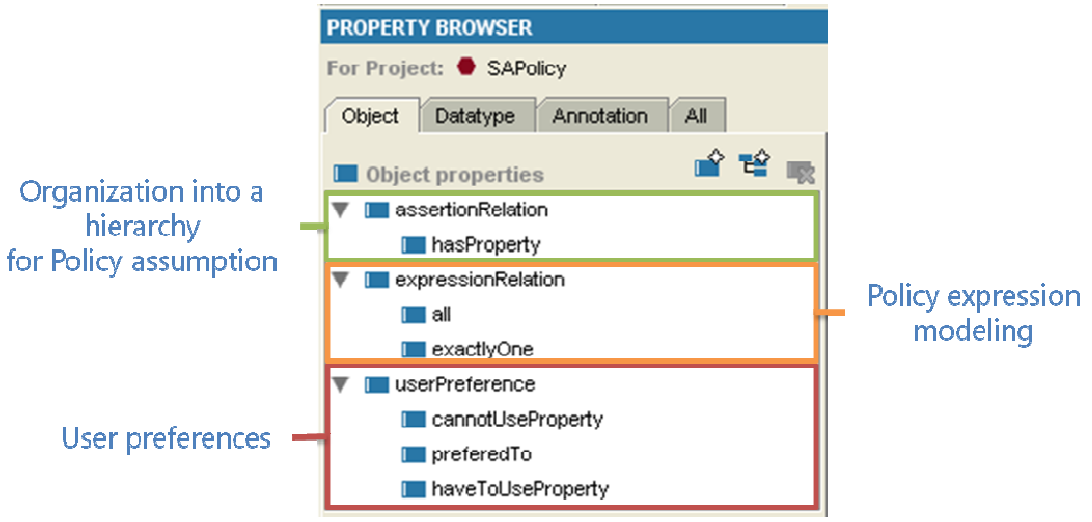

Figure 4: Semantic properties of our Policy ontology

It could be interesting to distinguish two kinds of ontology components (concepts, instances, properties…):

- Specification relative components (PolicyAssertion and assertionRelation) which are available and viable in any use of WS-Policy. These components can be shared between projects.
- Project dependant components (PolicyExpression, expressionRelation and userPreference) that are specific to one reasoning or maybe few ones.

4.2.2 User Policy expressions and preferences

All the assertions are clearly defined in specifications so exact matching does not need semantic reasoning. Contribution of semantic seems to be interesting only for complex expressions or information apart from specifications matching (such as user preferences).

Following figures illustrate some user preferences, which are useful for the example:

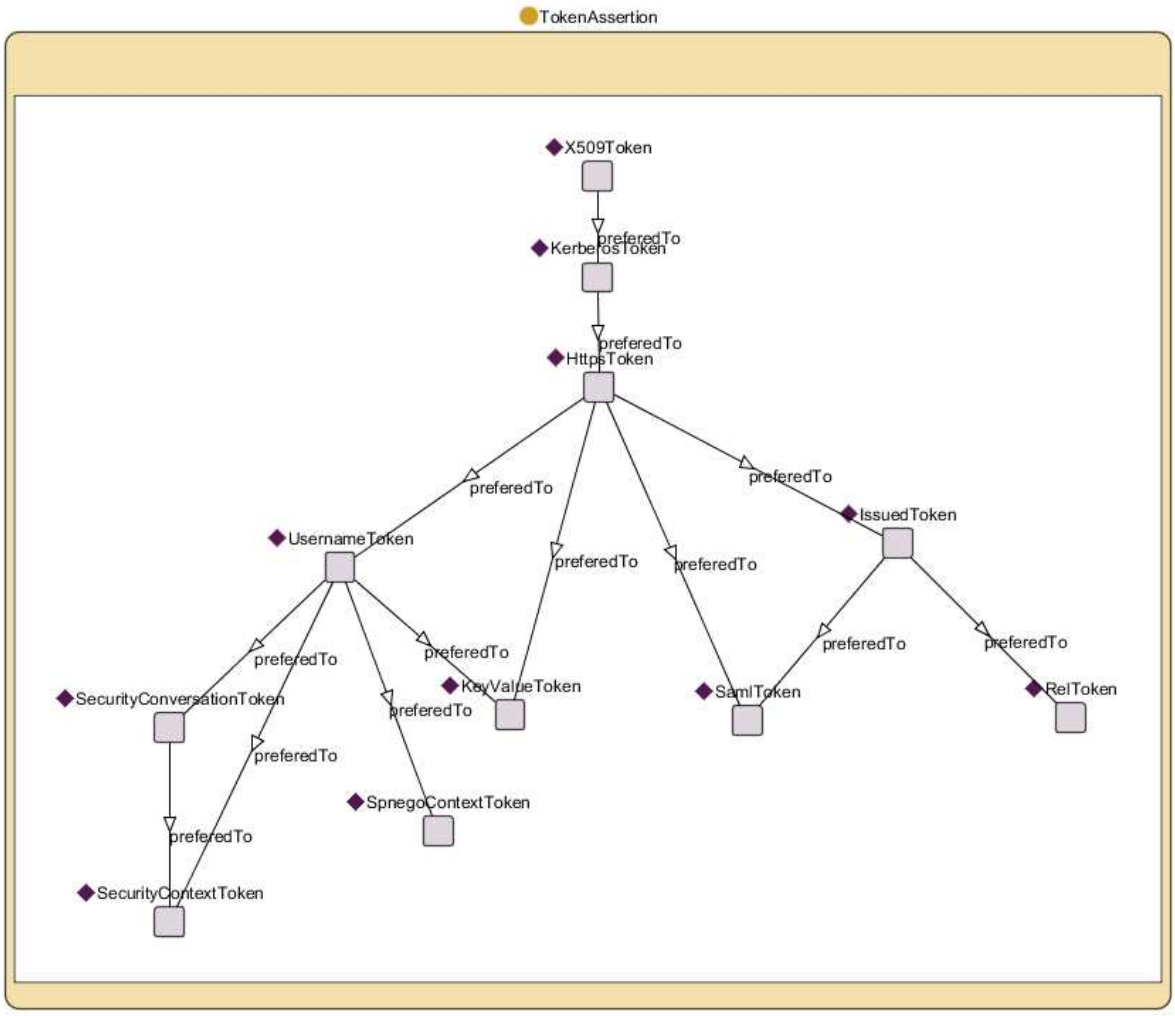

Figure 5: `preferedTo` property between Token assertions

The Figure 5 shows the different token assertions present in SecurityPolicy specification with some `preferedTo` properties put by the user. We regain some provider capabilities such as X509, Https, Username and Issued Tokens.

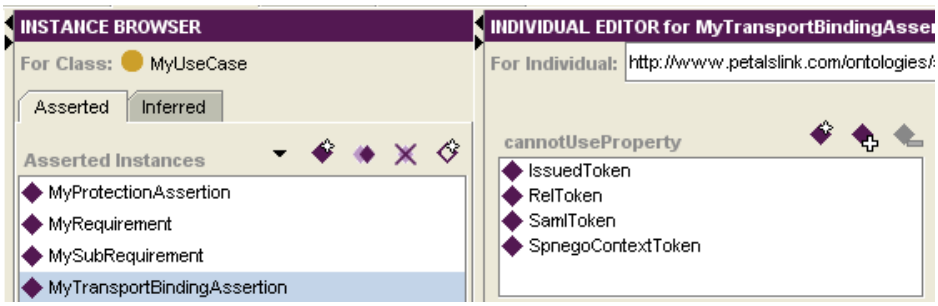

Figure 6: `cannotUseProperty` restriction between expression and assertions

The figure below presents some restrictions imposed by user preferences about MyTransportBindingAssertion, a specific Policy expression. In this expression, we forbid few token assertions (including IssuedToken).

We could define more preferences but it should be sufficient for our example.

4.3 Semantic annotation in policies

This deliverable focuses on semantic annotations in WS-Policy. Like other annotation mechanisms (e.g. SAWSDL), SA-Policy allows to affect ontology concepts/instances to technical description. The mechanism is simple in our case: in the previous part, we expressed our requirements in PolicyExpression part in order to be used as search input. Now, we have to add semantic to web services using SA-Policy.

We first need to express Policy Expression for each service, using previous ontology. We can embed semantic description in WSMO-Lite description or in any other ontology whatever the language. Then, we have to annotate policies with those concepts.

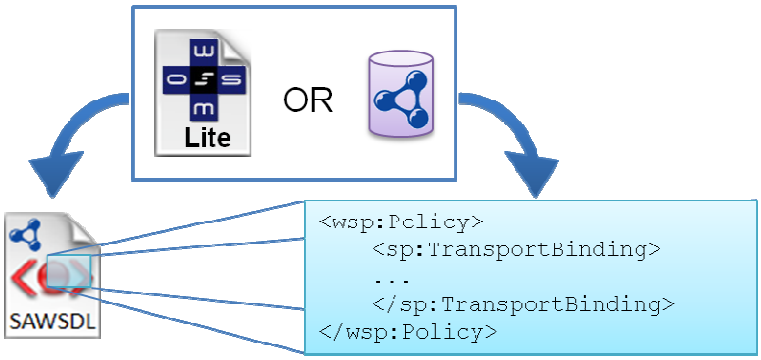

Figure 7: Policy semantic annotations from WSMO-Lite or other ontology

4.4 Service ranking

Finally, we are able to make service selection and ranking thanks to semantic annotations and inferences on the ontology(ies). In this example, we make the matchmaking in two step in order to detail the process.

First, we deduce from user preferences and involved assertions (provider capabilities) an ordered list of possible assertions:

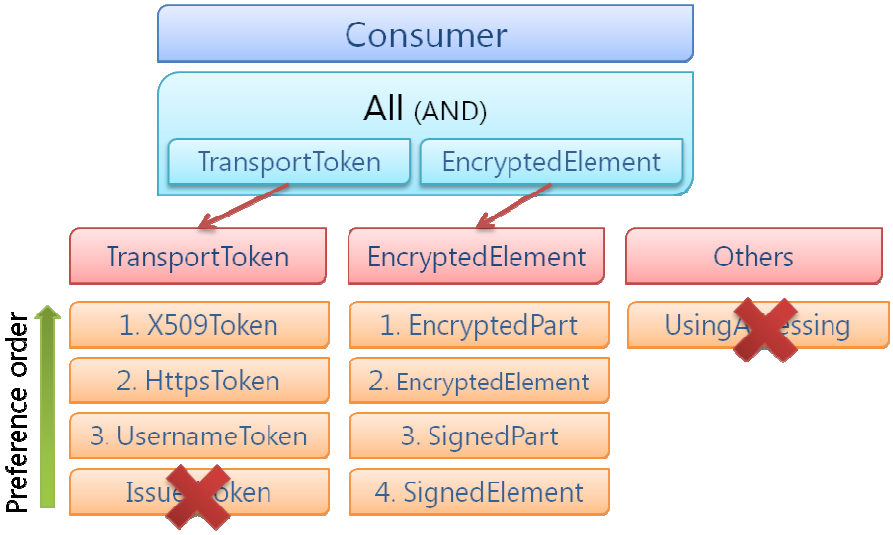

Figure 8: Possible assertions ranking

In the figure below, we regain previous assertions classified according to the ontology (see Figure 5 and Figure 6).

Then, we can use previous list for global ranking of services:

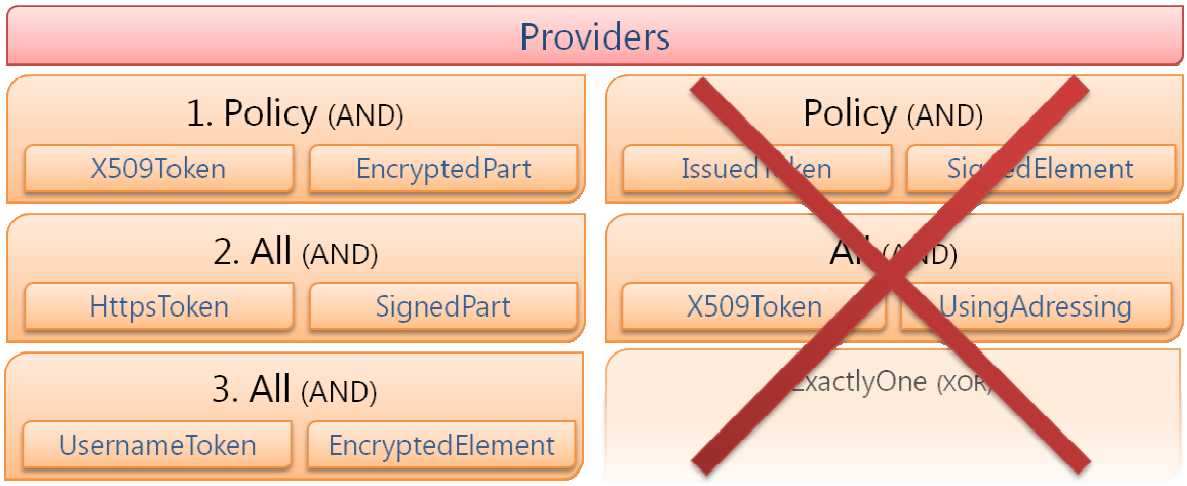

Figure 9: Final service ranking by Policy user preferences

All services which do not meet our needs are removed from possibilities, others are classified by user preferences.

The critical points in this process are conceptions of ontology and compatible matchmaker.

The first step requires semantic expertise in order to create useful, re-usable and exhaustive ontology. Some projects or research groups already work on the connection between Policies and semantics. For example, the MINDSWAP group proposes some tools and ideas to represent WS-Policies in OWL language [8]. Other research groups are interested in web service selection using reasoning around WS-Policy ontologies [9].

The second step needs development of new features in the semantic matchmaker involved in SOA4All project.

5. Conclusions

In this deliverable, we presented an overview of WS-Policy then an extension of this specification, which allows users to add semantic annotations. This new specification, called SA-Policy enables semantic reasoning on policies. It could be useful in order to compare customer requirements and provider capabilities, particularly in complex cases.

6. References

1. von Riegen C., *WS-Policy Overview*, W3C Workshop on Constraints and Capabilities for Web Services, 2004
2. Vedamuthu A., Orchard D., Hirsch F., Hondo M, Yendluri P., Boubez T., Yalçinalp Ü., *Web Services Policy 1.5 – Framework*, W3C Recommendation, 2007: http://www.w3.org/TR/2007/REC-ws-policy-20070904/
3. Lawrence K., Kaler C., Nadalin A., Goodner M., Gudgin M., Barbir A., Granqvist H., *WS-SecurityPolicy 1.2*, OASIS Standard, 2007
4. Manola F., Miller E., *RDF Primer*, W3C Recommendation, 2004: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
5. Brickley D., Guha R.V., *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, 2004: http://www.w3.org/TR/rdf-schema/
6. McGuinness D., van Harmelen F., *OWL Web Ontology Language - Overview*, W3C Recommendation, 2004: http://www.w3.org/TR/owl-features/
7. De Bruijn J., Lausen H., Krummenacher R., Polleres A., Predoiu L., Kifer M., Fensel D., D16.1v0.2 The Web Service Modeling Language WSML, WSMO, 2005: http://www.wsmo.org/TR/d16/d16.1/v0.2/
8. Parsia B., Kolovski V., Hendler J., *Expressing WS Policies Using OWL*, *Policy Management for the Web Workshop, WWW2005, 2005*
9. Sriharee N., Senivongse T., Verma K., Sheth A., *On Using WS-Policy, Ontology and Rule Reasoning to Discover Web Services*, Springer, 2004

7. Annexes

7.1 XML Namespaces

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

Table 1: Prefix and XML Namespaces used in this document

Prefix	XML Namespace	Specifications
owl	http://www.w3.org/2002/07/owl	[OWL]
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns	[RDF]
rdfs	http://www.w3.org/2000/01/rdf-schema	[RDF Schema]
sawsp	http://www.petalslink.com/ns/sa-policy	This document
soap	http://www.w3.org/2003/05/soap-envelope	[SOAP 1.2]
sp	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	[WS-SecurityPolicy]
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing	[WS-Addressing]
wsaw	http://www.w3.org/2006/05/addressing/wsdl	[WS-Addressing WSDL Binding]
wsdl	http://schemas.xmlsoap.org/wsdl/	[WSDL 1.1]
wsdl2	http://www.w3.org/ns/wsdl	[WSDL 2.0]
wsp	http://www.w3.org/ns/ws-policy	[WS-Policy Framework, WS-Policy Attachment]
wsrmp	http://docs.oasis-open.org/ws-rx/wsrmp/200702	[WS-Reliable Messaging Policy]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-Security 2004]
xs	http://www.w3.org/2001/XMLSchema	[XML Schema Structure]