



# D24.3v0.1 Aligning WSMO with WS-Policy

WSMO Working Draft 16 February 2007

**This version:**

<http://www.wsmo.org/TR/d24/d24.3/v0.1/20070216/>

**Latest version:**

<http://www.wsmo.org/TR/d24/d24.3/v0.1/>

**Previous version:**

<http://www.wsmo.org/2005/d24/d24.1/v0.1/20070112/>

**Authors:**

Jacek Kopecký  
Dumitru Roman

This document is also available in non-normative [PDF](#) version.

Copyright © 2007 DERI®, All Rights Reserved. DERI liability, trademark, document use, and software licensing rules apply.

---

## Table of contents

- 1. Introduction
  - 2. WS-Policy Overview
  - 3. Combining Policy with SESA and WSMO
    - 3.1 Attaching Policies to Semantic Descriptions in WSMO
    - 3.2 WSMO Assertions in WS-Policy
    - 3.3 Policy in SEE components
    - 3.4 Policy Tasks in SEE
  - 4. Related Work
  - 5. Conclusions and Further Work
  - References
  - Acknowledgement
  - Change Log
- 

## 1. Introduction

The research areas of Semantically Enabled Service-oriented Architectures (SESA) and Semantic Web Services (SWS) aim to increase the level of automation of some tasks commonly performed in service-oriented systems (e.g. discovering available services and composing them to provide more complex functionalities) using Semantic Web technologies, i.e. ontologies and reasoning. Among the tasks that can be automated in SESA are policy matching and enforcement.

A *policy* is a set of non-functional constraints and capabilities, for instance the set of authentication protocols supported or required by some system. In traditional distributed systems (e.g. most of those existing before the era of Web services), policies are hard-coded into the system according to the functional requirements, language features, and design decisions, without separating policy specification from policy implementation. This makes it difficult and expensive to change and match policies. Policy specification languages like WS-Policy enable policies to be captured independent from a concrete system implementation. In such a setting, policies are more flexible, reusable, verifiable, extensible and efficient. Policy languages are to be interpreted by a policy engine at runtime, which makes dynamic policy changes possible; they formalize the intent of the designer into a document that can be analyzed and interpreted by a policy-aware system.

Please note that while WS-Policy is defined as capturing non-functional constraints and capabilities, the latter is very different from WSMO Web service capabilities. In WSMO, a Web service capability describes semantically what the Web services does, what is its purpose and function. In WS-Policy, on the other hand, policy assertions may indicate other, non-functional capabilities. For example, the functional capability of a travel service is to reserve hotels, whereas a non-functional capability

(policy) of the service may be that it gives final reservations within 24h of the request and that it supports strong encryption for communication.

In this deliverable we focus on the role of explicit policy specification in semantically enriched service-oriented environments, like those represented by WSMO and WSMX. Policy specification and evaluation frameworks can be integrated into Semantic Web Services systems in many ways. For example, Uszok et al. describe in [Uszok2004b] how they use their system called KAoS as an authorization service in a Grid setting, as a policy specification and enforcement system in a semantic matchmaker, and for verification of compositions and contracts. We do not tackle in this deliverable the question of how semantic technologies can help in policy frameworks, even though in Section 5 we shortly touch on how semantic policy expression would interplay with our analysis of how and where policies can be taken into account in SESA.

For the purpose of this deliverable, we have chosen to use the Web Services Policy Framework (WS-Policy) [WSPolicy], a general purpose model and a syntax to describe the policies of a Web Service. Our choice is motivated by the fact that WS-Policy has significant industry backing. Interested parties are already creating policy assertions for various domains and the major commercial Web services infrastructure stacks already have or are building support for policy-based Web service invocations.

Web Services Policy Framework allows describing specific Web services metadata (so called policies) used for expressing the capabilities, requirements, and general characteristics of Web services. WS-Policy is a specification with significant industry backing. Interested parties are already creating policy assertions for various domains and the major commercial Web services infrastructure stacks already have or are building support for policy-based Web service invocations.

The aim of this deliverable is to investigate the potential relations between SESA and WSMO and the WS-Policy framework, and to offer a basis for further research on the use of policies in the context of semantically enriched service environments. Issues like policy conformance checking, policy-based semi-automatic negotiations between users and services, or policy-based matchmaking etc., are important in our context, however they are out of scope of this deliverable. The scope of this deliverable is limited to identifying the uses of WS-Policy in SESA, and proposing a concrete way for combining them.

The following section, i.e. Section 2 presents a brief overview of WS-Policy specifications. The concrete contributions of this section are presented in Section 3, and in short they are: the syntax for attaching WS-Policy policies to WSMX components, the syntax for putting WSMX descriptions as policy assertions inside policy alternatives, and the analysis of what WSMX/SEE (Semantic Execution Environment) components can include policy evaluation and for what tasks. Finally Section 4 talks about related work.

## 2. WS-Policy Overview

In this section we give a brief overview of the Web Services Policy Framework ([WS-Policy]).

The Web Services Policy Framework provides a general purpose model and a syntax to describe the policies of a Web Service. The key feature of WS-Policy is its flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. In this context, WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements and capabilities.

At the core of WS-Policy stands the concept of *policy expression* which contains domain-specific, Web Service policy information. WS-Policy further defines a core set of constructs to indicate how choices and/or combinations of domain-specific policy assertions apply in a Web services environment. In the following we shortly describe the concepts of WS-Policy and the way they are related.

**Policy.** At the abstract level a policy is a potentially empty collection of *policy alternatives*. Alternatives are not ordered. They may differ significantly in terms of the behaviors they indicate, as well as conversely, they may be very similar. In either case, the value or suitability of an alternative is generally a function of the semantics of assertions within the alternative and is therefore beyond the scope of WS-Policy.

**Policy Alternative.** A policy alternative is a logical construct which represents a potentially empty collection of *policy assertions*, all of which apply in conjunction. The *vocabulary* of a policy alternative is the set of all assertion types within the alternative. The vocabulary of a policy is the set of all assertion types used in all the policy alternatives in the policy. Assertions that do not appear in an alternative, but whose type is in the vocabulary of the whole policy, are implied to be forbidden by the

alternative.

**Policy Assertion.** A policy assertion identifies a behavior that is a requirement (or capability) of a policy subject. Assertions indicate domain-specific (e.g., security, transactions) semantics and are expected to be defined in separate, domain-specific specifications.

Additionally, WS-Policy defines the following terms:

**Policy Assertion Type**

represents a class of policy assertions and implies the structure of the assertion and its assertion-specific semantics.

**Policy Assertion Parameter**

qualifies the behavior indicated by a policy assertion.

**Policy Vocabulary**

of a policy is the set of all policy assertion types used in the policy.

**Policy Subject**

is an entity (e.g., an endpoint, message, resource, interaction) with which a policy can be associated.

**Policy Scope**

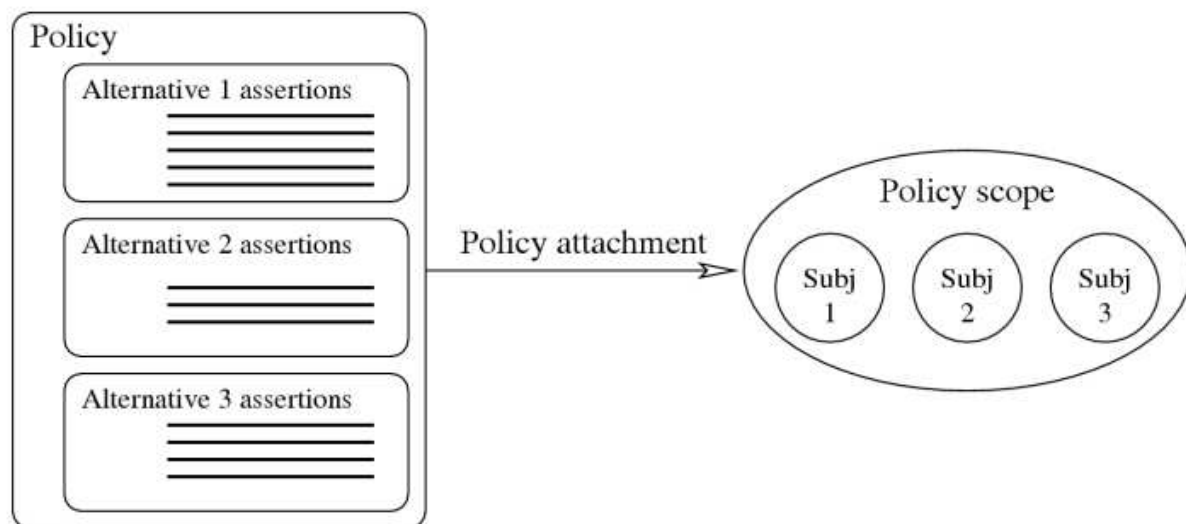
is a collection of policy subjects to which a policy may apply.

**Policy Attachment**

is a mechanism for associating policy with one or more policy scopes.

The structure of WS-Policy, with assertions, alternatives and policy subjects, is shown in Figure 1.

Figure 1: WS-Policy building blocks



Syntactically, WS-Policy offers several ways of expressing policies, but all are equivalent to a single set of alternatives containing only atomic assertions. For example, the following listing shows a policy requiring the use of Kerberos or X509 security tokens.

```
01 <wsp:Policy>
02   <wsp:ExactlyOne>
03     <wsse:SecurityToken>
04       <wsse:TokenType>
05         wsse:Kerberosv5TGT
06       </wsse:TokenType>
07     </wsse:SecurityToken>
08     <wsse:SecurityToken>
09       <wsse:TokenType>
10         wsse:X509v3
11       </wsse:TokenType>
12     </wsse:SecurityToken>
13   </wsp:ExactlyOne>
14 </wsp:Policy>
```

WS-Policy Framework is accompanied by further specifications:

- WS-Policy Attachment [WSPolicyAttachment] specifies two general-purpose mechanisms for associating Policies with one or more policy subjects

- WS-Policy Assertions [WSPolicyAssertions] defines several general-use policy assertions
- Web Services Security Policy Language [WSSecurityPolicy] specifies policy assertions regarding the use of Web Services Security [WSecurity]

WS-Policy is meant for "expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system". The distinction between "entity" and "general characteristic" is not defined in WS-Policy. WSDL descriptions of Web services are an example of the entities to which policies are attached (see Section 4 of WS-Policy Attachment) even while they could also be assertions about "the capabilities, requirements and general characteristics" from the above definition; this split indicates that not all capabilities, requirements and general characteristics qualify as policies.

### 3. Combining Policy with SESA and WSMO

In this section, we intend to show how policies fit into WSMO and SESA.

In the first two subsections, we define the syntactical constructs necessary to bring WS-Policy together with WSMO, the formal language for WSMO. Despite the differences in terminology, we can say that both WSMO and WS-Policy describe the capabilities and constraints of Web services. On one side, WSMO uses very clearly defined terms like *Web service*, *Capability*, *Interface* etc. to capture all the relevant properties of Web services. On the other side, WS-Policy talks about generic assertions, focusing on their combinations in whole policies, and then attaching these policies to various subjects, among which Web service endpoints are especially relevant for our work.

We can identify two generic concepts from the WS-Policy framework that could encompass elements from WSMO: Policy Subject and Policy Assertion. In other words, a policy can be attached to parts of a WSMO description, or parts of a WSMO description could be mentioned as assertions in some policy. We describe these two directions and the syntax realizing the actual connections in the sections 3.1 and 3.2.

Further in Section 3.3 we examine the components of WSMX/SEE (Semantic Execution Environment), in terms of whether their function should include policy manipulation of some sorts, and how. The policy-related tasks that the various components perform, are summarized in Section 3.4.

#### 3.1 Attaching Policies to Semantic Descriptions in WSMO

WS-Policy defines an attachment mechanism that specifies how policies can be attached to WSDL service descriptions. WSMO uses WSDL for *grounding* (see WSMO deliverable D24.2), therefore policies present in a WSDL description will have effect also on Web services specified semantically in a WSMO document that grounds to that WSDL description. However, WSMO need not be grounded only in WSDL (we envision also grounding to Triple Spaces, for example), and grounding is taken into account only in choreography, whereas policy can affect steps before that, therefore we introduce below a way to attach policies directly to WSMO constructs.

To analyze how policies can be attached to WSMO, we need first to introduce a distinction between functional and non-functional properties. In any application, the *functional* part of any data contains crucial information necessary for the application to do its job. *Non-functional* properties (NFPs), on the other hand, contain such additional data that may help the application do a better job, or to refine its functionality.

For example, one of the aims of WSMO is Web service discovery (we can see WSMO as an application for service discovery), and to enable discovery, WSMO describes client *Goals* and the *Capabilities* of the available service. *Goals* and *Capabilities* are the necessary inputs to a matching algorithm, therefore they are functional aspects of WSMO descriptions. While pure Web service discovery only requires the *Capabilities* and *Goals*, the match maker can also take into account preferences and constraints over parameters like the price, availability or quality of a service. Because such parameters are not critical for the match maker, they are modeled as non-functional properties.

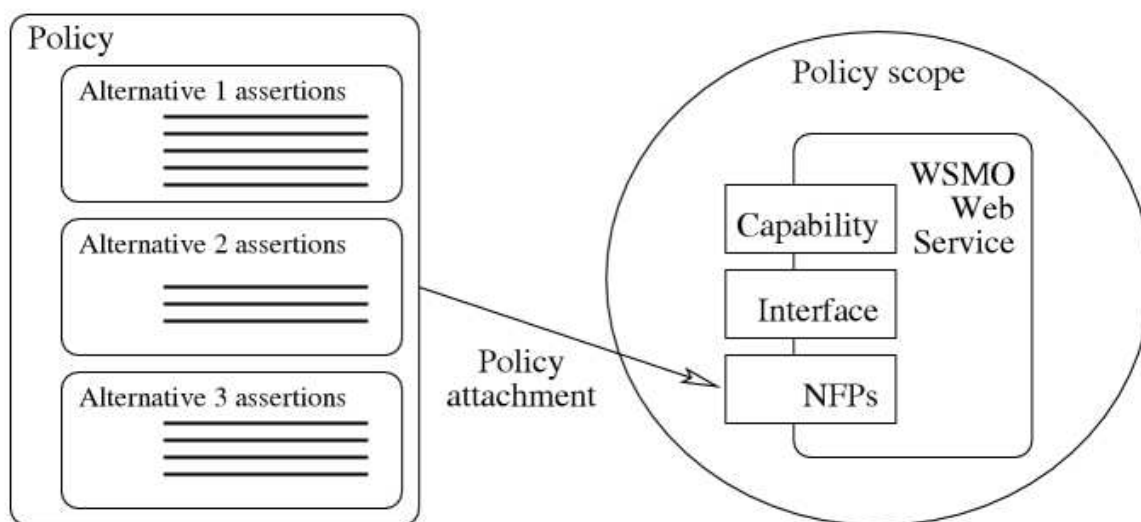
The distinction between functional and non-functional parameters depends highly on the application that uses the particular parameter — a functional parameter of one application can be non-functional in another. For instance, Semantic Web Services are an application that automates the use of Web services, and the price of a service is generally modeled as a non-functional property; however a shopping agent application will have price as one of its main functional parameters.

In WSMO, the distinction between functional and non-functional properties is made very clear: WSMO enumerates all the relevant functional properties (for example *Web service* has *Capability*

and *Interface* as its functional properties) and it allows an extensible bag of NFPs everywhere. WS-Policy does not have any such distinction, so it can be used to express both functional and non-functional policy assertions, depending on the application that employs policies. Since WSMO enumerates in its conceptual model all the parameters that are functional for the aim of WSMO, policies can be treated as non-functional data, therefore when a WS-Policy is attached to a WSMO element, it is abstractly added to the non-functional properties of that element.

Figure 2. shows how WSMO elements can be used as policy assertions in a policy that is then attached to a particular policy scope. In particular, a policy is attached here to a WSMO Web service description, and it is treated as a non-functional property of that service. Due to the way policy attachment works syntactically, the policy is in fact embedded or referenced from within the non-functional properties block of the Web service description.

Figure 2: Attaching Policies to WSMO Elements



To summarize, WSMO elements can serve as WS-Policy Policy Subjects, and any policies attached to those WSMO elements are treated as non-functional properties. In the following, we present the syntax for including policies as non-functional properties in WSMO descriptions. Using this mechanism, the existing and future policy assertions can usefully complement the Dublin Core NFPs [DublinCore] currently used by WSMO.

To attach policies generically to XML elements such as WSDL descriptions, the WS-PolicyAttachment specification [WSPolicyAttachment] defines an XML attribute called PolicyURIs and an XML element called PolicyReference. Both the attribute and the element point to external policies using URIs. WS-PolicyAttachment introduces both of them because some XML languages restrict attribute or element-based extensibility.

In WSMO, we use the namespace-qualified name `wsp:PolicyReference` for an NFP; its value is one or more URIs of policies attached to the owner WSMO element:

```

01 service ACMEService
02   nonFunctionalProperties
03     wsp#PolicyReference hasValue
04       {_"http://fabrikam123.example.com/policies/DSIG",
05         _"http://fabrikam123.example.com/policies/SECTOK"}
06   endNonFunctionalProperties

```

Note that in WSMO, namespace-qualified names are written with a hash-sign (i.e. `wsp#PolicyReference`) and not with a colon as in XML (i.e. `wsp:PolicyReference`).

In some cases it can be useful for manageability reasons to include the whole policy in the WSMO description, especially when the policy is fairly small. For this purpose we reuse the namespace-qualified name `wsp:Policy` as the name of a non-functional property whose content is the XML serialization of the whole Policy element. This is illustrated by the following example:

```

01 service ACMEService
02   nonFunctionalProperties
03     wsp#Policy hasValue
04       "<wsp:Policy>
05         <wsp:ExactlyOne>

```

```

06     <wsse:SecurityToken>
07     <wsse:TokenType>wsse:Kerberosv5TGT
08     </wsse:TokenType>
09     </wsse:SecurityToken>
10     <wsse:SecurityToken>
11     <wsse:TokenType>wsse:X509v3
12     </wsse:TokenType>
13     </wsse:SecurityToken>
14     </wsp:ExactlyOne>
15     </wsp:Policy>
16     endNonFunctionalProperties

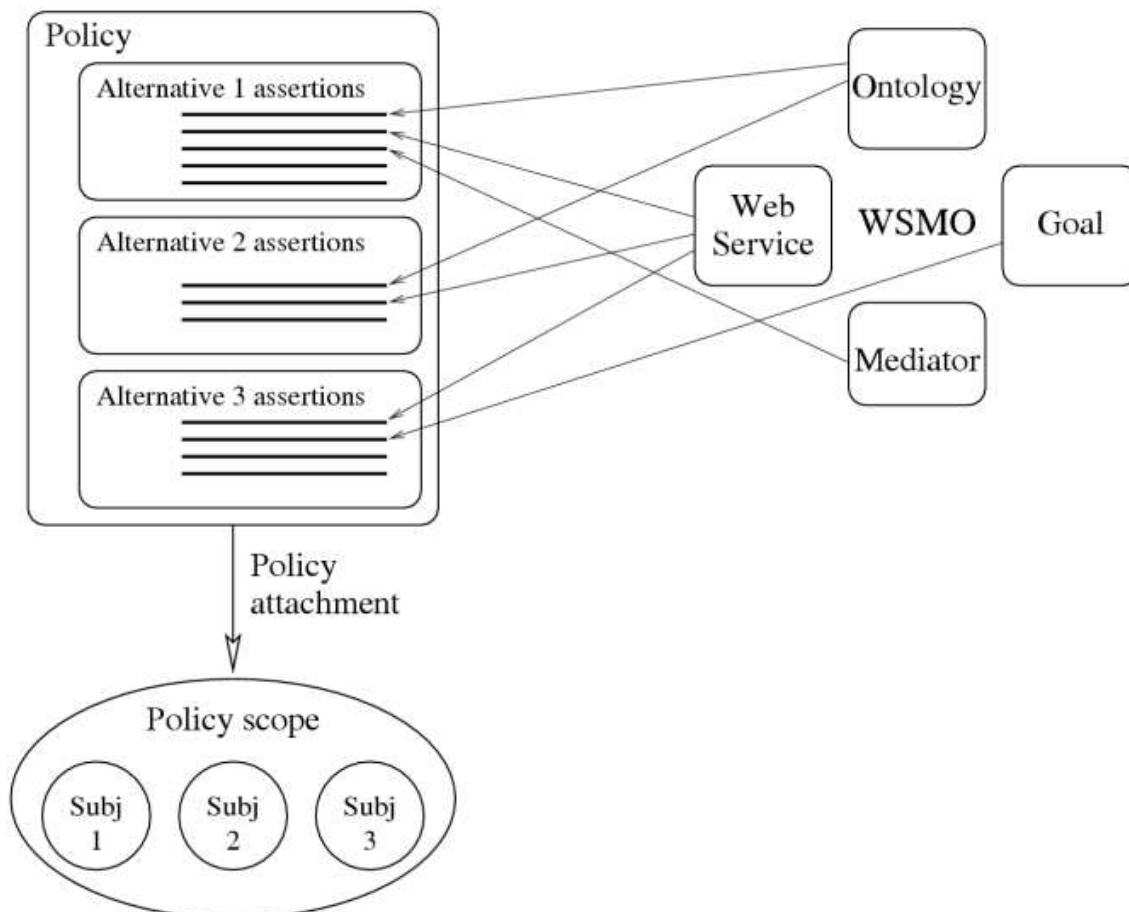
```

To summarize, we allow attaching both external and embedded policies to WSMO elements, treating the attached policies as non-functional properties. For this, we reuse the WS-Policy element names (`wsp:PolicyReference` and `wsp:Policy`) as NFP identifiers in WSMO.

### 3.2 WSMO Assertions in WS-Policy

WS-Policy is a mechanism of combining domain-specific policy assertions and attaching them to various policy subjects. WSMO descriptions can be viewed as policy assertions and combined with others in policy alternatives. Figure 3. shows how WSMO elements can be used as policy assertions in a policy that is then attached to a particular policy scope, for example a Web service endpoint. Such a policy would thus attach the WSMO Web service description to that endpoint.

Figure 3: Using WSMO Elements as Policy Assertions



We can envision, for example, a policy that ties the capabilities of a Web service with various security and Quality of Service (QoS) settings. A service may offer some basic functionality with strong authentication but weak communication channel encryption, and more advanced functionality can be available provided that strong encryption is employed.

WSMO currently does not have any specific mechanism for expressing that alternative WSMO descriptions are in effect in conjunction with various non-functional properties, and WS-Policy seems to be a widely-adopted mechanism for expressing exactly such alternatives, therefore even though WSMO descriptions would not normally be treated as policy assertions, such an approach may prove beneficial. In fact, WSMO descriptions captured as policy assertions would not *know* that they are

policy assertions, i.e., the WSMO descriptions are oblivious to the policy context, but from the point of view of the policy, the chosen policy alternative would guide the processor in choosing the *effective* WSMO description, ignoring all others.

Because policy assertions in WS-Policy must be XML elements, we can reuse WSML/XML serialization format for representing WSMO descriptions as policy assertions in WS-Policy. To attach a whole WSMO description as a single policy assertion, we use the element `wsm:wsml` (the first `wsm` is a namespace prefix and the second is the name of the XML element container for WSML/XML syntax). For finer granularity (e.g. only asserting a single capability description) we can reuse the appropriate elements like `wsm:capability`.

In some situations it may be beneficial only to refer to a WSMO description (as opposed to including it inline as a policy assertion). For referring to a whole WSML file we introduce the element `wsm:descriptionReference` that refers to a WSML document, and similarly we can introduce specific elements for referring to specific elements of WSMO, for instance to refer to a capability or an interface we can use elements `wsm:capabilityReference` and `wsm:interfaceReference` that would refer to the identifiers of the WSMO capability or interface descriptions.

The listing below is a policy that claims the policy subject is described by the WSMO services `http://example.org/services/ticketService` and `http://example.org/services/billingService`. Lines 2--7 show a WSML/XML element `wsm:webService` that, in this context, means a policy assertion assigning a WSMO service description defined inline. Similarly, lines 13--15 show a reference to such a description, using the new element `wsm:serviceReference`. Finally, lines 8--12 contain an ontology which is used by the `ticketService` definition.

```

01 <wsp:Policy>
02 <wsm:webService
03   name="http://example.org/services/ticketService">
04   <wsm:capability name="ticketing">
05     ...
06   </wsm:capability>
07 </wsm:webService>
08 <wsm:ontology
09   name="http://example.org/ontologies/ticketing/">
10   <wsm:concept name="Ticket"/>
11     ...
12 </wsm:ontology>
13 <wsm:serviceReference>
14   http://example.org/services/billingService
15 </wsm:serviceReference>
16 </wsp:Policy>

```

The conclusion is that to represent a policy assertion "the policy subject has the following WSMO description" we can use any global WSML/XML element as appropriate, and to represent an assertion only referencing a WSMO description we have to create specific elements for each type of WSMO entity that we want to reference.

The table below summarizes the syntax introduced both in this section and in the preceding one — the table shows all the syntactical ways in which WSMO can be combined with WS-Policy.

Name	Type	Description
<code>wsp#PolicyReference</code>	NFP identifier	non-functional property referring to an external policy file by URI
<code>wsp#Policy</code>	NFP identifier	non-functional property containing an XML serialization of a policy
<code>wsm:wsml</code>	XML Element	the root element of WSML/XML syntax, reused as a policy assertion "the embedded WSML description applies (to the policy subject)"
<code>wsm:webService</code> <code>wsm:goal</code> ...	XML Element	other WSML/XML elements reused as a policy assertion "the embedded WSML description applies (to the policy subject)"
<code>wsm:descriptionReference</code> <code>wsm:webServiceReference</code> <code>wsm:goalReference</code> ...	XML Element	policy assertions that refer to WSMO definitions by their identifier URIs

### 3.3 Policy in SEE components

In this section we attempt to analyze how policy frameworks with their known uses can be incorporated in the various components of the SEE. The following list explains how each component is affected by policies.

#### **Formal languages:**

As most of the components in SEE work with semantic descriptions of goals and Web services, the formal language used for those descriptions needs to include ways of attaching policies. In the Web Service Modeling Language (WSML), policies could be attached as non-functional properties `wsp#Policy` and `wsp#PolicyReference`, as we propose in [Section 3.1](#). In this deliverable, we do not propose any further extension to WSML, which could have been similar in function to WS-Policy's `All` and `ExactlyOne` constructs. Such WSML extensions would fall into the realm of semantically representing policies, which is outside the scope of this deliverable.

#### **Reasoning:**

This component provides inference and reasoning capabilities over logical formalisms, therefore while it may be used to evaluate policies, its function is not affected.

#### **Storage and Communication:**

Serving purely the internal purposes of the SEE, this component is also not affected by application policies. If the SEE itself is distributed over multiple administration domains, it would be natural to extend this component with policies, yet the application of policies in such a setting has little to do with the semantic functionalities of SESA. Therefore we propose that from the point of view of using policies in SESA, enhancing the Storage and Communication component with policies should be viewed as an implementation and deployment detail.

#### **Discovery:**

The purpose of the Discovery component in SESA is to perform functional matching between goals and Web services. Since we view policies as non-functional characteristics which are processed in the Adaptation component, we propose that the Discovery component is neither affected by policies.

#### **Adaptation:**

As this component contains negotiation and non-functional property evaluation, it is a natural fit for policy evaluation as well. In particular, this component would use a WS-Policy evaluator to filter out the Web services whose policies conflict with the policy constraints of the user, with the additional possibility of ranking the services according to the user's policy preferences.

#### **Composition:**

After it composes a set of Web services, this component can use their policies to verify that the composed services will be able to work with each other. Apart from the typical policy-match considerations like whether all the services support the required level of confidentiality, also higher-level policies come into play here, for example specifying that services from two given providers cannot be used together — for instance [\[Uzok2004b\]](#) presents a coalition policy against using aircraft from a given country in rescue operations. Similarly to the Adaptation component, the Composition component would use a WS-Policy evaluator to implement such considerations.

#### **Choreography:**

The choreography interface of a service prescribes (or limits) the steps that need to be followed by the client in order to successfully interact with the service. Choreographies in SESA are specified again on the level of functionality, independent of non-functional properties like policies. However, as choreographies are seldom completely strict (i.e., there can be optional and even parallel branches), the choreography engine should take operation-level policies as additional choreography constraints.

#### **Mediation:**

This component currently contains data mediation for handling heterogeneous data formats, and process mediation for resolving choreography mismatches. To address differences in policy specifications, we propose to add policy mediation (cf. [\[Galiasso2000\]](#), [\[Wohlstadter2004\]](#)) to this component's functionalities.

#### **Grounding:**

As this component is responsible for external communication, it is in the position to act as a policy enforcer. On one side, the SEE can have, for example, authorization and security policy affecting the clients who wish to invoke the SEE. On the other side, when the SEE communicates with Web services, it needs both to enforce its internal policy and to comply to the policies of the invoked Web services.

#### **Fault Handling:**

The functionality of this component can be completely or partially driven by policy specifications. In fact, fault handling policy can be specified both by the SEE and by the Web service that originated the fault (e.g. WS-Policy allows the specification of a policy on a WSDL fault description).

#### **Monitoring:**

Similarly to fault handling, monitoring can also be controlled by policies, both an internal SEE policy and the policies of monitored Web services.

**Ontologies:**

As part of the "Problem Solving Layer", this component is oriented towards the applications of the SEE. Domain ontologies that fall within this component can also be used for specifying the various policies evaluated by the components mentioned above, but the definition of the component is not affected by the introduction of policies into the SEE.

**Applications:**

Any use case scenarios developed within this component should be able to use policies, however just as for the Ontologies component, the definition of the Applications component is not affected by the introduction of policies.

**Developer Tools:**

The SEE should provide a tool for creating and verifying policies. Such a tool should be adopted from existing policy frameworks (cf. [Bhargavan2004], [KAoS]), if available.

**Execution Management:**

All the policy evaluation and enforcement steps are incorporated in the appropriate SEE components, therefore the execution semantics of the system is not affected by the presence of policies.

**Security:**

Policies can be used for expressing and enforcing various security requirements, therefore on the implementation level, some of the functionality of this component will be preempted by policy evaluation and enforcement in the other components. However, the definition or functionality of this abstract component does not necessarily change.

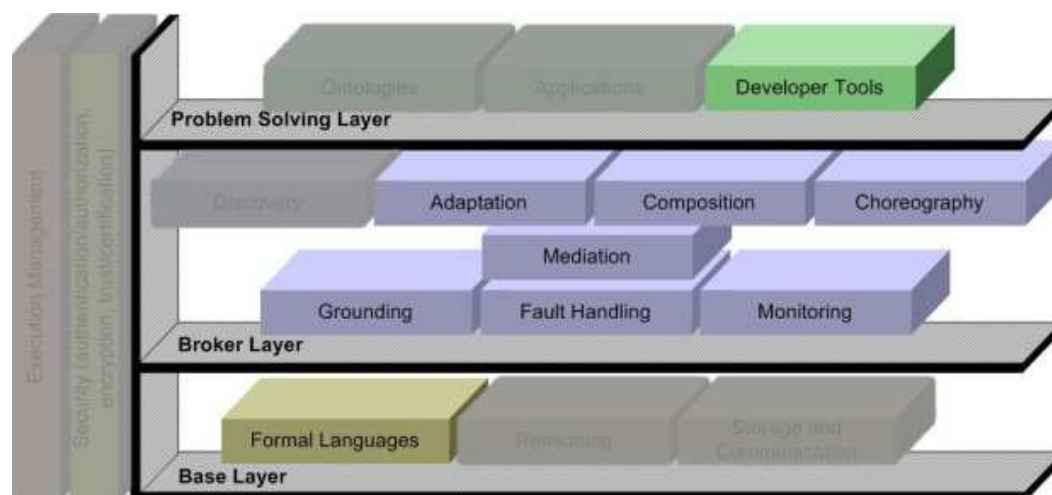
We can see that introducing policies in SESA mainly affects the Adaptation and Grounding components, as policy evaluation and enforcement are major parts of the functionalities of these components. Also Mediation, as a crucial component for working in such a distributed environment as that of Web services, can be enhanced with the significant task of policy mediation.

Components like Composition and Choreography, use policies as additional inputs, whereas the Fault Handling and Monitoring components are policy-driven, i.e. the policies fully dictate what these components will do.

Finally, some components stay oblivious to policies, for example Execution Management, Reasoning and Discovery. Execution management is unchanged because we propose no new SEE execution steps, instead all policy evaluation happens in the pre-existing components. The reasoning task is on a lower level than that of policy evaluation, therefore a policy engine can use a reasoner, but not vice versa. The task of discovering suitable services, however, is not as clearly separated from policy evaluation. The definition of the Discovery component in our SEE model focuses on functional discovery based on a registry of Web service descriptions, and the Adaption component handles negotiation and non-functional properties; but in other models, the term "Discovery" can include what we call adaptation, and even composition, and therefore it would include policy evaluation as well.

The figure below shows that introducing policy to SEE affects mostly the Broker Layer components, with the language support in the Formal Languages component and tool support in the Developer Tools component. While, in the diagram, we could just add policy as another vertical service next to the Security component, we wanted to show in this deliverable how policy functionality would spread through the existing components.

Figure 4: SEE Components affected by policy



### 3.4 Policy Tasks in SEE

In the previous section, we analyze all the SEE components and we show how they can involve policies. However, one can get lost in the enumeration of components, so this section summarizes the policy-related tasks that are implemented in the various components:

**Creating policies** is naturally the first task, before any other policy-related activities can occur. The Developer Tools component should provide a user tool for specifying, changing and verifying policies, so that they can be deployed into the SEE. Additionally, the WSML tools should support attaching policies to WSML descriptions, using the non-functional properties specified within the Formal Languages component.

**Policy matching** is done in the Adaptation and Composition components, where the Web service policies are matched for example against the policy of the SEE user for adaptation and against other services for composition.

**Policy enforcement** occurs in two different settings. First, user policy or an internal SEE policy governs the tasks of the Grounding, Fault Handling and Monitoring components, and here these components basically implement what the policies say. On the other hand, all these three components, together with the Choreography component, must also comply to policies of external services. To comply with all the appropriate policies, any component may use Composition to get help from external services.

Finally **policy mediation**, the new task of the Mediation component, can also be used in any of the other components, when mismatches are encountered.

The following table summarizes these policy-related tasks and the components that are involved:

Task	Components
Policy creation	Developer Tools Formal Languages
Policy matching	Adaptation Composition
Policy enforcement	Grounding Fault Handling Monitoring Choreography Composition
Policy mediation	Mediation

## 4. Related Work

To the best of our knowledge, this deliverable represents the first comprehensive analysis of how policy frameworks can be integrated with fully-fledged semantic execution environments, and especially how WS-Policy can be combined with WSMO. Nevertheless, there are publications that apply various policy frameworks in Web service systems, including Semantic Web Service applications, and then there are publications that apply semantic approaches for specifying and evaluating policies.

Policies have been applied in Semantic Web Service systems mainly to handle security, for instance Kagal et al. [Kagal2003] "mark up web entities with a semantic policy language" in order to "use distributed policy management as an alternative to traditional authentication and access control schemes." Similarly, Uszok et al. [Uszok2004b] use a policy framework (KAoS, also described below) for authorization in Grid computing, with policies of the form "It is permitted for actor(s) X to perform action(s) Y on target(s) Z." However, they use policies outside of security as well: for instance, in a system that uses a generic semantic matchmaker to help with organizing rescue operations, the policy framework is employed to enforce a coalition policy that rescue operations must not engage aircraft from a specific fictitious country. This approach additionally makes it possible to hide said coalition policy, so that the affected country is not aware of it. On top of this, Uszok et al. also use policy for verification of semantically-created compositions and contracts, and to ensure that at runtime, all interactions will comply to given policies.

Other works deal with combining policies and Semantic Web technologies in general. We can classify such works in such those that deal directly with WS-Policy, and those which take a more general approach to policies, and thus not commuting to WS-Policy as a framework for representing policies.

Works that deal directly with WS-Policy include [Kolovski05], [Verma05], and [SrihareeSVS04]. In

[Kolovski05], Kolovski et al. provide a mapping of WS-Policy to the description logic fragment species of the Web Ontology Language (OWL-DL), and describe how standard OWL-DL reasoners can be used to check policy conformance and perform an array of policy analysis tasks. In [Verma05], Verma et al. describe how to use domain ontologies in OWL for creating policy assertions with semantics in WS-Policy; the aim there matching the non-functional properties of Web Services represented using WS-policy. Sriharee et al. propose an approach [SrihareeSVS04] to behaviour-based discovery of Web Services by which business rules that govern service behaviour are described as a policy. The policy is represented in the form of ontological information and is based on actions relating to the service and conditions for performing them. The standard WS-Policy is used to associate such a policy to the Web Service.

Works that do not deal directly with WS-Policy (but which takes into account semantic web based approaches to policies) focuses more on administrative policies such as security and resource control policies. KAoS [KAoS] is one of the first efforts to represent policy using a Semantic Web language (in this case OWL). KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of policies within the specific contexts established by complex organizational structures represented as domains. While initially oriented to the dynamic and complex requirements of software agent applications, KAoS services have been extended to work equally well with both agent and traditional clients on a variety of general distributed computing platforms. In [Nejdl05], Nejdl et al. propose the use of ontologies to simplify the tasks of policy specification and administration, discusses how to represent policy inheritance and composition based on credential ontologies, and formalize these representations and the according constraints in Frame-Logic. Toninelli et al. classify in [Toninelli2005] approaches to policy specification as ontology-based approaches (that rely heavily on the expressive features of Description Logic languages), and rule-based approaches (that encode policies as Logic Programming rules), and proposes a hybrid approach that exploits the expressive capabilities of both DL and LP approaches.

Rein ([Kagal2006] and [Kagal2006a]) is a framework for representing and reasoning over policies in the Semantic Web; it exploits the inherently decentralized and open nature of the Web by allowing policies, meta-policies, and policy languages to be combined, extended, and otherwise handled in the same scalable, modular manner as are any Web resources. Resources, their policies and meta-policies, the policy languages used, and their relationships together form Rein policy networks. These networks are described using Rein ontologies and these distributed but linked descriptions are collected off the Web by the Rein engine and are reasoned over to provide policy decisions. Rein does not propose a single policy language for describing policies; it allows every user to potentially have her own policy language or re-use an existing language and if required, a meta policy. The ontologies and reasoning mechanisms work with any policy language and domain knowledge defined in RDFS, OWL, or supported rule languages.

Within DIP, there has been some work done on Quality of Service (QoS) support in discovery, e.g. in Deliverable D4.17 and the prototype D4.19. QoS conditions and constraints are often cited as example policy assertions, therefore we should mention here the relation of those DIP deliverable and D4.13, this deliverable. D4.17 defines QoS parameters semantically (as axioms) in the non-functional properties of Web service interfaces. Such descriptions could be combined with semantic approaches to representing policies, but such approaches are outside of the scope of D4.13. On the other hand D4.17 does not use WS-Policy, as that would be an unnatural detour between the QoS axioms and WSMML. A similar situation occurs with deliverable D4.21 on security and trust, which could also use WS-Policy, in which case security and trust statements would be attached using the mechanisms from this deliverable.

## 5. Conclusions and Further Work

This deliverable is a first step to combine WS-Policy (a policy framework with significant industry backing) and WSMO (one of the most important proposals for Semantic Web Services to date) or SESA (Semantically Enabled Service-oriented Architecture) in general. We identified potential ways of combining them, and provided the necessary syntax.

The proposals presented in the deliverable represent the basis for enabling the use of policies in SWS environments for tasks such as policy-based semi-automatic negotiations between users and services, policy-based matchmaking between users requests and services, scheduling of service compositions under certain constraints, etc; but also for enabling the use of semantic descriptions as policy assertions.

Having gone through the envisioned components of a semantic execution environment (SEE), we identified the components that may make use of policy evaluation. In a short summary, the Adaptation and Grounding components are the most affected, representing the bulk of policy evaluation and enforcement in the SEE. We also identified components that can be fully controlled by policies, i.e., the Fault Handling and Monitoring components.

From our analysis it is apparent that most of the SEE can be affected by policies, and we aim for this deliverable to serve mainly as "the big picture" when SEE implementations start to incorporate policy frameworks. Also, it is clear that policy cannot easily be treated as an additional, independent component, because the different policy-related tasks executed in the SEE are also distributed across the whole spectrum of the components. There cannot be a single step in the execution semantics of a SEE, where policy is evaluated and enforced. We could say that, at the diagram level, policy could just be added as another vertical service next to the Security component, but in this deliverable we wanted to show how policy functionality spreads through the existing components. A similar analysis could also be done for security.

As future work, apart from implementing full WS-Policy support in WSMX, an investigation should also be done on how semantic approaches to policy specification and evaluation could be combined with semantic Web service descriptions. Semantic policy could mean that policy evaluation would dissolve into the existing semantic matching tasks within the Adaptation and Composition components, and maybe semantic policy evaluation could even trickle into the Discovery component.

## References

**[Bhargavan2004]** K. Bhargavan, C. Fournet, A. D. Gordon: *Verifying policy-based security for web services* In CCS '04: Proceedings of the 11th ACM conference on Computer and communications security, 2004, Washington DC, USA.

**[DublinCore]** S. Weibel, J. Kunze, C. Lagoze, and M. Wolf: *Dublin Core Metadata for Resource Discovery*, IETF RFC 2413, September 1998, available at <http://mirror.switch.ch/rfc/2413.txt>.

**[Galiasso2000]** P. Galiasso, O. Bremer, J. Hale, S. Sheno, D. Ferraiola, V. Hu: *Policy mediation for multi-enterprise environments*. In ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference, 2000, Washington, DC, USA.

**[Kagal2003]** Lalana Kagal, Tim Finin, Anupam Joshi: *A Policy Based Approach to Security for the Semantic Web*. In 2nd International Semantic Web Conference (ISWC2003), 2003.

**[Kagal2006]** L. Kagal, T. Berners-Lee, D. Connolly, D. J. Weitzner: *Using Semantic Web Technologies for Policy Management on the Web*. In 21st National Conference on Artificial Intelligence (AAAI), July 2006.

**[Kagal2006a]** L. Kagal, T. Berners-Lee, D. Connolly, D. Weitzner: *Self-Describing Delegation Networks for the Web*. In Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06), 2006.

**[KAoS]** A. Uszok, J. M. Bradshaw, R. Jeffers, A. Tate, J. Dalton: *Applying KAoS Services to Ensure Policy Compliance for Semantic Web Services Workflow Composition and Enactment*, In International Semantic Web Conference, 2004.

**[Kolovski05]** V. Kolovski, B. Parsia, Y. Katz, J. A. Hendler: *Representing Web Service Policies in OWL-DL*. In International Semantic Web Conference, 2005.

**[Nejdl05]** W. Nejdl, D. Olmedilla, M. Winslett, C. C. Zhang: *Ontology-Based Policy Specification and Management*. In 2nd European Semantic Web Conference (ESWC), 2005.

**[SrihareeSVS04]** N. Sriharee, T. Senivongse, K. Verma, A. P. Sheth: *On Using WS-Policy, Ontology,, Rule Reasoning to Discover Web Services*. In INTELLCOMM, 2004.

**[Toninelli2005]** A. Toninelli, J. Bradshaw, L. Kagal, R. Montanari: *Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments*. In Proceedings of the Semantic Web and Policy Workshop, November 2005.

**[Uszok2004b]** A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, S. Aitken: *Policy and Contract Management for Semantic Web Services*. In AAAI Spring Symposium on Semantic Web Services, 2004.

**[Verma05]** Verma, K., Akkiraju, R., Goodwin, R.: *Semantic Matching of Web Service Policies*. In SDWP Workshop, 2005.

**[Wohlstadter2004]** E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, P. Devanbu: *GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions*. In 26th International Conference on Software Engineering, 2004.

**[WSPolicy]** J. Schlimmer (editor): *Web Services Policy Framework*, available at

<http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp>

**[WSPolicyAssertions]** A. Nadalin (editor): *Web Services Policy Assertions Language*, available at <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassertions.asp>

**[WSPolicyAttachment]** C. Sharp (editor): *Web Services Policy Attachment*, available at <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyattachment.asp>

**[WSSecurity]** A. Nadalin, C. Kaler, P. Hallam-Baker, R. Monzillo (editors): *Web Services Security*, available at <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

**[WSSecurityPolicy]** A. Nadalin (editor): *Web Services Security Policy Language*, available at <http://msdn.microsoft.com/ws/2002/12/ws-security-policy/>

## Acknowledgement

The work is funded by the European Commission under the projects ASG, DIP, EASAIER, enIRaF, Knowledge Web, Musing, Salero, Seemp, SemanticGOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/113 ; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects Grisino, RW<sup>2</sup>, SemBiz, SeNSE and TSC.

The editors would like to thank to all the members of the WSMO working group for their advice and input to this document.

## Change Log

Date	Author	Description
2007/2/16	jacek	created from d24.1 and from the DIP deliverable d4.13



\$Date: 2007/01/12 17:08:08 \$

webmaster