DIP
Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP4: Service Usage
D4.13

Aligning WSMO and WSMX with Existing Policy Specifications

Jacek Kopecký
Dumitru Roman

January 12, 2007
THE DIP project has focused on Semantic Web Services (SWS), mainly contributing to the development of the Web Service Modeling Ontology (WSMO), and its execution environment, WSMX. SWS research aims 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. Automation will make Web service usage cheaper and free the human resources for higher-level work, but it can also enable proliferation of Web services on a much higher scale than currently feasible.

Semantic Web Services are a major aspect of the so-called Semantically Enabled Service-oriented Architecture (Semantic SOA, or SESA), and also 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, policies are hard-coded into the system. 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.

However, policies were mostly out of scope for the DIP project. This deliverable aims to analyze how policies can be applied to SESA in general and WSMO in particular. We have found that most of the architecture of a semantic execution environment like WSMX can be affected by policy evaluation and enforcement, and that policy cannot be treated as a single component in the system.

This deliverable is intended for researchers and implementors of semantically enabled service-oriented architectures, especially if they base their efforts on the prospective standard for a Semantic Execution Environment (SEE) architecture coming from the SEE TC in OASIS.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
# Abstract (for dissemination)

Service-Oriented Architectures (SOAs) suggest that IT systems should be developed from coarse-grained, loosely coupled, business-aligned components, so called services. One way towards loose coupling is to refrain from hard-coding policies in the system and to represent them explicitly. In this deliverable we analyze where Web service policies fit within the big picture of Semantic SOAs and we present a proposal for combining WSMO, our Semantic SOA framework, and WS-Policy Framework, a set of specifications with heavy industrial backing.

## Keywords
- WS-Policy
- WSMO
- WSMX
# PROJECT CONSORTIUM INFORMATION

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway                                | NUIG    | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: chris.bussler@deri.ie  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter                         | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| British Telecommunications Plc.                              | BT      | Dr. John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne              | EPFL    | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
E-mail: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council                                          | Essex   | Mary Rowlatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
E-mail: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik                                  | FZI     | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | UIBK    | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Logo</th>
<th>Contact Person</th>
<th>Address/Location</th>
<th>Email/Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td><img src="image" alt="ILOG Logo" /></td>
<td>Christian de Sainte Marie</td>
<td>9 Rue de Verdun, 94253, Gentilly, France</td>
<td>E-mail: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a> Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td><img src="image" alt="inubit Logo" /></td>
<td>Torsten Schmale</td>
<td>inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany</td>
<td>E-mail: <a href="mailto:ts@inubit.com">ts@inubit.com</a> Tel: +49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td><img src="image" alt="iSOCO Logo" /></td>
<td>Dr. V. Richard Benjamins, Director R&amp;D</td>
<td>Pedro de Valdivia 10 28006 Madrid, Spain</td>
<td>E-mail: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a> Tel: +34 913 349 797</td>
</tr>
<tr>
<td>MDR Partners</td>
<td><img src="image" alt="MDR Logo" /></td>
<td>Rob Davies</td>
<td>MDR Partners 8 St. Andrew Street Hertford, Herts. United Kingdom, SG14 1JA, E-mail: <a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a> Tel: +44 (0)208 8763121</td>
<td></td>
</tr>
<tr>
<td>Hanival Internet Services GmbH</td>
<td><img src="image" alt="Hanival Logo" /></td>
<td>Alexander Wahler</td>
<td>Hanival Internet Services GmbH Kirchengasse 13/1a A-1070 Wien Email: <a href="mailto:wahler@niwa.at">wahler@niwa.at</a> Tel:+43(0)1 3195843-11 —</td>
<td></td>
</tr>
<tr>
<td>The Open University</td>
<td><img src="image" alt="OU Logo" /></td>
<td>Dr. John Domingue</td>
<td>Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a> Tel.: +44 1908 655014</td>
<td></td>
</tr>
<tr>
<td>SAP AG</td>
<td><img src="image" alt="SAP Logo" /></td>
<td>Dr. Elmar Dorner</td>
<td>SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a> Tel: +49 721 6902 31</td>
<td></td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td><img src="image" alt="Sirma Logo" /></td>
<td>Atanas Kiryakov</td>
<td>Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a> Tel.: +359 2 9768 303</td>
<td></td>
</tr>
</tbody>
</table>
Aligning WSMO and WSMX with Existing Policy Specifications

Carlo Wouters,
Starlab- VUB
Vrije Universiteit Brussel
Pleinlaan 2, G-10
1050 Brussel, Belgium
E-mail: carlo.wouters@vub.ac.be
Tel.: +32 (0) 2 629 3719
LIST OF KEYWORDS/ABBREVIATIONS

DC ................. Dublin Core
MOF ................. Meta Object Facility
NFP ................. Non-functional properties
OASIS .............. Organization for the Advancement of Structured
                  Information Standards
OWL ................. Web Ontology Language
OWL-DL ............ Web Ontology Language – Description Logics fragment
OWL-S .............. OWL-based Web service ontology
RDF ................. Resource Description Framework
RDFS ............... RDF Schema
SAML ............... Security Assertion Markup Language
SEE .................... Semantic Execution Environment (e.g. WSMX)
SEE TC ................ Semantic Execution Environment Technical Committee
                  at OASIS
SESA .................. Semantically Enabled Service-oriented Architecture
SOA .................... Service-Oriented Architecture
SWS ..................... Semantic Web Services
WS ...................... Web Services
WSDL .................. Web Service Description Language
WSML .................. Web Service Modeling Language
WSMO .................. Web Service Modeling Ontology
WSMX .................. Web Service Execution Environment
XACML ................ Extensible Access Control Markup Language
XML .................... Extensible Markup Language
# Table of Contents

1 **Introduction** 1  
2 **Semantically Enabled Service-oriented Architecture** 3  
   2.1 WSMO Conceptual Model 4  
   2.2 SEE Components 7  
3 **Web Services Policy** 10  
4 **Combining Policy with SESA and WSMO** 13  
   4.1 Attaching Policies to Semantic Descriptions in WSMO 13  
   4.2 WSMO Assertions in WS-Policy 16  
   4.3 Policy in SEE components 19  
   4.4 Policy Tasks in SEE 22  
5 **Related Work** 23  
6 **Conclusions and Outlook** 25
LIST OF FIGURES

2.1 Components of a semantic execution environment .................................. 7
3.1 WS-Policy building blocks ......................................................................... 11
3.2 An example policy requiring the use of Kerberos or X509 security tokens .... 11
4.1 Attaching Policies to WSMO Elements ..................................................... 14
4.2 Using WSMO Elements as Policy Assertions ........................................... 16
4.3 Example policy with WSMO webService as policy assertion ..................... 17
4.4 SEE Components affected by policy ......................................................... 21
LIST OF TABLES

4.1 Syntax for combining WSMO and WS-Policy ........................................ 18
4.2 Policy-related tasks in SEE components ............................................. 22
1 Introduction

Web services and, more generally, the so-called Service-Oriented Architecture (SOA) are gaining significant adoption in areas of application integration, wide-scale distributed computing, and business-to-business cooperation. Based on World-Wide Web standards, Web services technologies lower the barrier to entry, enable loosely coupled system development, and simplify evolution and maintenance of distributed systems.

The research areas of Semantically Enabled Service-oriented Architectures (SESA) and Semantic Web Services (SWS), embodied in the DIP project, 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 [2] technologies, i.e. ontologies and reasoning. Automation will make Web service usage cheaper and free the human resources for higher-level work, but it can also enable proliferation of Web services on a much higher scale than currently feasible.

Also 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\(^1\), 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 [22] enable policies to be captured independent from a concrete system implementation. In such a setting, policies are more flexible, reusable, verifiable, extensible and efficient [1]. 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 [28] 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 Chapter 6 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.

\(^1\)A more detailed discussion on functional vs. non-functional properties can be found in the position paper for the W3C Workshop on Constraints and Capabilities for Web Services [1].
Common denominators have been identified between policies that may be leveraged to improve policy uniformity and streamline service-oriented enterprise-wide policy implementation. In particular, a set of new enforcement policies were proposed in the industry using novel policy concepts together with building blocks from e.g. XACML\(^2\), WS-Policy\(^{[22]}\) and SAML\(^3\). For the purpose of this deliverable, we have chosen to use the Web Services Policy Framework (WS-Policy)\(^{[22]}\), 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.

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 concrete contributions of this deliverable, presented in Chapter 4, are: the syntax for attaching WS-Policy policies to WSML components, the syntax for putting WSML descriptions as policy assertions inside policy alternatives, and the analysis of what WSMX/SEE components can include policy evaluation and for what tasks.

In this deliverable, we often use the name “WSMO” to represent the whole family of the conceptual model (WSMO), the formal languages (WSML) and the execution environments (e.g. WSMX), but we do so in places where these distinctions do not play a significant role.

To provide the background for our work, we describe in Chapter 2 what we mean by Semantically Enabled Service-oriented Architecture (SESA), with a description of the WSMO conceptual model and WSMX architecture. This chapter can be skipped by readers familiar with WSMO and WSMX, and it is here to make this deliverable useful by itself to readers who come from outside the project. In Chapter 3 we give an overview of the WS-Policy framework. Chapter 4 contains the main contributions of this deliverable – we show how policies fit in SESA, we detail what implementation-level components can use a policy engine, and what syntactical constructs are necessary for combining WSMO with WS-Policy. Finally, in Chapter 5 we talk about related work and Chapter 6 presents our conclusions and outlook for future work, most probably in projects following DIP.

\(^2\)http://www.oasis-open.org/committees/xacml/
\(^3\)http://www.oasis-open.org/committees/security/
2 Semantically Enabled Service-oriented Architecture

In this chapter, we give an overview of what we mean by Service-oriented Architecture (SOA) and why it needs to be enhanced with semantic technologies, leading to a Semantically Enabled Service-oriented Architecture (SESA). We also describe how SESA is realized in the WSMO framework, what tasks the framework accomplishes and what components it has, so that further in this deliverable we can analyze how WS-Policy fits in all this. This chapter can be skipped by readers already familiar with WSMO and WSMX.

In contrast with its name, Service-oriented Architecture (SOA) does not mean a single computing system architecture, instead it is a set of guidelines towards service-orientation. The essence of SOA is a computational model based on remote service invocation [5]. Services are software systems that tightly represent well-defined business functionality, and according to Randy Heffner [9], a service-oriented system supports business change, connection and control. Loosely coupled services can be changed faster as the requirements evolve, or as we refine our understanding of the requirements. Services can be connected where and when needed in larger compositions, even when such compositions were not anticipated by the original designer of a service. And finally, services and the data and transaction flows can be easier monitored, analyzed and optimized, which enhances the business control.

In a service-oriented world, the boundaries of applications may disappear. What is currently achieved by monolithic, inflexible applications may be done by “applications” composed, possibly on demand, from a number of services. Services should be designed so that they can be (re)used in any meaningful context. Large legacy applications may well translate into thousands of services, therefore we can imagine enterprises or service marketplaces with millions of available services [5]. This vision implies serious efforts in discovering suitable services, composing them and mediating any data or protocol mismatches. With only the current standard technologies (WSDL [30], SOAP [24] etc.), human programmers are needed to perform these tasks.

For SOA to achieve its benefits on a reasonable scale, it requires discovery, composition and mediation to be meaningful and dynamic [5]. To help automate these tasks, and to free the human resources, we can apply semantic technologies, i.e. ontologies and reasoning. Research on Semantic Web focuses on better searching and data integration capabilities. Service discovery is very similar to search, and mediation uses data integration, therefore we can see how Semantically enabled SOA (or SESA) is an extension of the Semantic Web. Indeed, Berners-Lee describes in his article [2] a vision of Semantic Web that includes Web services, so SESA will be a part of the fulfillment of Berners-Lee’s vision.

In the DIP project, our realization of SESA consists of the following technologies: the Web Service Modeling Ontology [1] (WSMO, [20]) provides a conceptual model for semantic description of Web services. It defines the necessary actors and modeling constructs that are required to capture the semantic information relevant for SESA. The formal semantics of this conceptual model are specified in Web Service Modeling Language (WSML, [6]), with four major dialects in a lattice based on rule languages and description logics. Our reference implementation of WSMO is called

Web Service Execution Environment\textsuperscript{2} (WSMX, \cite{8}), an execution environment for the dynamic discovery, selection, mediation, invocation and interoperation of the Semantic Web Services. The architecture of WSMX is the basis for the standardization work of the Semantic Execution Environment Technical Committee (SEE TC) at OASIS\textsuperscript{3}, therefore in many places in this deliverable we refer to a SEE (Semantic Execution Environment) as opposed to the one concrete implementation, WSMX.

In the following two sections we describe, in turn, the WSMO conceptual model and the components of SEE/WSMX.

\section{WSMO Conceptual Model}

WSMO \cite{20} provides ontological specifications for the core elements of SWS. In fact, SWS aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services.

Appropriate frameworks for SWS need to integrate the basic Web design principles, those defined for the Semantic Web, as well as design principles for distributed, service-orientated computing of the Web. WSMO is therefore based on the following design principles: Web Compliance (WSMO inherits the concept of URI for unique identification of resources, it adopts the concept of Namespaces, supports XML and other W3C Web technology recommendations), Ontology-Based (Ontologies — a widely accepted state-of-the-art knowledge representation — are used as the data model throughout WSMO), Strict Decoupling (WSMO resources are defined in isolation, each resource is specified independently without regard to possible usage or interactions with other resources), Centrality of Mediation (It addresses the handling of heterogeneities that naturally arise in open environments; it can occur in terms of data, underlying ontology, protocol or process), Ontological Role Separation (The underlying epistemology of WSMO differentiates between the desires of users or clients and available services), Description versus Implementation (WSMO differentiates between the descriptions of SWS elements (description) and executable technologies (implementation)), Execution Semantics (in order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO. Service versus Web service (a Web service is a computational entity which is able (by invocation) to achieve a users goal, and a service in contrast is the actual value provided by this invocation; WSMO is designed as a means to describe the former and not to replace the functionality of the latter, i.e. WSMO provides means to describe Web services that provide access to services).

The following briefly outlines the conceptual model of WSMO. The elements of the WSMO ontology are defined in a meta-meta-model language based on the Meta Object Facility (MOF)\textsuperscript{4}. In order to allow complete item descriptions, every WSMO element is described by non-functional properties. These are based on the Dublin Core (DC) Metadata Set \cite{31} for generic information item descriptions, and other service-specific properties related to the quality of service\textsuperscript{5}.

\textsuperscript{2}http://www.wsmx.org/
\textsuperscript{3}Organization for the Advancement of Structured Information Standards, http://oasis-open.org/
\textsuperscript{4}http://www.omg.org/technology/documents/formal/mof.htm
\textsuperscript{5}For a detailed description of all the elements defined in WSMO, we refer the reader to \cite{20}.
Ontologies. Ontologies provide the formal semantics for the terminology used within all other WSMO components. A set of non-functional properties are available for characterizing ontologies; they usually include the DC Metadata elements. Imported ontologies allow a modular approach for ontology design and can be used as long as no conflicts need to be resolved between the ontologies. When importing ontologies in realistic scenarios, some steps for aligning, merging and transforming imported ontologies in order to resolve ontology mismatches are needed. For this reason ontology mediators are used (ooMediators). Concepts constitute the basic elements of the agreed terminology for some problem domain. Relations are used in order to model interdependencies between several concepts (respectively instances of these concepts); functions are special relations, with a unary range and a n-ary domain (parameters inherited from relation), where the range value is functionally dependent on the domain values, and instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values.

Web services. WSMO provides service descriptions for describing services that are requested by service requesters, provided by service providers, and agreed between service providers and requesters. Within the service class the non-functional properties and imported ontologies attributes play a role that is similar to that found in the ontology class, with the addition of an extensible set of Web service-specific non-functional properties. An extra type of mediator to deal with protocol and process related mismatches between Web services is also included.

The final two attributes define the two core WSMO notions for semantically describing Web services: a capability which is a functional description of a Web Service, describing constraints on the input and output of a service through the notions of preconditions, assumptions, postconditions, and effects; and Web service interfaces which specify how the service behaves in order to achieve its functionality. A service interface consists of a choreography which describes the interface for the client-service interaction required for service consumption, and an orchestration which describes how the functionality of a Web Service is achieved by aggregating other Web services.

Goals. A goal specifies the objectives that a client may have when consulting a Web Service, describing aspects related to user desires with respect to the requested functionality and behavior. Ontologies are used as the semantically defined terminology for goal specification. Goals model the user view in the Web Service usage process and therefore are a separate top level entity in WSMO. The requested capability in the definition of a goal represents the functionality of the services the user would like to have, and the requested interface represents the interface of the service the user would like to have and interact with.

Mediators. The concept of Mediation in WSMO addresses the handling of heterogeneities occurring between elements that shall interoperate by resolving mismatches between different used terminologies (data level), on communicative behavior between services (protocol level), and on the business process level. A WSMO Mediator connects elements and provides mediation facilities for resolving mismatches. The description elements of a WSMO Mediator are its source and target elements, and the mediation service for resolving mismatches. WSMO defines different types of mediators for connecting the distinct WSMO elements: OO Mediators connect and mediate heterogeneous ontologies, GG Mediators connect Goals, WG Mediators link Web services to Goals, and WW Mediators connects interoperating Web services resolving mismatches between them.
In order to provide the basis for SWS, WSMO not only provides a conceptual model for describing service-related aspects, but it incorporates this conceptual model in a fully-fledged framework which requires three functional layers: a foundational conceptual model, a formal language to provide formal syntax and semantics (based on different logics in order to provide different levels of logical expressiveness) for the conceptual model, and an execution environment that binds together the several components that use the language to performing various tasks. In this context, the Web Service Modeling Language (WSML) provides a formal syntax and semantics for WSMO, and the Web Service Modeling Execution Environment (WSMX) provides an execution environment which is a reference implementation for WSMO, and offers support for interacting with SWS. WSMX can also be called a Semantic Execution Environment (SEE), and the following section describes the architecture of a SEE.
2.2 SEE Components

Figure 2.1 shows the components of a Semantic Execution Environment, as specified in the WSMX Wiki. The base layer contains the foundations for a SEE, i.e. the formal languages, reasoners, storage and communication channels. The top layer, called Problem Solving Layer, has the domain ontologies, actual applications and developer tools that make use of the SEE. The middle layer, called Broker Layer, contains all the components that handle the various tasks required for SESA automation. On the side, there are components (sometimes also called “vertical services”) that affect the whole SEE.

The following are brief descriptions of all the SEE components. The level of detail of this section is necessary for the coming analysis of how SESA can integrate policies.

First, the “Base layer” contains the common components that are used in the rest of the SEE architecture.

**Formal Languages:** this component represents the different formal languages used for specification of different aspects of knowledge and services. In SESA, the descriptions or of four kinds: static knowledge (ontologies), functional description (capabilities), behavioral description (choreography and composition), and non-functional properties.

**Reasoning:** to work with semantic data encoded in the formal languages from the component above, most of the following components need various sorts of reasoning (logical entailment, query answering, satisfiability checking etc.). For example, discovery needs to check whether Web services can satisfy a given goal, and data mediators need to use rules and other inferences to translate between data in different formats.

**Storage and Communication:** this component provides repositories to store the various kinds of data needed to ensure successful processing of user request to SESA (for example Web service descriptions, compositions, mediator mapping descriptions etc.), and communication channels to pass requests and data between the other components.

The “Broker layer” groups the components that represent distinct functionalities within SESA:

**Discovery:** the purpose of a discovery engine is to find services that can functionally fulfill a user’s goal. This component matches the available semantic Web service descriptions against the goal and filters out the ones that cannot help accomplish the given goal. Discovery also returns services that match only potentially – for instance if the goal is arranging a trip to Puerto Rico, a travel agent service will be discovered even though its description does not say whether or not this particular travel agent also offers trips to the Caribbean.

**Adaptation:** taking the results of discovery, this component checks the partial matches (using negotiation techniques to gather more information about concrete offers) and also takes into account non-functional properties like quality of service (QoS) or price, in the end ranking the available services according to how well they fit the goal and the user’s preferences.

**Composition:** in some situations, especially when no single service can fulfill the given goal, a number of services can be composed to work together. The typical example is the combination of airlines, hotels and car rental agencies to provide a full holiday package. In composition, various parameters of the composed services may have non-trivial dependencies; for instance a hotel that only allows check-in before 10pm cannot be combined with a flight arriving at 9:50pm.

**Choreography:** an interaction with a service often consists of a number of steps which have to be followed in a prescribed order. For instance, a flight reservation service will work in two steps – first reserving a ticket for a given flight, and then purchasing the reserved ticket. In order to automate interaction with the service, the SEE must be able to follow the prescribed steps, more formally known as the service’s choreography interface.

**Mediation:** in any distributed system that is not strictly controlled by a single entity, there will be mismatches both in data formats and in behaviors. Whenever such a mismatch is encountered, the SEE may be able to find or construct appropriate adaptors (mediators); for instance, data in different form can be transformed using predefined mappings. For example, a user may be prepared to pay for the holiday package in Euro but an airline will only accept US dollars – in this case a currency conversion service may be found and put in as a mediator.

**Grounding:** this component is responsible for external communication, i.e. between the client and the SEE, and between the SEE and the Web services. When communicating with Web services based on XML (e.g. SOAP [24]), a SEE has to stoop to the level of XML messages in the formats (schemas) in which the service requires them. Because most of the processing in a SEE likely happens with semantic data (instances of ontologies), this data has to be grounded to XML to be sent to the Web service, and the XML messages must be sent using a network protocol that the service supports. Grounding makes Semantic Web services an extension of Web services, as opposed to their alternative.

**Fault Handling:** at any point during interactions with Web services, especially in complex compositions, various exceptions and faults may occur. The Fault
Handling component takes care of any re-tries, replacements, compensations and clean-ups as appropriate.

**Monitoring:** to enable reporting, process analysis, identification of weak spots and bottlenecks etc., this component monitors and logs the activities within the SEE and its interactions with Web services. Both external services and internal components can be monitored for quality of service (QoS) parameters like performance and reliability, but also for repeating or rare patterns of communication for more advanced analysis.

The “Problem Solving Layer” groups the higher-level components that build upon the lower layers:

**Ontologies:** this component represents the effort of building both foundational and domain ontologies for use in SWS descriptions and in the applications.

**Applications:** this component groups any efforts for developing use case scenarios and domain-specific implementations that use SESA services.

**Developer Tools:** these include tools for managing and manipulating ontologies and descriptions of Web services and goals, and tools for creating mappings between WSMO ontologies, among others.

The two components on the left-hand side of Figure 2.1 affect the whole SEE:

**Execution Management:** this component controls the execution of all the components in the SEE Broker Layer. When a user makes a request to the SEE, the execution manager routes it through all the appropriate tasks, ensuring the execution semantics (cf. [19]) of the system.

**Security:** this component groups all the security interfaces and mechanisms that are necessary in the SEE. More detail on SEE security is not necessary for the purpose of this deliverable.

Naturally, the components are interconnected. Not only do higher-layer components use the lower layers, but even components on the same layer need to cooperate. Some components may share bits of functionality, e.g. both Grounding and Mediation will do data transformations, and both Composition and Choreography will execute process descriptions; the distinctions between components are on high-level functionality and not on how they are implemented. Components may also reuse one another, for instance Discovery may request Composition when no single service can satisfy a goal, and similarly, Composition may request Discovery and Adaptation for finding concrete services that can perform some step in the composition. In fact, the Mediation component is intended to be reused by the other components when necessary – it does not have a standalone function.
3 WEB SERVICES POLICY

In this chapter we give a brief overview of the Web Services Policy Framework (WS-Policy) [22].

The Web Services Policy Framework (WS-Policy) [22] 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 3.1.

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, Figure 3.2 shows a policy requiring the use of Kerberos or X509 security tokens.

WS Policy Framework is accompanied by further specifications:

- WS Policy Attachment [23] specifies two general-purpose mechanisms for associating Policies with one or more policy subjects
- WS Policy Assertions [17] defines several general-use policy assertions

WS-Policy is meant for “expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system” [22]. The distinc-

```xml
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>
```

Figure 3.2: An example policy requiring the use of Kerberos or X509 security tokens
tion 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 [23]) 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.
4 Combining Policy with SESA and WSMO

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

In the first two sections, we define the syntactical constructs necessary to bring WS-Policy together with WSML, 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 4.1 and 4.2.

Further in Section 4.3 we examine the SEE components of a SEE, as described in Section 2.2, 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 4.4.

4.1 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 DIP deliverable D3.12), therefore policies present in a WSDL description will have effect also on Web services specified semantically in a WSML 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\(^1\). 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

\(^{1}\)Please note that in this section we do not distinguish functional properties from behavioral properties, only from non-functional ones. A more detailed discussion on functional vs. non-functional properties can be found in the position paper for the W3C Workshop on Constraints and Capabilities for Web Services [1].
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 4.1 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.

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 currently used by WSMO.

To attach policies generically to XML elements such as WSDL descriptions, the WS-PolicyAttachment specification 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:

```xml
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 WSML, 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:

```xml
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.
4.2 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 4.2 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.

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.\(^2\)

WSMO currently does not have any specific mechanism for expressing that alternative WSMO descriptions are in effect in conjunction with various non-functional properties,\(^3\) 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.

\(^2\)Note that DIP deliverable D4.21 deals with security and trust, but it currently does not use WS-Policy.

\(^3\)See Section 4.1 for discussion of why WSMO treats WS-Policy as non-functional properties.
Because policy assertions in WS-Policy must be XML elements, we can reuse WSML/XML serialization format (see \[6\]) 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 `wsml:wsml`. For finer granularity (e.g. only asserting a single capability description) we can reuse the appropriate elements like `wsml: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 `wsml: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 `wsml:capabilityReference` and `wsml:interfaceReference` that would refer to the identifiers of the WSMO capability or interface descriptions.

The listing in Figure 4.3 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 `wsml: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 `wsml:serviceReference`. Finally, lines 8–12 contain an ontology which is used by the ticketService definition.

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.

Table 4.1 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.

---

4The first `wsml` is a namespace prefix and the second is the name of the XML element container for WSML/XML syntax.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsp#PolicyReference</td>
<td>NFP identifier</td>
<td>non-functional property referring to an external policy file by URI</td>
</tr>
<tr>
<td>wsp#Policy</td>
<td>NFP identifier</td>
<td>non-functional property containing an XML serialization of a policy</td>
</tr>
<tr>
<td>wsml:wsml</td>
<td>XML Element</td>
<td>the root element of WSML/XML syntax, reused as a policy assertion “the embedded WSML description applies (to the policy subject)”</td>
</tr>
<tr>
<td>wsml:webService</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wsml:goal</td>
<td>XML Element</td>
<td>other WSML/XML elements reused as a policy assertion “the embedded WSML description applies (to the policy subject)”</td>
</tr>
<tr>
<td>wsml:descriptionReference</td>
<td>XML Element</td>
<td>policy assertions that refer to WSMO definitions by their identifier URIs</td>
</tr>
</tbody>
</table>

Table 4.1: Syntax for combining WSMO and WS-Policy
4.3 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 is in the same order as in Section 2; here the descriptions explain 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 4.1. 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.\(^5\)

**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\(^6\). Similarly to the Adaptation component, the Composition component would use a WS-Policy evaluator to implement such considerations.

---

\(^5\)Such WSML extensions would fall into the realm of semantically representing policies, which is outside the scope of this deliverable.

\(^6\)For instance [28] presents a coalition policy against using aircraft from a given country in rescue operations.
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. [7, 32]) 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. [3, 27]), 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 Adaptation 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.

Figure 4.4 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.
4.4 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 (cf. [13]) should support attaching policies to WSML descriptions, using the non-functional properties specified within the Formal Languages component (see [21]).

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.

Table 4.2 summarizes these policy-related tasks and the components that are involved:

<table>
<thead>
<tr>
<th>Task</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy creation</td>
<td>Developer Tools, Formal Languages</td>
</tr>
<tr>
<td>Policy matching</td>
<td>Adaptation, Composition</td>
</tr>
<tr>
<td>Policy enforcement</td>
<td>Grounding, Fault Handling, Monitoring, Choreography, Composition</td>
</tr>
<tr>
<td>Policy mediation</td>
<td>Mediation</td>
</tr>
</tbody>
</table>

Table 4.2: Policy-related tasks in SEE components
5 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. [12] “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. [28] 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 [14], [29], and [25]. In [14], 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 [29], 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. Srilharee et al. propose an approach [25] 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 [27] 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 [18], Nejdl et al. propose the use of ontologies to
simplify the tasks of policy specification and administration, discusses how to repre-
sent policy inheritance and composition based on credential ontologies, and formalize
these representations and the according constraints in Frame-Logic. Toninelli et al.
classify in [26] approaches to policy specification as ontology-based approaches (that
depend 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\(^1\) [11, 10] 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 re-
lationships 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 poten-
tially 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 WSML. 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.

\(^1\)http://dig.csail.mit.edu/2006/06/rein/
6 Conclusions and Outlook

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


