SEMANTIC MATCHING OF WEB SERVICE POLICIES

Kunal Verma[1], Rama Akkiraju[2], Richard Goodwin[2]
[1]LSDIS Lab, University of Georgia, Dept of Computer Science, Athens, GA 30605
[2]IBM T. J Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532
verma@cs.uga.edu, {akkiraju, rgoodwin}@us.ibm.com

Abstract: In this work, we present a novel approach for matching the non-functional properties of Web Services represented using WS-policy. To date, most policy matching has been done using syntactic approaches, where pairs of policies are compared for structural and syntactic similarity to determine compatibility. In our approach, we enhance the policies of a Web Service with semantics by creating the policy assertions based on terms from ontologies. The use of semantic terms enables richer representations of the intent of a policy and allows matching of policies with compatible intent, but dissimilar syntax. Also, in cases where policies are defined at multiple levels/domains (such as security, privacy, trust, transactional etc.), we handle inter-domain policy interactions by processing the rules represented in the semantic domain models. This helps identify any inconsistencies in policies during specification as well matching. This approach of using semantic concepts and rules during policy matching leads to better Web Service matches that may not have been possible with syntax based matchers, or prior semantic based methods.

Key words: WS-Policy, Semantic Policy Matching, Ontologies, OWL, SWRL

1. INTRODUCTION

Web Services Policy Framework (WS-Policy) is a general purpose framework for describing capabilities and requirements of Web Service entities. While Web Services Description Language (WSDL) is meant to represent the functional aspects of a Web Service, WS-Policy is used to represent its non-functional attributes. To determine if a Web Service is suitable for a particular use, both the functional and non-functional capabilities and requirements of a Web Service have to be matched with those of a request. Specifically, in matching the non-functional properties, the policy requirements of the entity invoking the Web Service must be compatible with the policies of the entity providing the Web service. In their current specification, both WSDL and WS-Policy are capable of representing the syntactic aspects of the functional and non-functional properties respectively but lack semantics. For example, using WSDL one can describe the interfaces of a service and details on how to invoke it but not what a service actually does. Similarly, using WS-Policy one can describe the policies that a service requires or provides but not the context in which these policies operate. This lack of semantics hampers the effectiveness of computing the compatibility between the policies.

Realizing this need for semantics, the semantic web community [Berners-Lee et al., 1999] is proposing to formalize the representation of meaning (semantics) of content with the help of model theory based RDF [RDF, 1999] and description logics based OWL [OWL, 2002]. In addition, academic projects like OWL-S [OWL-S, 2002], METEOR-S [METEOR-S, 2002] and WSMO [WSMO, 2004] have proposed approaches to semantically represent Web services. Using OWL, OWL-S provides a vocabulary to represent the semantics of both the functional and non-functional aspects of Web Services. However, much of semantic matching work, to date, has been focused on the matching of functional properties of Web Services; namely inputs, outputs, preconditions and effects [Paolucci et al] [Mc.Ilraith et al].

In this paper, we present a novel approach to represent and match the non-functional attributes of Web Services. There are two distinct aspects to our approach. First, we use a combination of OWL and SWRL [SWRL 2004] based rules to capture domain concepts and rules and present a mechanism to enhance policies represented using WS-Policy with semantics. Second, we present an approach to validate such semantically described policies specified across multiple domains (such as security, privacy, trust, transactional, business) during specification and then prescribe an algorithm for policy matching.

The rest of the paper is organized as follows. In Section 2, we present a motivating example to make a case for semantics in representing the non-functional properties of Web Services. Section 3 presents how policies represented using WS-

Policy can be matched today using syntactic matching approach. In Section 4, we describe how WS-Policy can be extended to incorporate semantics. In Section 5 we present our semantic policy matching algorithm. Section 6 provides a comparison of this work with other projects in this area. Our conclusions and future work is outlined in section 7.

2. MAKING A CASE FOR SEMANTICS IN POLICY MATCHING

The current WS-Policy specification relies on XML based vocabularies like WS-Security and WS-Trust to make assertions in those domains. In principle, two parties can make assertions in any number of domains, as long as they have an agreed upon vocabulary. However, if the matching is unaware of the semantics of the assertions, it will be somewhat restricted. In order to illustrate our viewpoint, let us consider the following example. Say that a service provider would like to specify policies at three domains for a specific service. They are:

- *Business domain*: BusinessLevel of requestor must be Enterprise
- *Security domain*: Encryption is required. Can support any of {RSA, SSH and DEC}
- *Security domain*: Security token must be Kerberos V 5.1 and keySize >= 512
- *Transaction domain*: Service execution time is <= 60 msec. Average network response time <= 30 msec.

Similarly, say that there is a requesters' service that has certain policies on its service. They are:

- *Business domain*: Company requesting the service has a Dun & Bradstreet rating of A3.
- *Security domain*: Can supply information encrypted in RSA.
- *Security domain*: Can supply security token Kerberos 5.0 with keySize = 1024
- *Transaction domain*: Expected response time is <= 100 msec.

In the above assertions, if we were to use syntactic matching, the matcher would have no additional knowledge about the domain and it would only perform string matching on the attributes of the assertions. This would lead to false negatives, even though the assertions are equivalent. For example, in the above example a syntactic matcher would not know what relation, if any, the enterprise level status of a company holds to a Dun & Bradstreet rating of a company. Also, the syntax matcher would not know how to match the expected response time of the request service with the information on execution time and network response time given by the service providers' service. Say that now we create a domain ontology to capture the following semantic information:

- If a company has Dun &Bradstreet rating A3 then it is enterprise level
- Kerberos 5.1 can accept Kerberos 4.0 tokens.
- Total Processing time = Service execution time + network response time

- Response time is Equivalent to Total processing time

When this additional domain information along with semantic reasoning is applied, the relationships between seemingly unrelated policies become apparent thereby making it possible to match them correctly. For example, based on a domain rule which states that 'TotalProcessingTime = NetworkTime + executionTime', and with the inference that a response time is equivalent to total time, the matcher can now match the transaction domains of the two policies. Similarly, applying the rule that Dun and Bradstreet (D&B rating) A3 means that the company is enterprise level business domains can now be matched. Also, the compatibility of security tokens is an additional piece of information that a syntax based matcher did not have in a syntax based matcher. This example illustrates how domain knowledge captured as semantic models can improve the quality of matches.

3. WS-POLICY AND WS-POLICY MATCHING

In this section, we briefly describe the WS-Policy specification and provide an overview of how policy matching can be done today in the absence of semantics.

3.1 WS-Policy

WS-Policy provides a grammar for representing the non-functional attributes of entities in a Web services based XML environment [WS-Policy]. A policy is defined as a collection of alternatives and an alternative is defined as a collection of assertions. An assertion is used to represent a requirement, capability or a behavior of a Web service. An assertion can have arbitrary number of child assertions and attributes. A WS-Policy can be represented in XML using the following tags: The "Policy" tag is used to start and end a policy. The "ExactlyOne" tag is used to contain a set of alternatives and the "All" tag contains all the assertions in an alternative. Each assertion belongs to some vocabulary and can have any number of attributes or child assertions.

3.2 Policy Matching

Once a client or a service finds another service with the desired functionality, it must evaluate whether it is compatible by matching the policies. WS-Policy describes a policy normal form which is a disjunction of alternatives and conjunction of all assertions in an alternative. In this paper, we always use the normal form for policy matching.

A policy P is defined as a finite set of alternatives. It can also be expressed as a disjunction of all its alternatives

$$P = \{ Alt1, Alt2, \ldots. Alt\ N \} = Alt1 \mid Alt2 \mid \ldots. Alt\ N$$

An alternative Alt is defined as a finite set of assertions. It can also be expressed as a conjunction of all its assertions.

$Alt = \{A1, A2 \ldots AN\} = A1 \wedge A2 \wedge \ldots. AN$

Matching two policies is reduced to finding two equivalent alternatives.

$((\exists Alt_i)$ S.T. $Alt_i \in P1$ and $(\exists Alt_j)$ S.T. $Alt_j \in P2$ and $Alt_i \Leftrightarrow Alt_j) \Rightarrow (P1 \Leftrightarrow P2)$ ……Rule 1

Finding equivalent alternatives can be defined in the following manner. Two alternatives are equivalent, if for each assertion in both alternatives, there exists an equivalent assertion.

$((\forall A_i)$ S.T. $A_i \in Alt1$ and $(\exists A_j)$ S.T. $A_j \in Alt2$ and $A_i \Leftrightarrow A_j)$ and $((\forall A_i)$ S.T. $A_i \in Alt2$ and $(\exists A_j)$ S.T. $A_j \in Alt1$ and $A1 \Leftrightarrow A2)) \Rightarrow (Alt1 \Leftrightarrow Alt2)$ ..Rule 2

In the current policy framework, equivalent assertions can be computed using syntactical matching of the assertions. In the end, from the application of the above two rules 1 and 2, a working policy must be created from the equivalent alternatives. In some cases, the intersection of policies may also be useful to find common properties and creating a limited working policy. When we apply this approach to match the pair of policies described in section 2, we note that the business domain and transaction domain cannot be matched based on syntax alone. In the next section we show how we extend WS-Policy to capture semantic annotations and how policy matching can be achieved using this newly available semantic information.

4. EXTENDING WS-POLICY TO INCORPORATE SEMANTICS

Policies must be augmented with semantic information in order to enable semantic matching. Below we describe how we use domain ontologies for creating policy assertions with semantics. As mentioned in section 3, assertions are used to represent a requirement or a behavior of Web services in any number of domains. The type of the assertion is specified by using the QName [XML-Namespaces, 1999] specified in a domain specific vocabulary. A sample assertion is shown in Figure 1.

```
<wsse:SecurityToken>
<wsse:TokenType>wsse:Kerberosv5TGT
</wsse:TokenType>
```

</wsse:SecurityToken>

Figure 1: Sample Assertion

This assertion is a security assertion, which is specified by the "wsse", which is the URI part of the name of the root element of the assertion. In the WS-Policy specification, this URI is mapped the URI of the OASIS site hosting the XML schema description of WS-Security. The local part of the name of root element "SecurityToken" specifies that it is a security token assertion. It has a child assertion which states that the token type of the security token should be Kerberosv5TGT.

For adding semantics to WS-Policy, we recommend using OWL ontologies instead of XML schema based vocabularies like WS-Security. Consider an OWL ontology, which semantically describes all the elements of WS-Security using the more expressive description logics constructs available in OWL ontologies. The fact that OWL ontology concepts can be referenced through URIs allows us to use concepts from the ontologies directly in the policy assertions. In Figure 2, the URI part of QName "semsecurity:SecurityToken", is mapped to an URI hosting a security ontology in OWL, which has semantic descriptions of SecurityToken, Kerberosv5TGT and TokenType.

In addition to creating assertions from domain ontologies, we propose adding three optional attributes to each assertion for explicating the semantics of the assertion. They are:

1. assertionType: This attribute is used to specify whether an assertion is a requirement or a capability. The current specification states an assertion could be either a requirement or a capability of a Web service. As a result during matchmaking, all assertions have to be matched and are considered as both a capability and requirement. This makes the matching cumbersome. Explicitly categorizing an assertion as either a requirement or a capability resolves this ambiguity and makes the matching simpler and cleaner. The assertions in Figure 2 are requirements. The default value of assertionType is requirement.

2. valueType: This attribute is used to specify whether an assertion is a numeric or a non-numeric assertion. The value owl:object specifies that the assertion is belongs to an OWL ontology and OWL based operators can be used to reason about it. The value xsd*:type* specifies that it a literal, where *type* can be any data type (xsd:string, xsd:float, xsd:int, etc.) supported by XML schema. . We use this attribute to decide whether to use numeric/string based comparisons or description logics based reasoning for matching the particular assertion. By default, an assertion is xsd:string, and string matching is used . The assertions in Figure 2 are non numeric.

3. comparisonOperator: This attribute is used to represent the relationship between the assertion name and value. For example the child assertion in Figure 1 states that "TokenType = Kerberosv5TGT". WS-Policy assumes an "equals-to" relationship. While the "equals-to" is the default value we allow "greater than", "less than",

"greater than or equal to", "less than or equal to" for number numeric assertions and "subclassof", "superclassof", "instanceof" for non numeric assertions.

```
<semsecurity:SecurityToken
  assertionType="sempolicy:Requirement"
  valueType="owl:object"
  comparisonOperator="sempolicy:EQ">
  <semsecurity:TokenType
 assertionType="sempolicy:Requirement"
     valueType="owl:object"
     comparisonOperator="sempolicy:EQ">
     semsecurity:Kerberosv5TGT
  </semsecurity:TokenType >
</semsecurity:SecurityToken >
```

Figure 2: Assertion using ontologies

5. WEB SERVICE POLICY MATCHING USING SEMANTICS

In this section, we will describe how semantic matching can help match the policies of the request and advertisement shown in section 2 with the help of domain ontologies and rules.

5.1 Policy and Rules Representation

To reason about domain ontologies, we use a semantic network based ontology management system known as SNoBASE [Lee et al., 2003] that offers DQL-based [DQL, 2003] Java API for querying ontologies represented in OWL. SNOBASE uses IBM's ABLE [Bigus et al 2002] engine for inferencing. We have created an OWL ontology for WS-Policy. This allows to represent individual policies as OWL instances. We implemented a module to add SWRL rules to SNoBASE. Consider the following rules, and their SWRL and ABLE representations.

If there exists a policy P, which has an alternative ALT, which has an Assertion A, which states that "DunAndBradStreetRating = A3", **then** create a new Assertion A1 which states "BusinessLevel=Enterprise" and belongs to Alternative Alt. This rule can be represented in SWRL abstract syntax as:

Policy (P) and hasAlternative(P, ALT) and hasAssertion (ALT, A) and DunAndBradStreetRating(A, "A3") => Assertion (A1) and BusinessLevel (A1, "Enterprise") and hasAssertion (ALT, A1)

The ARL "when-do" for the above SWRL rule is given below:

when: Policy (P) and hasAlternative (P, ALT) and hasAssertion (ALT, A)

and DunAndBradStreetRating(A, "A3")

do: Assertion (A1) and BusinessLevel (A1, "Enterprise") and hasAssertion (ALT, A1)

If there exists a policy P, which has an alternative ALT, which has an Assertion A1, which states that "ExecutionTime = X" and an Assertion A2, which states that "NeworkTime = Y", **then** create a new Assertion A3 which states that TotalProcessingTime = X + Y. This rule can be represented in SWRL syntax as:

Policy (P) and hasAlternative (P, ALT) and hasAssertion (ALT, A1) and hasAssertion (ALT, A2) and ExecutionTime(A1, X) and NetworkTime (A2, Y) => Assertion (A3) and TotalProcessingTime (A3, X + Y) and hasAssertion (Alt, A3)

This rule can be written in ARL as:

when: Policy (P) and hasAlternative (P, ALT) and hasAssertion (ALT, A1) and hasAssertion (ALT, A2) and ExecutionTime(A1, X) and NetworkTime (A2, Y)

do: Assertion (A3) and TotalProcessingTime (A3, X + Y) and hasAssertion (Alt, A3)

Rules are used in ABLE using a RETE [Forgy, 1982] network and new facts are added to the policies if the right conditions exist for firing the particular rules. A key challenge was to model these rules over OWL facts. However, the ABLE based implementation of SNOBASE, allowed us to write these in Able Rules Language. Similar rules can be written to invalidate alternatives, if the inter-domain assertions do not make sense or contradict each other. For example, if a security algorithm assertion and compression algorithm assertion are in the same alternative, and they cannot co-exist, then the alternative can be invalidated, by having a rule to do so.

5.2 Semantic Policy Matching Algorithm

In this section, we will explain our matching algorithm. As discussed in section 3.2, matching two policies is reduced to finding two equivalent alternatives. However, based on requirements and capabilities, we present a different definition for equivalent alternatives. Equivalent alternatives can be defined as alternatives whose capabilities satisfy each others requirements. In order to write the definition, we define functions "req" and "cap", which represents requirement and capability assertions of an alternative respectively.

$((\forall A_i)$ S.T. $A_i \in$ req (Alt1) and $(\exists A_j)$ S.T. $A_j \in$ cap (Alt2) and A_i satisfies A_j) and $((\forall A_i)$ S.T. $A_i \in$ req (Alt2) and $(\exists A_j)$ S.T. $A_j \in$ cap(Alt1) and A_i satisfies A_j)) $\Rightarrow$ (Alt1 $\Leftrightarrow$ Alt2)

In order to explain the satisfiability of one assertion by another we define the following functions.

"val" returns the value of an assertion or an attribute
"type" returns the type of an assertion or an attribute
"children" returns all the children of an assertion
"attributes" returns all attributes of an assertion

A capability assertion satisfies a requirement assertion if its value satisfies the requirement assertion's value and if all the attributes of the requirement assertions are satisfied by its attributes. In addition, all the child assertions of the capability assertion, must satisfy the child assertions of the requirement assertions.

[val (A2) satisfies val (A1) and

$(\forall\, attr_i)$ S.T. $attr_i \in$ attributes(A1) and $(\exists\, attr_j)$ S.T. $attr_j \in$ attributes (A2) and type($attr_i$) = type ($attr_j$) and val ($attr_j$) satifisfies val ($attr_i$)) and

$((\forall\, child_i)$ S.T. $child_i \in$ children(A1) and $(\exists\, child_j)$ S.T. $child_j \in$ children (A2) and $child_i$ satisfies $child_j$)] $\Rightarrow$ (A2 satisfies A1)

Following operators are used to check value or attribute satisfiability

1. For XML type based assertions: equals, =, <, >, <=, >=
2. For OWL based assertions: sameClassAs, sameInstanceAs, subClassOf, instanceOf, superClassOf, subsumes

5.3 An Example of Semantic Policy Matching

We use the same example introduced in section 2. Below, we categorize the policies of the provider into requirements and capabilities.

Provider Requirements:

- *Business domain:* BusinessLevel of requestor must be Enterprise
- *Security domain*: Security Token must be Kerberos 5.1 and keySize >= 512
- *Security domain*: Encryption should be used

Provider Capabilities:

- *Transaction domain*: NetworkTime <= 30 ms & ExecutionTime <= 60 ms
- *Security domain*: Can support any of {RSA, SSH and DEC}

Based on the domain knowledge in rules and ontologies, the original policy is augmented with new facts. The following rule is fired when the above assertions are added to the policy object

ExecutionTime(P) + networkTime (P) = TotalProcessingTime (P)

Hence, a new assertion is generated

- TotalProcessingTime <= 90 ms

Due to the assertions in the domain ontology

TotalProcessingTime *sameAs* ResponseTime

Another, new assertion is generated

- ResponseTime <= 90 ms

Similarly, we categorize the policies of requester in section 2 as follows:

Requestor Requirements:

- *Security domain*: Can supply security token Kerberos 5.0 with keySize = 1024
- *Security domain*: Encryption Algorithm must be RSA
- *Transaction domain*: ResponseTime <= 100 ms

Requestor Capabilities:

- *Business domain*: DunAndBradStreetSize is A3

The following rule is fired when the above assertions are added to the policy object**.**

DunAndBradStreetSize(P, A3) => businessLevel(P) = Enterprise

Hence, a new assertion is generated

- BusinessLevel is Enterprise

Our matchmaking framework can now match the policies of the requestor and the provider with the help of the new facts that have been added to both the policies.

Checking all requirments of policy 1:

1. BusinessLevel (P1 , Enterprise) is satisfied by assertion BusinessLevel (P1 , Enterprise) as Enterprise = = Enterprise
2. SecurityToken (P1, Kerberos 5.1) is satisfied by assertion SecurityToken (P2, Kerberos 5.0) as Kerberos 5.0 is substituteable for Kerberos 5.1 and keysize (1024 >= 512)
3. EncryptionAlgorithm (P1, EncryptionAlgorithm) is satisfied by assertion EncryptionAlgorithm (P1, RSA) as RSA *subClassOf* EncryptionAlgorithm

Therefore, all requirements of policy 1 are satisfied by policy 2.

Checking all requirements of policy 2:

1. ProcessingTime (P2 , <=, 100ms) is satisfied by assertion ProcessingTime (P1 , <=, 90ms) as 90ms <= 100ms

Therefore, all requirements of policy 2 are satisfied by policy 1.

Since all requirements of both policies are satisfied by each other, the matcher declares these policies as being equivalent.

6. RELATED WORK

Much of the previous work on policy matching has been based on syntactical models. Wohlstadter et al [Wohlstadter et al., 2004] extend the grammar of WS-Policy to add qualifying conditions and numerical predicates, but is still based on syntactical domain models. Having XML based models limits the expressivity of the assertions and also limits the matching to syntactical matching. Our work addresses these limitations by using OWL based domain ontologies along with SWRL like rules. This allows our matcher to use rich domain knowledge for better matching. [Mukhi and Plebani, 2004] discuss issues in WS-Policy based frameworks. One of the problems mentioned in their work is capturing and reasoning on inter domain dependencies between assertions of different domains. Once again, such

dependencies can be captured using ontologies and rules and used in the matchmaking.

There has also been some work in policies using Semantic Web technologies. [Uszok et al., 2004] have developed the KAOS system for representing policies using OWL. Their approach is similar to ours, but we use SWRL rules, instead of value-maps used by them. [Kagal et al., 2004a] use a rule based engine for handling trust and privacy of Web services. In another paper [Kagal et al., 2004b] the authors have discussed the interaction of OWL ontologies and SWRL rules as an open problem. The approach discussed in this paper allows us to combine OWL ontologies and SWRL rules, by representing only those aspects of which are beyond the expressivity of description logics to OWL. [Li and Horrocks, 2004] provides an approach for matching non-functional attributes, but their framework is restricted as they rely solely on subsumption for the matching and their expressivity is limited by description logics. While our approach also takes care of hierarchical relationships (the basis for subsumption), it is more flexible we can use either subsumption based reasoning as well as domain rules for matching. In addition, we support relationships (inter and intra domain) using rules which cannot be representing using description logics alone. [Parsia et al., 2005] have developed an OWL ontology for representing WS-Policy. However, they have not considered adding SWRL rules to the assertions, which is a key contribution of this approach.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an approach for matching non-functional requirements of Web services based on creating rich domain models using ontologies and rules. The research contributions of this paper are providing a extensible domain crossing approach for semantic policy matchmaking as well as providing an approach for implementing horn logic based rules to be used in conjunction with OWL based ontologies. Our implementation acts as a proof of concept and shows the usefulness of such an approach.

REFERENCES

[Berners-Lee et al., 1999] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, May 2001.

[Bigus et al., 2002] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, ABLE: A toolkit for building multiagent autonomic systems, IBM Systems Journal, Volume 41, Number 3, 2002

[DQL, 2003] DQL Technical Committee 2003. DAML Query Language (DQL) http://www.daml.org/dql

[Forgy, 1982] Charles L. Forgy, Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Artificial Intelligence 19 (1982), 17-37)

[Grosof et al., 2004] B. N. Grosof, I. Horrocks, R. Volz, S. Decker: Description logic programs: combining logic programs with description logic, The Proceedings of the World Wide Web Conference (WWW2003), 2003, pp:48-57

[Kagal et al., 2004a] L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara, Authorization and Privacy for Semantic Web Services, The Proceedings of the AAAI Spring Symposium on Semantic Web Services, 2004

[Kagal et al., 2004b] Lalana Kagal, Tim Finin, and Anupam Joshi, Declarative Policies for Describing Web Service Capabilities and Constraints, Proceedings of W3C Workshop on Constraints and Capabilities for Web Services

[Lee et at., 2003] Lee J., Goodwin R. T., Akkiraju R., Doshi P., Ye Y. SNoBASE: A Semantic Network-based Ontology Ontology Management. http://alphaWorks.ibm.com/tech/snobase. 2003

[Li and Horrocks, 2004] L. Li and I. Horrocks, A software framework for matchmaking based on semantic web technology, The Proceedings of the World Wide Web Conference (WWW2003), 2003, pp:331-339

[METEOR-S, 2002] METEOR-S: Semantic Web Services and Processes, http://lsdis.cs.uga.edu/Projects/METEOR-S/

[Mukhi and Plebani, 2004] N. K. Mukhi and P. Plebani, Supporting Policy-driven behaviors in Web services:Experiences and Issues, To appear in the proceedings of the International Conference on Services Oriented Computing (ICSOC), 2004

[OWL, 2002] OWL Technical Committee (T.C). 2002. Web Ontology Language (OWL). http://www.w3.org/TR/2002/WD-owl-ref-20021112/

[OWL-S, 2002] OWL-S Technical Committee (T.C). 2002. Web Ontology Language for Web Services. http://www.daml.org/services/owl-s/

[Parsia et al., 2005] B. Parsia, V. Kolovski, and J. Hendler. Expressing WS-Policies in OWL. Submitted to Policy Management for the Web Workshop, 14th International World Wide Web Conference, Chiba, Japan, May 2005.

[RDF, 1999] Resource Data Framework, http://www.w3.org/RDF/

[Sivashanmugam et al., 2003] K. Sivashanmugam, K. Verma, A. P. Sheth, J. A. Miller: Adding Semantics to Web Services Standards, The Proceedings of the First International Conference on Web Services (ICWS 2003), 2003, pp:395-401

[SWRL, 2004] Semantic Web Rule Language, http://www.daml.org/2003/11/swrl/

[Uszok et al., 2004] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, S. Aitken, Policy and Contract Management for Semantic Web Services, The Proceedings of the AAAI Spring Symposium on Semantic Web Services, 2004

[Wohlstadter et al., 2004] E. Wohlstadter , S. Tai , T. Mikalsen , I. Rouvello , and P. Devanbu, GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions, The Proceeding of the 26th International Conference on Software Engineering (ICSE 2004), pp. 189-199

[WSMO, 2004] Web Service Modeling Ontology, http://www.wsmo.org

[WS-Policy, 2003] The Web Service Policy Framework
http://www-106.ibm.com/developerworks/library/ws-polfram/

[XML-Namespaces, 1999] Namespaces in XML, World Wide Web Consortium 14-January-1999, http://www.w3.org/TR/REC-xml-names/