Expressing WS Policies in OWL

Bijan Parsia
Maryland Information and
Network Dynamics Laboratory
University of Maryland
College Park, MD 20742
bparsia@isr.umd.edu

Vladimir Kolovski
Dept. of Computer Science
University of Maryland
College Park, MD 20742
kolovski@cs.umd.edu

Jim Hendler
Maryland Information and
Network Dynamics Laboratory
University of Maryland
College Park, MD 20742
hendler@cs.umd.edu

ABSTRACT
In this paper, we present two translations of the Web Service Policy Framework (WS-Policy) into OWL-DL. First, we provide an introduction to WS-Policy and we argue the benefits of using OWL and RDF to express web service policies. Then, we provide two translations from WS-Policy to OWL, one of them representing policies as instances, and the second one as classes. Finally, we provide a survey of existing web policy languages and a general idea of their expressivitiy.

Categories and Subject Descriptors
I.2.4 [**Artificial Intelligence**]: Knowledge Representations and Formalisms – Semantic Web, *OWL, RDF, Web Service Policy.*

General Terms
Performance, Design, Standardization, Languages.

Keywords
Web services, Policy languages, OWL, RDF, WS-Policy

1. Introduction
Web services interact with each other by exchanging SOAP messages. To provide for a robust development and operational environment, services are described using machine-readable metadata. This metadata serves several purposes, one of them being describing the capabilities and requirements of a service — often called the service *policy.* In recent years, there have been many different web service policy language proposals, all of them describing languages with varying degrees of expressivity and complexity. However, with most current proposals it is difficult to determine their expressivity and computational properties as most lack formal semantics. One characteristic of the proposed languages is that they involve policy *assertions* and combinations of assertions. For example, a policy might assert that a particular service requires some form of reliable messaging or security, or it may require both reliable messaging and security. Several industrial proposals (e.g., WS-Policy [13] and Features and Properties [2]) appear to restrict them to a kind of *propositional* logic with policy assertions being atomic propositions and the combinations being conjunction and disjunction. By mapping the policy language constructs into a logic (e.g., some variant of first order logic) we can acquire a clear semantics for the policy languages, as well as a good sense of the computational aspects of the languages.

If we can map the policy languages into a standardized logic, then we can benefit from the tools and general expertise one expects to come with a reasonably popular standard. By mapping two policy languages into the same background formalism, we will be able to provide some measure of interoperability between policies written in distinct languages. If we are smart in our mapping, we should also be able use pre-existing reasoners for the standardized logic to do useful reasoning about policies.

Our language of choice is the Web Ontology Language, OWL [4], and the Resource Description Framework (RDF [6]). Both RDF and OWL are strict subsets of first order logic, with the subspecies OWL-DL being a very expressive yet decidable subset. OWL-DL builds on the rich tradition of *description logics* where the tradeoff between computational complexity and logical expressivity has been precisely and extensively mapped out and practical, reasonably scalable reasoning algorithms and systems have been developed.

In this paper, we have mapped one of the policy languages, WS-Policy, to OWL-DL. WS-Policy is a policy language being developed by IBM, Microsoft, BEA, and other major web services vendors and is generally considered to be the policy language with the most momentum. We have chosen two approaches: expressing policies as instances, and expressing them as classes. With the latter, we are able to use our OWL-DL reasoner, Pellet [8] as a policy engine with analysis services that go far beyond what is usually offered. In the next section we describe our mappings.

2. Mappings
Our implementation consists of two different translations, one being where the WS-Policy grammar is encoded in OWL and the other where we are trying to capture the formalism underlying the WS-Policy grammar. In the first case, individual policies are translated to OWL-DL instances, whereas in the second case they are translated into OWL-DL class expressions. This is no surprise as WS-Policy is pretty clearly intended to be a subset of propositional logic and OWL-DL is propositionally closed.

2.1. Policies as Instances

The first ontology is an attempt at designing an OWL ontology that accurately reflects the WS-Policy grammar which is originally expressed as an XML Schema. This translation essentially captures the syntax of WS-Policy, but not its semantics.

As mentioned before, WS-Policy introduces a simple grammar for expressing policy assertions. These assertions allow developers to add metadata to service description at development time or at runtime. Examples of development time policy would include a specification of which character encodings are supported, or which specifications, and which versions of those specifications are supported by the service. An example of runtime policy would include interruption in the availability of the Web service due to system maintenance.

Assertions are the building block of a Web service policy and satisfying them usually results in a behavior that satisfies the conditions for the service endpoints to communicate. A policy assertion is supported by a requestor if and only if the requestor satisfies the requirement, or accommodates the capability, corresponding to the assertion. Policy assertions usually deal with domain-specific knowledge, and they can be grouped into policy alternative. An alternative is satisfied only if the requestor of the service satisfies all of the policy assertions contained in the alternative. Note that in our ontology policy assertions and alternatives are represented with separate OWL classes related with the **containsAssertions** property. Determining whether a policy alternative is supported is done automatically using the results of the policy assertions.

A policy is supported by a requestor of a service if the requestor satisfies at least one of the alternatives in the policy. Once the policy alternatives have been evaluated, it can be automatically deduced whether a policy is supported by the requestor.

There are two operators used to express relations between policies, alternatives and assertions: All and ExactlyOne. These operators are implemented as OWL classes **OperatorAll** and **OperatorExactlyOne** in our ontology. OperatorAll requires all the assertions to hold in order for the policy alternative to be satisfied. OperatorExactlyOne specifies that exactly one of the assertions has to hold in a collection of policy alternatives for the policy assertion to be satisfied.

In order to illustrate our work, we present an OWL version of a policy requiring the web service to use X.509 certificates or Kerberos tickets as security token types.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:wsp="http://www.mindswap.org/~kolovski/ws-policy.owl#"
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
         secext-1.0.xsd">
<wsp:Policy>
  <wsp:constrainedBy>
    <wsp:OperatonExactlyOne>
      <wsp:constrainedBy>
        <wsse:SecurityToken wsp:Preference="100">
          <wsp:Usage rdf:resource="http://www.w3.org/2004/08/20-ws-pol-pos/ns#Required"/>
          <wsse:tokenType>wsse:Kerberosv5TGT</wsse:tokenType>
        </wsse:SecurityToken>
      </wsp:constrainedBy>
      <wsp:constrainedBy>
        <wsse:SecurityToken wsp:Preference="1">
          <wsp:Usage rdf:resource="http://www.w3.org/2004/08/20-ws-pol-pos/ns#Required"/>
          <wsse:tokenType>wsse:X509v3</wsse:tokenType>
        </wsse:SecurityToken>
      </wsp:constrainedBy>
    </wsp:OperatolExactlyOne>
  </wsp:constrainedBy>
</wsp:Policy>
</rdf:RDF>
```

Listing 1. RDF representation of a policy using a WS-Policy grammar expressed in OWL[1].

The approach described above gives us a clear way of expressing the syntax of WS-Policy in OWL. This approach has its advantages, as described in Section 2. However, the previous ontology does not capture the semantics of WS-Policy, so it is

[1] The ontology capturing the WS-Policy grammar is available at http://mindswap.org/dav/ontologies/ws-policy_instance.owl

difficult, for example, to determine whether two policies are consistent with each other. The following translation does a better job of capturing the semantics of WS-Policy.

2.2. Policies as Classes

Our second translation maps the WS-Policy formalism directly in OWL. We accomplish that by translating the WS-Policy constructs from a normal form policy expression into OWL constructs. A normal form policy expression is a straightforward XML Infoset representation of a policy, enumerating each of its alternatives that in turn enumerate each of its assertions. Following is a schema outline for the normal form of a policy expression:

```
<wsp: Policy…>
  <wsp:ExactlyOne>
    [ <wsp:All> [<Assertion…> … </Assertion>]* </wsp:All> ]*
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 2. Normal form of a policy expression

Policy expressions can also be represented in more compact forms, using additional operators such as **wsp:Optional**, however as shown in [13] the policy expressions can all be expanded to normal form. Therefore we only provide a mapping of the constructs used in a normal form policy expression: **wsp:ExactlyOne** and **wsp:All**.

First, we map policy *assertions* directly into OWL-DL atomic classes (which correspond to atomic propositions). Though WS-Policy assertions often have some discernible substructure, it is not key to their logical status in WS-Policy. Or rather, that substructure is idiosyncratic to the assertion set, rather than being a feature of the background formalism. So a general WS-Policy engine must be adapted to deal with their structure, if it is to do so. The WS-Policy specification asserts:

> "Assertions indicate domain-specific (e.g., security, transactions) semantics and are expected to be defined in separate, domain-specific specifications."

It seems unfortunate that each domain-specific specification comes with its own domain specific syntax. If we are to capture the semantics of each assertion language, we must separately map each assertion language into OWL. Our default of treating each assertion as a simple atomic proposition is reasonable for general policy manipulation, since a general purpose policy engine will work roughly the same way.

Mapping wsp:All to an OWL construct is straightforward because wsp:All means that all of the policy assertions enclosed by this operator have to be satisfied in order for communication to be initiated between the endpoints. Thus, it is a logical conjunction and can be represented as an intersection of OWL classes. Each of the members of the intersection is a policy assertion, and the resulting class expression (using the operator **owl:intersectionOf**) is a custom-made policy class that expresses the same semantics as the WS-Policy one.

Handling wsp:ExactlyOne might be trickier, depending on the interpretation of the construct. There are two possible interpretations:

a) wsp:ExactlyOne means that a policy is supported by a requester if and only if the requester supports at least one of the alternatives in the policy. In the previous version of WS-Policy there was a wsp:OneOrMore construct capturing this meaning. In such case, the wsp:ExactlyOne is an inclusive OR , and can be mapped using owl:unionOf.

b) The other interpretation is that wsp:exactlyOne means that only one, not more, of the alternatives should be supported in order for the requester to support the policy. This is supported by [13], where it's stated that although policy alternatives are meant to be mutually exclusive, it cannot be decided in general whether or not more than one alternative can be supported at the same time. Our translation covers this more complicated case.

Wsp:ExactlyOne can be translated in OWL in the following way: for n different policy assertions, expressed as OWL classes themselves, wsp:ExactlyOne is the class expression consisting of the members of each separate policy class that do NOT also belong to another policy class. In OWL terms, it is the union of all of the classes with the complement of their pair-wise intersections. Because of the pair-wise intersections there is a quadratic increase in the size of the OWL construct that is used as a mapping for wsp:ExactlyOne. Following is a table summarizing both of the translations:

Table 1. Mapping WS-Policy to OWL

WS-Policy Construct	OWL Expression
Wsp:All (policies A and B)	intersectionOf (A B)
Wsp:ExactlyOne (policies A and B)	intersectionOf(complementOf (intersectionOf (A B)) unionOf (A B))

In order to illustrate how the mapping of wsp:All and wsp:ExactlyOne works, we present a sample policy ontology. The general WS-Policy Assertions are stored as OWL classes, for example there is a SecurityTokenType class with subclasses KerberosTicket, UsernameToken and X509Certificate. Other assertions are stored, too: Language, Messaging, SpecVersion and TextEncoding. Figure 1 illustrates the WS-Policy class hierarchy.

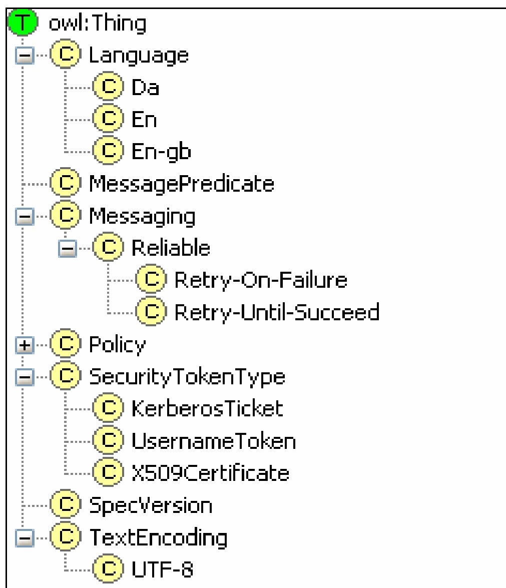

Figure 1. Sample Policy Ontology

Having stored a part of WS-PolicyAssertions [14] as OWL classes, now it's possible to develop our own custom policies. For example, say we wanted a policy such that the requestor supports Kerberos tickets and reliable messaging. Those two conditions can be represented as two assertions in a policy alternative, implying that they can be mapped to an owl:intersectionOf. The corresponding OWL expression shown in Figure 2.

```
<owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#GeneralReliabilityKerberosPolicy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Reliable">
        </owl:Class>
        <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Policy">
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 2. Wsp:All representation in OWL

For the wsp:ExactlyOne example, we consider a policy that expects the requestor to provide either a Kerberos ticker, or an X509 certificate, but not both. In OWL, it would be represented by the class expression composed of the elements that are exclusive to KerberosTicket and X509Certificate. Figure 3 represents a serialization of the class expression in RDF/XML.

```xml
<owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosOrX509Policy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
            </owl:Class>
            <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#x509Certificate">
            </owl:Class>
          </owl:unionOf>
        </owl:Class>
        <owl:Class>
          <owl:complementOf>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#X509Certificate">
                </owl:Class>
                <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
                </owl:Class>
              </owl:intersectionOf>
            </owl:Class>
          </owl:complementOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Policy">
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 3. Wsp:ExactlyOne representation in OWL

2.3. Policy processing

One of our arguments for expressing policies using OWL was the ability to reason about policy containment – whether the requirements for supporting one policy are a subset of the requirements for another. That would allow us to be more flexible in determining whether a particular requestor supports a policy, in the cases where the requestor supports a superset of the requirements established by the policy.

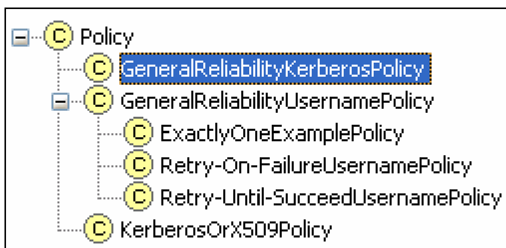

Figure 4. Example of policy containment

In the previous figure, Retry-On-FailureUsernamePolicy is an intersection of Retry-On-Failure and UsernameToken. However, since Retry-On-Failure is a subclass of Reliable, our OWL-DL reasoner [8] classifies Retry-On-FailureUsernamePolicy as a subclass of GeneralReliabilityUsernamePolicy , meaning that any user that supports the latter will also support the former.

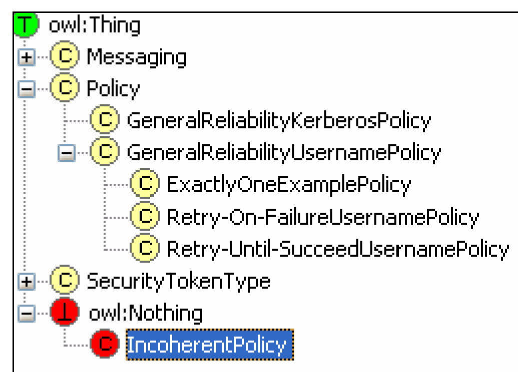

Figure 5. Example of policy incoherence

The above figure is an example of Swoop showing an incoherent policy. IncoherentPolicy selects two policy alternatives from an wsp:ExactlyOne, which, according to our current translation is forbidden. Note that Swoop displays an explanation of the incoherence, which can aid in repairing the policy.

In general, we get the following inferences out of the box:

1) policy inclusion (if x meets policy A then it also meets policy B; a.k.a., A rdfs:subClassOf B);
2) policy equivalence (A owl:equivalentTo B);
3) policy incompatibility (if x meets policy A then it cannot meet policy B; a.k.a, A owl:disjointWith B);
4) policy incoherence (nothing can meet policy A; a.k.a., A is unsatisfiable)
5) policy conformance (x meets policy A; a.k.a, x rdf:type A)

Some care must be taken given the open world semantics of OWL. For example, an OWL reasoner does not assume that because it cannot prove that x conforms to policy A, that x *does not* conform to policy A. It is unclear what the WS-Policy authors intend, though a closed world assumption is not unlikely. However, even if there is a closed world assumption on WS-Policies, we can handle at least some of those cases by adding explicit disjoint statements at translation time.

One futher reasoning service supported by Pellet, and integrated with Swoop, is *explanations* for inconsistencies,[16] which can be used to help debug policy incompatibility, incoherence, and the like. As we add further explanation capability to our systems, this debugging power will grow.

Thus we see that with a fairly simple mapping, we can use an off the shelf OWL reasoner as a policy engine and analysis tool, and an off the shelf OWL editor as a policy development and integration environment. OWL can also used to develop domain specific assertion languages (essentially, domain Ontologies) with a uniform syntax and well specified semantics. We can also experiment with extensions to WS-Policy, by using more expressive constructs from OWL at the policy language, as well as the assertion language, level. We can play with extensions before having to write a yet another processor for them. Of course, if it turns out that we really want to restrict ourselves to a very inexpressive subset, then we may still want to build specific reasoners and processors that are tuned for that sublanguage. But there again, our tools can help us. Pellet does expressivity analysis of ontologies, so can help determine what logic we are really using and the price of extensions.

Furthermore, ontology development techniques can be useful for policy development as well. Most human generate ontology develop iteratively, with specializations added to the class tree over time. Similarly, we can build up our policies from more general ones. A general policy could be very restrictive, setting tough guidelines for all of a companies policies.

Finally, given a similar style mapping for another policy language (say, Features and Properties, described in the next section) we can do policy analysis and integration across policy languages.

3. Other Policy Languages

In this section we provide a quick overview of the state-of-the-art in web service policy specification, by looking at the policy languages presented at [12]. To the best of our knowledge, they are sorted by increasing level of expressivity, even though lack of formal semantics and analysis hampered our effort to provide a fully correct listing. The list follows:

3.1. The Platform for Privacy Preferences Project (P3P) [9] enables Web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents. P3P user agents allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices when appropriate. According to [15], there exists a data-centric relational semantics for P3P in which a P3P policy is modeled as a relational database. This simple semantics allows us to express P3P using RDF.

3.2. The Features and Properties architecture [2] originates from SOAP 1.2, and was integrated into WSDL 2.0 in order to support the SOAP-specific concepts. Afterwards the architecture was further expanded in order to allow Features to be more abstract. Simply put, a Feature identifies a piece of functionality, identified by a URI. An example of a Feature would be encryption. Properties are the parameters of a Feature, also identifiable by a URI. For an encryption feature, property might be the algorithm used, part of message encrypted, etc. Features & Properties are similar to WS-Policy in terms of expressivity, with one exception – they lack operators for combining policy assertions. It is argued at [3] that adding a choose one/all operators (called combinators) will prove to be useful in expressing higher-level semantics combining multiple policies.

3.2. WS-Policy [13] provides a general-purpose model and syntax to describe and communicate the policies of a Web service. It specifies a base set of constructs that can be used and extended by other Web service specifications to describe a broad range of service requirements and capabilities. WS-Policy introduces a simple and extensible grammar for expressing policies and a processing model to interpret them. The policy assertions are expressing using XML and the grammar itself is specified with XML Schema.

By using OWL we increase the expressiveness of the WS-Policy representation and it will simplify the interaction between any new protocols on one hand, and WS-Policy and WSDL on the other hand. By using OWL/RDF we do not need to focus on the ways in which the policy is attached to the web service, instead we can concentrate on analyzing the policy itself.

Also, WS-Policy uses an open content model on policy assertions to provide extensibility, and the usage of OWL and RDF can provide more expressiveness by way of subclass and subproperty constructs – re-using of derived policy assertions.

[10] provides a comparison of XML and RDF in terms of expressing WS-Policies. It also provides arguments for usage of RDF to represent WS-Policy by describing how RDF meets document merging and extensibility goals described in the WS-Policy specifications. To support this, the paper presents an RDF schema for representing web service policies upon which our policies as instances mapping was built.

3.4. KaOS Policy and Domain Services [11] use ontology concepts encoded in OWL to build policies. These policies constrain allowable actions performed by actors which might be clients or agents. The KAoS Policy Service distinguishes between authorizations (i.e., constraints that permit or forbid some action) and obligations (i.e., constraints that require some action to be performed when a state- or event-based trigger occurs, or else serve to waive such a requirement). The applicability of the policy is defined by a class of situations which definition can contain components specifying required history, state and currently undertaken action. In the case of the obligation policy the obligated action can be annotated with different constraints

restricting possibilities of its fulfillment. KAoS services have been extended to work equally well with both agent-based (e.g., CoABS Grid, Cougaar, SFX, Brahms) and traditional clients on a variety of general distributed computing platforms.

3.5. WSPL

WSPL[14] is being developed at Sun Microsystems. The Web Services Policy Language (WSPL) is suitable for specifying a wide range of policies, including authorization, quality-of-service, quality-of protection, reliable messaging, privacy, and application-specific service options. WSPL is of particular interest in several respects. It supports merging two policies, resulting in a single policy that satisfies the requirements of both, assuming such a policy exists. Policies can be based on comparisons other than equality, allowing policies to depend on fine-grained attributes such as time of day, cost, or network subnet address. By using standard data types and functions for expressing policy parameters, a standard policy engine can support any policy. The syntax is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML [5], discussed below) Standard. WSPL has been implemented, and is under consideration as a standard policy language for use with web services.

3.6. XACML provides a policy language which allows administrators to define the access control requirements for their application resources. The language and schema support include data types, functions, and combining logic which allow complex (or simple) rules to be defined. XACML also includes an access decision language used to represent the runtime request for a resource. When a policy is located which protects a resource, functions compare attributes in the request against attributes contained in the policy rules ultimately yielding a permit or deny decision. It is a powerful language, able to also express first order and higher order functions.

3.7. Rei [7] is a policy specification language based on a combination of OWL-Lite, logic-like variables and rules. It allows users to develop declarative policies over domain specific ontologies in RDF, DAML+OIL and OWL. Rei allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. A distinguishing feature of Rei is that it includes specifications for speech acts for remote policy management and policy analysis specifications like what-if analysis and use-case management. As Rei is geared towards distributed environments, it also includes conflict resolution specifications like modality preferences or priority assignments between policies or between individual rules of a policy.

Having produced a mapping for WS-Policy to OWL, we have shown that also Features and Properties and P3P can also be mapped, since they are less expressive than WS-Policy. We plan to focus on the more expressive languages (WSPL, XACML, Rei) in the future, to determine how much of them can be mapped into OWL, or whether we must move to a more expressive language (like SWRL), or out of first order logic altogether. We believe that translation considerations for existing and used policy languages should be a factor in future extensions to OWL.

4. Conclusion

We have presented a translation of the base formalism of WS-Policy into OWL-DL and described how those translations can be used for policy analysis, processing, and development. If our translation is correct, we have provided a formal semantics for WS-Policy. At worst, we have exposed some of the assumptions and ambiguities about the current specification.

We have demonstrated that an OWL-DL reasoner provides useful services for policy analysis, including policy containment, incompatibility, conformance, and incoherence. We expect that having such services available will raise the bar for policy engines overall.

In our future work, we intend to provide a standard mapping of all the current WS-Policy assertion languages with some structural fidelity. We also plan to attempt translations of at least parts of the other policy languages we described in order to get a more precise sense of their expressivity. If they cannot be mapped into OWL, we intend to isolate the incompatible expressivity in order to determine whether there are reasonable extensions to OWL that could accommodate it.

Finally, we intend to further develop our tools *as* WS-Policy processing tools. We shall investigate the gap between general purpose tools like Pellet and Swoop and things tuned for WS-Policy. For example, our explanation facility might do better for WS-Policies if it knew the characteristic structure of their translations.

5. ACKNOWLEDGEMENTS

This work was completed with funding from Fujitsu Laboratories of America- College Park, Lockheed Martin Advanced Technology Laboratory, NTT Corp., Kevric Corp., SAIC, National Science Foundation, National Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, NIST, and other DoD sources.

6. REFERENCES

[1] Anderson, A. H. An Introduction to the Web Services Policy Language. Sun MicroSystems. http://research.sun.com/projects/xacml/Policy2004.pdf

[2] Daniels, G. Comparing Features / Properties and WS-Policy. *W3C Workshop on Constraints and Capabilities for Web Servies*. Redwood Shoes, CA, USA, Oct 12 -13, 2004.

[3] Daniels, G. Features and Properties Musings. www-ws-desc@w3.org mailing list, October 2003. http://lists.w3.org/Archives/Public/www-ws-desc/2003Oct/0144.html

[4] Dean, M. and Schreiber G. OWL Web Ontology Language. Reference W3C Recommendation, http://www.w3.org/tr/owl-ref/. Feb 2004.

[5] Godik, S., and Moses,T., eds. OASIS eXtensible Access Control Markup Language (XACML) Version 1.1. Oasis Committee Specification, http://www.oasis-open.org/committees/download.php/4103/cs-xacml-specification-1.1.doc. 24 July 2003.

[6] Lassila, O. and Swick, R. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendations, WWW Consortium. Cambrigde,MA, USA. Feb 1999.

[7] Kagal, L. et al. A policy Language for a Pervasive Computing Environment. In Collection, *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. June 2003.

[8] Pellet – OWL-DL reasoner, http://www.mindswap.org/2003/pellet.

[9] Platform for Privacy Preferences Project. http://www.w3.org/P3P/

[10] Prud'hommeaux, E. RDF for Web Service Assertions. *W3C Workshop on Constraints and Capabilities for Web Services*. Redwood Shores, CA, USA. Oct 12-13, 2004.

[11] Uszokand, A. and Bradshaw, J. KAoS Policies for Web Services. *W3C Workshop on Constraints and Capabilities for Web Servies*. Redwood Shoes, CA, USA, Oct 12 -13, 2004.

[12] W3C Workshop on Constraints and Capabilities for Web Services. Redwood Shores, CA, USA. Oct 12-13, 2004. http://www.w3.org/2004/06/ws-cc-cfp.html.

[13] Web Services Policy Framework (WS-Policy). http://www-106.ibm.com/developerworks/library/specification/ws-polfram/.

[14] Web Services Policy Assertions Language. http://www-106.ibm.com/developerworks/library/ws-polas/

[15] Yu, T., Li N., and Anton A.L. A formal semantics for P3P. *ACM Workshop on Secure Web Services, October 29 2004, Fairfax VA, USA*.

[16] *Parsia, B., Sirin. E., Kalyanpur, A.* Debugging OWL Ontologies*, In The 14th International World Wide Web Conference (WWW2005), Chiba, Japan, May 2005.*