Authorization and Privacy for Semantic Web Services

Lalana Kagal[1], Massimo Paolucci[2], Naveen Srinivasan[2], Grit Denker[3], Tim Finin[1], Katia Sycara[2]

[1]University of Maryland, Baltimore County, Computer Science, 1000 Hilltop Circle, Baltimore, MD 21250
[2]Carnegie Mellon University, School of Computer Science, Newell-Simon Hall 1602D, Pittsburgh, PA 15213
[3]SRI International, Computer Science Laboratory, 333 Ravenswood Ave, Menlo Park, CA 94025
lkagal1@cs.umbc.edu, paolucci@cs.cmu.edu, naveen@cs.cmu.edu, grit.denker@sri.com, finin@umbc.edu , katia@cs.cmu.edu

Abstract

In this paper we address security of semantic Web services that are declaratively described in OWL-S. We propose ontologies to annotate OWL-S input and output parameters with respect to their security characteristics, including encryption and digital signatures. Moreover, we propose to incorporate privacy and authentication policies into OWL-S descriptions and requester profiles. We designed and implemented algorithms to check policy compliance and integrated them in the service selection process of the OWL-S MatchMaker. We extended the OWL-S VM with features for encrypting and signing messages exchanged between service requester and provider.

1. Introduction and Related Work

The introduction of web services has fundamentally changed the way business is conducted in today's market. Many financial transactions are nowadays web-service based application. Nevertheless, the current state-of-the-art of web services deployment and applications do not fully exploit the potential of web services and even less so of semantic web services. Standardization groups such as OASIS and W3C have primarily worked on syntactical issues of web service interoperability and web service security. The exploration of how semantically rich annotations will facilitate the discovery, selection, composition, invocation, and runtime monitoring of web services is only beginning. Semantic Web services promise to provide solutions towards this grand vision.

Our work focuses on the security aspects of semantic web services. We aim to provide semantically rich security and policy annotations for OWL-S service descriptions and enforce them by extending the existing OWL-S Matchmaker for policy matching and OWL-S Virtual Machine with security mechanisms. OWL-S is a set of ontologies for describing capabilities, interfaces and other details of web services and has been designed to facilitate automated web service discovery, composition, and invocation. An OWL-S description comprises profile (*what does the service do)*, process model *(how does the service achieve its functionality)* and grounding *(how can the service be invoked)* of the web service.

In earlier work [DKF+03], we proposed ontologies for modeling high-level security requirements and capabilities of web services and clients. This allows matching a client's request with the appropriate provided services. Thus, services are discovered based on not only their functional descriptions, but also on the basis of security criteria. For example, a web service can state that it is capable of performing Open-PGP encryption and requires the invoker to be capable of authenticating itself and communicating in XML. We added additional functionality to the DAML-S Matchmaker enabling it to verify that the security requirements of the invoker were fulfilled by the capabilities of the service and the requirements of the service were satisfied by the capabilities of the invoker. Our results are useful for coarse-grain matching decisions such as "Does the service provide encryption?" or "What kind of credential do I have to provide in order to authenticate myself to the service?"

In this paper we propose a more fine-grain security markup of service parameters in profile and process models. We further extend the framework through the addition of annotations about the security and privacy policies of services expressed in Rei, a rich, declarative policy language. Rei [KFJ03] is a logic-based language that allows rules and constraints to be defined over domain specific ontologies. This policy information is used in service selection and invocation.

Relevant related work stems from the areas of security for web services and trust and privacy policies for the semantic web. Lately there has been a significant body of standardization efforts for XML-based security, such as WS Security, SAML of the OASIS Security Services Technical Committee, and the Security Specifications of the Liberty Alliance Project. This work does not take the semantic aspects of web services into account. Work in the area of trust and privacy policies for the semantic web such as [GS04] and [BUJ+03] is also relevant for our work though this work in not specifically targeted towards semantic web services. There has also been a significant amount of research in security policies for distributed systems [UBJ+03, S94, LS97, LS99, JSS97, DDL+95]. KAoS provides a policy representation language based in DAML+OIL [UBJ+03]. Though this is an interesting approach, DAML+OIL itself is not able to adequately capture

the full range of policy constraints. Several efforts are being made to add syntax for rules in DAML+OIL and OWL [HPB+03]. Ponder is a policy specification language developed at Imperial College [DDL+95]. Though flexible and expressive, it does not lend itself to being useful for semantic web services as it more of a syntactic language. Rei, the language being used in this paper, draws on distributed policy work by Sloman and Lupu [S94, LS97, LS99]. It has an RDFS representation and includes a prolog-like notation for expressing rules on policy objects that goes beyond what can be done in DAML+OIL and OWL.

2. Introducing Policies in Web Service Descriptions

We claim that policies should be part of the representation of Web services and, in particular, of semantic Web services. Policies provide the specification of who can use a service under which conditions, how information should be provided to the service, and how provided information will be used later.

We consider the client-server model is one in which a client wants to invoke a web service. We claim furthermore that the use of policies is symmetric in the sense that both the provider and the requester are constrained by sets of policies that must be honored in their interaction. This model can be easily extended to a service-service architectural model.

We will address two kinds of policies in this paper: *privacy policies* and *authorization policies*. Privacy policies specify under what conditions information can be exchanged and what are the legitimate uses of that information. It may specify that the provider can give to the requester a key to access private information only if the key is encrypted during transmission. The requester, upon discovering the policy, should decide whether it can satisfy this condition or not. Similarly, the requester may have a privacy policy of its own requiring certain information to be kept confidential, and, thus, it may not share unencrypted private information. This policy would prevent the requester to interact with any Web service that does not perform the needed encryption.

Privacy policies are not only useful for specifying the confidentiality of data during transmission, but also after receipt of data. Consider for example a service that states that it will not distribute any details it receives as input. This might be an important requirement for a requester who requires privacy. This policy is interpreted as an obligation on the Web service and then acts as a contract. For example, if after invocation, the service does provide the requester's details to a telemarketer, the requester can take legal action against the service based on the policy. As financial transactions become more common among web services and as web services start dealing with confidential information (e.g., requester's name, address, social security numbers, credit cards, telephone numbers, etc.) more requesters will expect their privacy policies to be enforced.

We also consider authorization policies, i.e. policies that constrain the provider to only accept requests for service from certain clients. For example, an authorization policy could state that a requester must act on behalf of a person who is member of a certain organizational group, and that this membership must be proven with a digital certificate. Similarly, the requester may limit invocation to only selected providers.

3. A Motivating Example

Subsequently, we use an example that illustrates various aspects of OWL-S security policies. Consider a scenario in which a scientist is looking for an online computing service for its experimental data. The scientist requires that any personal information she is required to provide to the service (such as name or SSN) will be kept confidential. This means that she is only looking for web services that accept encrypted data and that will not release her personal information to other services or agents. This constitutes the privacy policy of the researcher.

Assume there is web service offered by an organization that can perform the necessary data computations. This web service is only accessible for members of a certain group of selected organizations and the scientist needs to register with the service in a way that can be authenticated. This constitutes the authorization policy of the web service.

We approach formalization of these privacy and authentication policies on two abstraction levels. On a more concrete level (see Section 4), we suggest ontologies to annotate Web service input and output parameters with security characteristics. Security characteristics of parameter state whether these parameters are encrypted or digitally signed. On a more abstract level (see Section 5), we provided formalizations of the privacy and authorization policies in Rei. Selecting Web services that satisfy requester policies will be part of an extended OWL-S matchmaking process.

Enforcement of cryptographic mechanisms such as encrypting or signing messages will be achieved via integration into the OWL-S VM, a generic processor for the OWL-S process model and a tool for automatic invocation of OWL services. This is described in Section 6.

4. Privacy and Authentication: OWL-S Parameter Markup

In this paper we propose ontologies and markup to capture security information of web service input and output parameters. For example, we would like to express that the scientist requires her personal information to be encrypted. The problem of representing data confidentiality in the markup of semantic web services such as OWL-S is that encrypted data by its very nature does not reveal its internal value or structure because it is just a byte string. We therefore suggest a semantic markup that specifies the security characteristics of input and output parameters of web services while keeping information about the structure of the

data without revealing its value. This meta-information about the kind of data exchanged with a service can be used for service selection.

A basic ontology to handle cryptographic details of input and output parameters of web services can be found at http://www.csl.sri.com/users/denker/owl-sec/infObj.owl. In order to capture encrypted or signed input or output data of services we define an `InfObj` class (*information object)* and subclasses `EncInfObj` (*encrypted information object)* and `SigInfObj` (*signed information object)*. The `InfObj` class will be used as a range for input and output parameters of OWL-S services. Information objects have a `baseObject` that describes the type or structure of the information that is encoded in it. For example, the base object of an I/O parameter of class `EncInfObj` can be a class such as SSN. This property allows deriving knowledge about the kind of data exchanged and can be used for reasoning purposes such as whether a service parameter fits the requirements of a client or output/input parameter of two web services match so that the services can be sequentially combined. Furthermore, an information object can have a property cryptoAlgUsed to refer to the specific cryptographic algorithm that was used for signing or encrypting the data.

This basic ontology is enough to describe the cryptographic details necessary in the motivating example. The first step is the discovery and selection of a service that satisfies the requirements of the scientist. The second is the invocation of the service. We describe below how data confidentiality should be handled in these two steps.

A matchmaker is used to find a data computation service that satisfies the functional requirements of the employee (such as type of data to be processed and turn-around time of computation). In this paper we omit the details about those requirements and focus on the security related requirements of client and service. Service input and output parameters will be described in form of information objects that reference the type of information (such as name, SSN, etc.) and the kind of security technique applied to it (such as encryption or signature including the specifics about algorithm). The same approach will be applied to the capabilities and requirements of clients.

For our example, we assume that the scientist is capable of providing an encrypted instance of class Person. We use the FOAF ontology (see http://xmlns.com/foaf/0.1) to specify domain-specific information such as person, name, organization, group, etc.

Assume the following (partial) instance definition of class `Person` to describe our scientist.

```
<foaf:Person rdf:ID="MarySmith">
 <foaf:name xml:lang="en">Mary Smith</foaf:name>
 <foaf:title>Dr.</foaf:title>
</foaf:Person>
```

The scientist is not willing to reveal her personal information to everybody. One realization of this privacy policy is to look for services that accept this information in an encrypted manner. OWL-S service descriptions contain input descriptions in the `process:hasInput` property, where `process` is the namespace abbreviation for the latest version of the OWL-S specification that can be found at http://www.daml.org/services/owl-s/1.0. Thus, the kind of service the scientist is looking for should have the following input parameter in its profile

```
<process:hasInput rdf:ID="PersonInf">
<process:parameterType rdf:resource="EncPersonInfObj"/>
</process:hasInput>
where
<Class rdf:ID="EncPersonInfObj">
 <SubClassOf rdf:resource="#EncInfObject"/>
 <Restriction>
  <onProperty rdf:resource="baseObject"/>
  <allValuesFrom rdf:resource="&foaf;#Person"/>
 </Restriction>
</Class>
```

A matchmaking service uses this information to select services with appropriate profile or process descriptions.

So far we have dealt with privacy aspects that can be handled with cryptographic techniques. Similarly, some aspects of authorization policies can be addressed through these techniques.

Assume that our data computation service decides authorization of the scientist on the basis of two aspects: The scientist needs to be member of a certain group of selected organizations, and the scientist needs to register with the service in a way that can be authenticated. The first policy will be treated in the next section as a Rei policy. The latter condition is expressed by the service with a requirement for the scientist to register with its name and other personal information in a verifiable way in order to avoid impersonation attacks. Digital signature is a cryptographic technique to achieve verifiable authentication. The web service could express its requirement about authenticated sign-in in the following way:

```
<process:hasInput rdf:ID="RegInf">
<process:parameterType rdf:resource="SigRegInfObj"/>
</process:hasInput>
<Class rdf:ID="SigRegInfObj">
 <SubClassOf rdf:resource="#SigInfObject"/>
 <Restriction>
  <onProperty rdf:resource="base"/>
  <allValuesFrom rdf:resource="&foaf;#Person"/>
 </Restriction>
</Class>
```

As one can see in our examples, the requirements of the client (here the scientist), might not be in congruence with the requirements of the web service (here the data computation service). The client requires encrypted person information as input for the service, and the service requires the client to sign its information. Nevertheless, a matchmaker can deduce from our markup that client and web service agree on the fact that both want person information to be submitted.

5. Representation of and Reasoning about Policies

In this paper we propose to integrate expressive policies relating to several aspects of security, including authorization and privacy into semantic web services. Policies are useful primarily during the discovery phase and for forming informal *contracts*.

Representing Policies in Rei

Policies are represented using Rei [KFJ03], an RDFS-based language for policy specification. Rei is modeled on deontic concepts of rights, prohibitions, obligations and dispensations. These constructs have four attributes, actor, action, provision, and constraint. Constraint specifies conditions over the actor, action and any other context entity that must be true at the time of the invocation, whereas provision describes conditions that should be true after the invocation. Provisions are obligations on the part of the actor.

The class `Policy` is at the root of the Rei ontology (http://www.csee.umbc.edu/~lkagal1/rei/swspolicy.owl). In our implementation this class is the range of the property `policyEnforced`, a new property of OWL-S description described in the following Section. We define three subclasses of `Policy` to specify the different types of policies that we can support, namely `PrivacyPolicy`, `AuthorizationPolicy`, and `Confidentiality-Policy`.

Rei also models several speech acts that modify the rights and obligations of the sender and receiver. The speech acts include delegation, revocation, request, cancel, promise and command. Rei has a unique nesting feature that allows speech acts to be part of the constraints and provisions of other deontic concepts. For example, a promise can be the provision of a right or a delegation can be part of the constraint of a right. These constructs allow different kinds of policies to be described including authorization, privacy and confidentiality.

For example, we can define in Rei an authorization policy that can be described in natural language as follows: "Permit everyone to access the data computation service who is in the same group as the provider of the service." To specify this policy, we exploit the OWL-S property `contactInformation`, which can be specialized to have the range `foaf:Agent`. This property can be used to describe the provider of the service. We assume that the OWL-S description of the data computation service exists at some name-space http://www.somenamespace.com/dcs. Moreover, we assume that there exists information about the groups to which the scientist belongs as well as information about the groups to which the service provider belongs. A section of the authorization policy is specified as follows in Rei (RDF/N3):

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rei: <http://www.csee.umbc.edu/~lkagal1/rei#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/l#>.
@prefix process: <http://www.daml.org/services/owl-s/1.0#>.
@prefix dcs: <http://www.somenamespace.com/dcs#>.
@prefix : <#>.

:actorVar a rei:Variable.
:providerVar a rei:Variable.
:projectVar  a rei:Variable.
:AuthPolicy1 a rei:Right;
   rei:agent actorVar;
   rei:constraint [a rei:AndCondition;
         rei:first[a rei:AndCondition;
            rei:first[a rei:SimpleCondition;
                  rei:subject dcs:profile;
                  rei:predicate process:contactInformation;
                  rei:object providerVar];
            rei:second[a rei:SimpleCondition;
                  rei:subject providerVar;
                  rei:predicate foaf:currentProject;
                  rei:object projectVar]];
         rei:second[a rei:SimpleCondition;
            rei:subject actorVar;
            rei:predicate foaf:currentProject;
            rei:object projectVar]].
```

On the other hand, a requester may have the privacy policy of never sharing personal information. This policy can be expressed similarly in Rei in the following way.

```
:serviceVar a rei:Variable.
:outputVar a rei:Variable.
:PrivPolicy1 a rei:Prohibition;
      rei:action serviceVar;
   rei:constraint [a rei:AndCondition;
         rei:first[a rei:SimpleCondition;
               rei:subject serviceVar;
               rei:predicate process:hasOutput
               rei:object outputVar];
         rei:second [a rei:SimpleCondition;
               rei:subject outputVar;
               rei:predicate rdfs:domain;
               rei:object foaf:Person]].
```

Specifically, this privacy policy assumes that all personal information of the scientist is specified by the FOAF ontology concepts. The policy states that any service that has as output any concept that describes a FOAF person should be prohibited. The privacy policy acts as a template for allowed or prohibited services based on output parameters. Additionally, the requester may want to specify that any personal information, if shared, must be encrypted.

```
:serviceVar a rei:Variable.
:someVar a rei:Variable.
: PrivPolicy2 a rei:Right;
rei:action serviceVar;
      rei:constraint [a rei:AndCondition;
            rei:first[a rei:AndCondition;
                  rei:subject serviceVar;
                  rei:predicate process:hasInput;
                  rei:object someVar];
```

```
rei:second[a rei:SimpleCondition;
        rei:subject someVar;
        rei:predicate process:parameterType;
        rei:object inf: EncPersonInfObj]].
```

Finally, Rei provides a meta-policy prioritization mechanism to resolve policy conflicts. For example, we defined above two policies that may be contradictory with each other, the first one specifies that the requester does not want to share personal information, the second one that personal information can be shared only if it is encrypted. The requester can then state that *PrivPolicy2* holds priority over *PrivPolicy1*, ensuring that services that meet *PrivPolicy2* are checked first.

Extending OWL-S with Policies

Web services are described in OWL-S with the help of three modules: a profile that provides a general description of the Web service, a process model that describes how the Web service performs its tasks and the Web service interaction protocol, and finally the grounding that specifies how the atomic processes in the Process Model map onto WSDL [CCM+01] representations.

Security information is needed in all three modules: the profile provides the place to specify the requirements for the Web service invocation, while the process model and the grounding need a specification of the security requirements of the messages exchanged between the Web service and its requester.

There is no explicit place for security policies in OWL-S, but the natural extension toward security is to link policies to the profile. The rational for this is that policies specify general properties of the Web service rather than properties that are specific of any process.

Based on our earlier work [DKF+03], we propose that policies are an extension of the security requirements of services and suggest the addition of a property, called `policyEnforced`, defined as a subproperty of `securityRequirement` (http://www.csl.sri.com/~denker/owl-sec/serviceSecurity.owl). `PolicyEnforced` describes the different policies that have to be enforced for the correct execution of service.

A web service requiring above described authorization policy could be annotated as follows

```
<profile:Profile rdf:ID="DataComputationService_Profile">
  <profile:textDescription>
    This data computation service requires authorization.
  </profile:textDescription>
  …
  <policyEnforced: rdf:resource="#AuthPolicy1"/>
</profile:Profile>
```

Similarly, for requesters we envision annotations of their policies. In earlier work [DKF+03], we suggested a property `securityRequirement` with domain `Agent`, a general class for clients and requester. Above mentioned property `policyEnforced`, is also subproperty of agents' `securityRequirement` and we define `foaf:Person` class to be a subclass of `Agent`. Thus, the scientist requiring that her personal information is transmitted as encrypted data and never appears as output of a service is defined as follows

```
<foaf:Person rdf:ID="MarySmith">
    <foaf:name xml:lang="en">Mary Smith</foaf:name>
    <foaf:title>Dr.</foaf:title>
    <policyEnforced: rdf:resource="#PrivPolicy1"/>
    <policyEnforced: rdf:resource="#PrivPolicy2"/>
</foaf:Person>
```

Using policies to select providers

Properties play an essential role in the discovery and selection of providers. Policies specify constraints on how to interact with a provider and what to do with the information exchanged. Any violation of these policies during the interaction would of course result in a failure of the interaction. Therefore, the requester should select a provider with compatible policies. For example, if the requester has a policy that all information transmitted must be encrypted, it will not be able to interact with a provider that has a policy of sending all information in the clear.

During the discovery process, the requester has the responsibility of selecting the best provider, and, as part of this process, the requester needs to verify the compatibility of its own policies with the policies of the provider. Rei provides a reasoning engine over policies and domain knowledge to provide an evaluation of compatibility of rights, prohibitions, obligations and dispensation of entities within the domain.

The primary contribution of this paper is the integration of Rei reasoning on policies within a capability-based matching engine, called the MatchMaker [PKP+02]. The resulting policy compliance algorithms for privacy and authorization policies are summarized in the following.

For privacy constraints the matching engine first selects the providers with the capabilities expected by the requester (step 1) and then the matchmaker extracts the privacy policies of both the requester and the provider (step 2) and uses the Rei policy reasoner to verify the compatibility of the policies (steps 3 and 4). If the policies are found to be incompatible then the provider is abandoned, otherwise it is selected.

1. The MatchMaker fetches the OWL-S description of a web service that matches the functional requirements of the requester.
2. It retrieves the previously defined privacy policy from the client/requester and extracts the privacy policies from the provider's profile.
3. The MatchMaker sends the OWL-S description along with the privacy policies to the Rei reasoner.
4. As the privacy policy defines service templates that are prohibited, the Rei reasoner verifies that the matched service is not prohibited. It checks that the service does not have as output any information that the client wishes to keep private. It also checks that

the privacy policies of provider and requester are not in contradiction.
5. If a privacy policy is not satisfied, the Rei reasoner returns false and the Matchmaker continues to check the next service for privacy compatibility. Otherwise the Rei reasoner returns true and the Matchmaker returns this service to the client.

Similarly the algorithm for authorization policies is as follows:

1. The MatchMaker extracts the precondition of the service that is of type `AuthorizationPolicy`.
2. It gathers all relevant information about the user, and sends this along with the authorization policy to the Rei reasoner
3. If the Rei reasoner returns true, then the authorization policy is satisfied and the service can be returned to the client. Otherwise the Matchmaker continues checking the next service for authorization compatibility.

Verifying adherence of policies during interaction

Policies can be declared in the profile, but they should be enforced in the process model which is responsible for the interaction between the provider and the requester. In the OWL-S specification, the process model expresses the interaction protocol of the provider that the requester should follow for successful interaction. Furthermore, the grounding module provides a mapping from the process model to the messaging specification, and specifically to WSDL and SOAP.

The emerging specifications for Web services security [ADH+02] assume that message security be specified at the WSDL and SOAP level. If the requester wants to verify whether the policies will be enforced in the interaction, it needs to verify the constraints placed by the provider on message passing.

If the requester wants to verify that the provider adheres to the published policies, it needs to analyze all the different specifications for the message passing. This is also required because the provider may not expose its policies completely, but it may compile some aspects directly in the interaction specifications.

The algorithm described below is a first attempt to enable the requester to verify the provider's adherence to policies. The requester uses the interaction protocol specifications to derive how different types of information are encoded in the messages that the provider sends, or the messages that it receives (steps 1 and 2). This process is achieved by exploiting the grounding which is a mapping from the process model, which describes the ontological type of the information to be exchanged, and WSDL that describes how this information is encoded in the message. Finally (step 3), a reasoning system is used to verify whether the encoding proposed adhere to the policies of the requester and the provider. For the future we envision to implement this algorithm also in the Rei reasoner. If the results of the reasoning about policies are not consistent with the requester's policies, then the requester knows that it will incur in a violation if it pursues the interaction with the provider. If instead the reasoning reveals an inconsistency between the policies specified and the actual interaction management, the requester may decide whether to select the provider depending on whether it can satisfy the additional requirements, and its own judgment on the provider's failure.

1. The requester gathers the process model, grounding, WSDL and SOAP specifications from the provider, and its own policies, as well as the policies of the provider.
2. The requester uses the process model, grounding, WSDL and SOAP specifications of the provider to detect what type of encryption is adopted for the different types of information
3. The reasoner is used to verify that
   a. The requester's policies are satisfied
   b. The provider enforces its own policies
4. If the first test fails, the requester does not use the provider. If the second fails, the requester makes its own decisions about the use of the provider.

In general, it behooves the provider to be explicit and honest about its policies. Indeed, if the provider is not honest, and it specifies a policy that it does not enforce, it will lose all the requesters that may not want to adhere with the policy, and lose the trust of the requesters that realize that policies are not enforced. Similarly, if the provider does not explicitly specify some of its policies, it may get to interact with requesters that cannot deal with those policies, and therefore failing the interaction.

6. Enforcing Privacy and Authentication in OWL-S VM

We mentioned in previously, one way to fulfill privacy or authentication is through encrypting or signing input and output parameter. We propose to keep the work involved with cryptographic operation transparent from the requester by extending the tool that invokes the web service (in our case OWL-S VM) with features for encrypting or signing data exchanged between client and server.

The OWL-S Virtual Machine (OWL-S VM) implements the process model and grounding of OWL-S to manage the interaction between web services. It is a processor that automates web service interaction without human intervention. The architecture of the OWL-S VM, shown in Figure 1, is represented by three components in the center column: the *Web service Invocation*, the *OWL-S VM* and the *OWL Parser*. The Web service invocation module is responsible for contacting other Web services and receiving messages from other Web services. The transaction with other Web services may be based on SOAP messag-

ing, or on straight HTTP or any other mode of communication as described by the WSDL specification of either of the Web service provider. Upon receiving a message the Web service invocation extracts the payload, or in other words the content of the message and either sends it to the OWL Parser or it is passed directly to the OWL-S VM.

The OWL parser is responsible for reading fragments of OWL ontologies and transforming them into predicates that can be used by the OWL inference engine. The OWL parser is also responsible for downloading OWL ontologies available on the Web, as well as OWL-S descriptions of other Web services to interact with.

The OWL-S VM is the center of our implementation: it

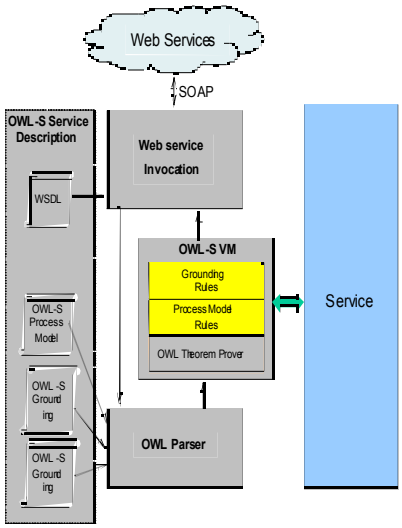

Figure 1: The DAML-S Web Service architecture

uses ontologies gathered from the Web and the OWL-S specifications of the Web services to make sense of the messages it received, and to decide what kind of information to send next. To make these decisions the OWL-S VM uses a set of rules that implement the semantics of the OWL-S Process Model and Grounding. The OWL-S VM is also responsible for the generation of the response messages; to accomplish the latter task, the OWL-S VM uses the Grounding to transform the abstract information exchanges described by the Process Model into concrete message contents that are passed to the Web service invocation module to be transformed into actual messages and sent off to their receivers.

The other two columns of the diagram in Figure 1are also very important. The column on the left shows the information that is downloaded from the Web and how it is used by OWL-S Web services. Specifically the WSDL is used for Web service invocation, while ontologies and OWL-S specifications of other Web services are first parsed and then used by the OWL-S VM to make decisions on how to proceed.

We have extended the OWL-S VM to enforce authorization and privacy policies. We have implemented the required security transformation on the I/O parameters in the OWL-S VM. This way we achieve that upon execution of an atomic process, the OWL-S VM uses the semantic parameter annotation in the corresponding process model to enforce the privacy and authorization constraints that can be implemented through cryptographic techniques (using encryption and digital signatures). SOAP security annotations are used to implement the actual encryption or signing of the messages.

With the implementation of the security mechanisms proposed in this paper, web services implementing the OWL-S VM are guaranteed to maintain a secure communication with their partners.

7. Summary and Future Work

In this paper we describe our initial foray into modeling the various security aspects of web services. We propose adding privacy and authentication annotations to input and output parameters to aid in selection of semantic web services. This annotation includes not only the range of the parameter but also the cryptographic type, if any. We discuss the role of authorization and privacy policies for semantic web services and describe a candidate policy specification language. Specialized policies for authorization and privacy are provided. An algorithm for checking whether a web service description is in compliance with the policies has been designed and implemented. An integration of our annotation and policy techniques in the OWL-S MatchMaker supports policy-compliant service selection. An extension of the OWL-S VM with cryptographic features for message encryption and signing allows the enforcement of security requirements at runtime.

Another dimension of complexity is added when the policies do not match and some form of negotiation is necessary. Let's assume one web service requires authentication of the other web service, but the credential provided is not sufficient. A negotiation phase, following certain communication protocols could be entered, to resolve this problem. Future work will address negotiation protocols. Furthermore, the abstraction level or expressiveness of the policy language also determines the complexity of the problem. Consider a policy stating that a client never wants to reveal information from which somebody can deduce his home address. Depending on the information exchanged with the service and additional context information, this could mean that the client's phone number is never released, since a reverse lookup could compromise the address. This exemplifies the broad range of policies that are relevant. More complex policies that address combinations of these security notions and other user-defined policies will be subject of future work.

References

[ADH+02] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Naga-

ratnam, H. Prafullchandra, J. Shewchuk, D. Simon.. *Specification: Web Services Security (WS-Security),* http://www-106.ibm.com/developerworks/webservices/library/ws-secure, 2002

[BL01] T. Berners-Lee. *Notation3 - An RDF language for the Semantic Web*, http://www.w3.org/DesignIssues/ Notation3.html, 2001

[BUJ+03] J.M. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisiti, B. Benyo, M.R. Breedy, M. Carvalho, D. Diller, M Johnson, S. Kulkarni, J. Lott, M. Sierhuis, R. Van Hoof. *Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads,* AAMAS, 2003.

[CCM+01] E. Christensen, F. Curbera, G. Meredith, S. Weerawaraana. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl, 2001

[CSG+03] L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, P.R. Smart. *Towards a Knowledge-Based Approach to Semantic Service Composition*, Second Int. Semantic Web Conference, Sanibel Island FL, October 2003.

[DDL+95] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. *The ponder policy specification language*. In The Policy Workshop 2001, Bristol U.K., LNCS 1995, Jan 2001.

[DKF+03] G. Denker, L. Kagal, T. Finin, M. Paolucci, K. Sycara. *Security for DAML Web Services: Annotation and Matchmaking*, Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003.

[GS04] F. Gandon and N. Sadeh. Semantic Web Technologies to Reconcile Privacy and Context Awareness. *Web Semantics Journal,* Vol 1, No. 3, 2004.

[HPB+03] I. Horrocks , P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof , M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML, Internet draft,* http://www.daml.org/2003/11/swrl/

[JSS97] S. Jajodia, P. Samarati, V. S. Subrahmanian. *A Logical Language for Expressing Authorizations*. IEEE Sym. on Security and Privacy. Oakland, *CA*, 1997.

[KFJ03] L. Kagal, T. Finin, and A. Joshi, *A Policy Based Approach to Security on the Semantic Web*, Second Int. Semantic Web Conference, Sanibel Island FL, October 2003.

[LS97] E. Lupu and M. Sloman. *A Policy Based Role Object Model*. In Proceedings EDOC'97, IEEE Computer Society Press, 1997.

[LS99] E. Lupu and M. Sloman. *Conflicts in policy-based distributed systems management*. IEEE Trans. on Software Engineering, 25(6):852–869, Nov/Dec 1999.

[PAS+03] M. Paolucci, A. Ankolekar, N. Srinivasan, K. Sycara. *The DAML-S Virtual Machine*, Second Int. Semantic Web Conference, Sanibel Island FL, October 2003.

[PKP+02] M. Paolucci, T. Kawamura, T. Payne, K. Sycara. *Semantic Matching of Web Services Capabilities*, First Int. Semantic Web Conference, Sardinia, Italy, June 2002.

[S94] M. Sloman. *Policy driven management for distributed systems*. Journal of Network and Systems Management, **2**:333, 1994.

[UBJ+03] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. Lott. *KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement*. Proceedings of Policy Workshop, Como, Italy, 2003..

[WPS+03] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau. *Automating DAML-S Web Services Composition Using SHOP2,* Second Int. Semantic Web Conference, Sanibel Island FL, October 2003.