OWL-S Technology for Representing Constraints and Capabilities of Web Services

Katia Sycara, David Martin, Deborah L. McGuinness, Sheila McIlraith, Massimo Paolucci
On behalf of the OWL-S Coalition
In collaboration with Tim Finin, Grit Denker, Lalana Kagal

Description of constraints and capabilities of Web services is crucial to enable flexible service discovery, interaction and management. Constraints and capabilities are at the heart of representing policies that govern the behavior of Web services. Because policies regulate the behavior of different parties they are effective if and only if they are unambiguously understood by all the parties that they regulate. To achieve this level of common understanding policies should be expressed in terms of shared ontologies that are based on a logic shared by every party. For this reason, we believe that the starting point of any activity on expressing policies, capabilities and constraints should take into account the work that has been developed in the context of Semantic Web services, and specifically OWL-S.

OWL-S

A key component of the Semantic Web Services vision is the creation of a language for describing Web Services. OWL-S is such a language [6]. It is an OWL [4] ontology for describing Web Services, created by a coalition of researchers from BBN, Carnegie Melon University, Nokia, SRI International, Stanford University, University of Maryland, University of Toronto, and Yale University. The initiative is supported by the Defense Advanced Research Projects Agency, as part of the DARPA Agent Markup Language (DAML) program.

OWL-S complements industry efforts such as SOAP, WSDL, UDDI, and BPEL4WS. It builds upon these efforts by adding rich typing and class information that can be used to describe and constrain the range of Web service capabilities much more effectively than XML data types. Further, it integrates such rich class representations with a process model, designed not only to capture the control flow and data flow of Web services, but also their side effects (preconditions and effects) in the world. The use of such a language enables the grouping of like services and like data types into taxonomic hierarchies, together with rich definitions of the relationships and constraints between classes and their instances. The well-defined semantics enables automated manipulation of these structures, with known outcome. In short, OWL-S makes automated interoperation possible.

The drivers for the design of OWL-S are the envisioned automation tasks, including semi-automated or fully automated *discovery*, *invocation*, *interoperation*, *composition*, *execution monitoring* and *recovery*, *simulation* and *verification*. To this end, the OWL-S ontology is divided into three sub-ontologies: the service *profile*, *process model*, and *grounding*.

Describing Web Service Capabilities

The OWL-S *profile* describes *what the service can do* for the purposes of advertising, discovery, mediation and matchmaking. It enables creation of a rich ``yellow pages'', taxonomically encoding the kinds of information needed by a service requestor (whether human or software agent) to determine if the service meets its needs. Service profiles can be exposed at a URL for crawlers to find. They can also be published in service registries, such as UDDI [9]. The formal structure and richness of description provided through the use of the OWL language enables powerful forms of querying.

OWL-S supports the construction of a hierarchy of subclasses of the Profile class, with (potentially multiple) inheritance of properties. Since classes are described in terms of defining properties, a Web Service can be characterized as belonging to multiple classes. For example, the LocateBook service at www.amazon.com can be characterized both as a service that determines whether Amazon carries a particular book, and as a bibliographic reference tool, simply as a consequence of its inputs, outputs, preconditions and effects. Many of the components of a profile are domain-specific. For example, geographic constraints are relevant to a catering service or an airline, but not to an archive of journals. OWL-S profiles will make it possible for well developed existing taxonomies of service categories, such as those in UNSPSC, to evolve into more expressive class-hierarchical categorization schemes.

Describing Web Service Workflows

The OWL-S *process model* describes *how the service works*. It describes the control flow and data flow of the program that realizes the service as well as the programs preconditions and side effects in the world. The process model was designed for use by service-requesters in connection with service selection, invocation, interoperation, composition, and monitoring, or by service tools for service simulations and verification. The process model can also be used to help populate the service profile. The OWL-S process model is a superset of what is typically found in process modeling and workflow languages, combining a process modeling language with both an AI-inspired action language and a language for describing classes and their inter-relationships. Further, the process model has a well-defined semantics.

Central to an OWL-S process model is the specification of a service's inputs, outputs, preconditions, and effects. Process *inputs* and *outputs* are named and typed using either OWL classes or data types provided by XML Schema. *Preconditions*, of which there may be any number, must all hold in order for the process to be invoked, and *effects* indicate what is accomplished by the service, or more generally, changes in the world brought about by the service. Conditions can be associated with outputs and effects, since the outputs and effects of a service are often predicated on some internal state of the system.

The OWL-S process ontology is subdivided into three process types: *atomic*, *simple*, and *composite* processes. Atomic processes are the units of invocation; that is, an atomic process, somewhat similarly to a programming language procedure, can be called by transmitting an invocation message (which carries its inputs) to the process, and its results returned in a response message.

Simple processes are like atomic processes in that they are conceived of as having single-step executions. Unlike atomic processes, however, they are not directly invocable and are not associated with a grounding. Simple processes provide a means of abstraction; that is, they can provide abstract views of atomic or composite processes.

Composite processes are constructed from subprocesses, which in turn can be either atomic, simple, or composite. Control constructs such as *Sequence* and *If-Then-Else* are used to specify the structure of a composite process. In addition to describing control flow, this structural specification also includes argument binding constructs for indicating the data flow, which, in conjunction with the grounding, provides a means of constraining the patterns of messages involved in interacting with a complex service.

Describing Web Service Interaction

The OWL-S *grounding* tells *how the service is used*; that is, it specifies the details of how a computer program or agent can access a service. Typically, a grounding will specify some well known communications protocol, service-specific details such as port numbers used in contacting the service, and an unambiguous means of exchanging data elements of the types required and produced by the service.

The default grounding approach provided by OWL-S relies on specification mechanisms already provided by WSDL, while at the same time exploiting the richer descriptions made available through the use of the OWL language [4].

Extending OWL-S with policies

OWL-S can also provide descriptions of policies and security requirements [2]. In the profile, security requirements can be used for Web service discovery; in the process model and grounding, security requirements are needed for invocation and message exchanges between the Web service and its requester.

OWL-S supports the representation of policies as extensions of services' security requirements by adding a property called policyEnforced, defined as a subproperty of securityCapability (see www.csl.sri.com/~denker/owl-sec/serviceSecurity.owl). policyEnforced describes the different policies that must be enforced for the service to execute correctly. Correspondingly, the property securityRequirement with domain Agent, can be used to define security requirements of service requester agents. Property policyEnforced is also a subproperty of a requester's securityRequirement. OWL-S has been linked to Rei [3]. Rei is a RDF-Schema-based language for policy specification. It 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 invocation. Provisions, which represent the actor's obligations, describe conditions that should be true after invocation. These basic constructs allow Rei to represent different kinds of policies, including authorization, privacy and confidentiality. The class Policy is at the root of the Rei ontology. Policy has subclasses of PrivacyPolicy, AuthorizationPolicy and ConfidentialityPolicy. Rei's Policy class is linked with the OWL-S ontology by defining a new OWL-S description property, policyEnforced, of which

Policy is the range (see www.csee.umbc.edu/~lkagal/rei/examples/ses-sec/swspolicy.owl). Linking OWL-S with Rei allows specification of privacy, authentication and confidentiality to be specified for service providers and requesters.

Using policies to select providers

During the discovery process, the requester must select the best provider. To do this, the requester must verify the compatibility of its policies with the provider's. We have integrated Rei's policy reasoning engine that evaluates the compatibility of rights, prohibitions and obligations of the different agents in a domain with the OWL-S Matchmaker, a capability-based engine that matches OWL-S service profiles to service requests. As an example, we present an implemented and tested algorithm where Rei and OWL-S Matchmaker work together to select a service based on matching of requester's privacy constraints. In the example, we consider a requester who requires that certain type of information will be transmitted as encrypted data and that it will never appear as a service's output.

1. The Matchmaker fetches the OWL-S descriptions of the set of services that match the functional specifications of the requester, i.e. the services that can perform the requested task.
2. For each service that matches the functional requirements, the Matchmaker retrieves the privacy policy of the requester and from the service provider's profile.
3. The Matchmaker sends the OWL-S description and the privacy policies to the Rei reasoner.
4. As the privacy policy defines the prohibited service templates, the Rei reasoner verifies that the matched service is not prohibited. It checks that the service doesn't have as output any information that the client wants to keep private. It also checks that the provider and requester's privacy policies don't contradict each other.
5. If a privacy policy isn't 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.

Verifying policy adherence

The policies that are declared as part of the service profile should be enforced in the process model that is responsible for the provider and requester interactions. 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 assume that message security is specified at the WSDL and SOAP levels [8]. If the requester wants to check whether the policies will be enforced in the interaction, it must 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 must analyze all specifications for the message passing. The requester also needs to do this because the provider might not expose its policies completely, but it could compile some aspects directly in the interaction specifications.

The following algorithm enables the requester to verify the provider's adherence to policies:

1. The requester gathers the process model, grounding, WSDL and SOAP specifications from the provider, its own, and the provider's policies.
2. The requester uses the provider's process model, grounding, WSDL and SOAP specifications to extract what encryption is used for the different types of information.
3. The reasoner verifies that
    a. The requester's policies are satisfied
    b. The provider enforces its own policies
4. If the first test fails, the requester doesn't use the provider. If the second fails, the requester makes its own decisions about using the provider.

Steps 1 and 2 are achieved by exploiting the grounding and WSDL, which describes how this information is encoded in the message. Step 3 of this algorithm can be implemented in the Rei reasoner. If the results of the reasoning about policies aren't consistent with the requester's policies, the requester knows that it will incur a violation if it pursues the interaction. If the reasoning reveals an inconsistency between the policies specified and the actual interaction, the requester may decide whether or not to select the provider depending on whether it can satisfy the additional requirements and its own judgment on the provider's failure.

Enforcing privacy and authentication

Privacy or authentication can be fulfilled through encrypting or signing I/O parameters. The work involved with cryptographic operation can be kept transparent to the requester by extending the tool that invokes the Web Service, in our case the OWL-S Virtual Machine (hereafter OWL-S VM) with features for encrypting or signing data exchanged between client and server.

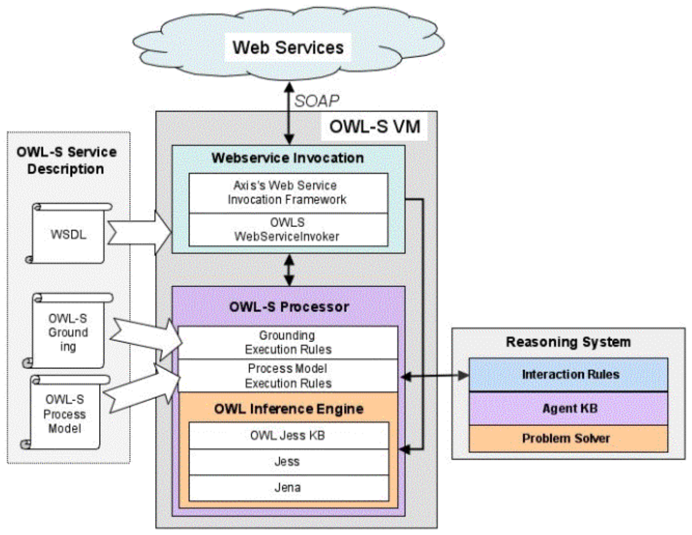

Figure 1: The OWL-S Virtual Machine

The OWL-S VM [7] implements a general purpose Web service client that relies on the OWL-S process model and grounding to automate the interaction between Web Services minimizing human intervention. The OWL-S VM architecture, shown in Figure 1 is organized in three columns: on the left side are the inputs to the OWL-S VM, specifically the process model, grounding and WSDL description of a Web service. The central column describes the OWL-S VM proper, while the box on the right describes the Reasoning System of the agent that uses the OWL-S VM. Upon receiving the process model and grounding of a Web service, the OWL-S VM activates the OWL-S Processor module, which implements the semantics of OWL-S and OWL, to control the interaction of the Web service through the execution of the process model. In addition, the WSDL description of the Web service is used to parameterize the Web service invocation module that manages the actual message passing between the OWL-S VM and the Web service. Whenever the process model requires a message passing between the client and the Web service, the OWL-S processor asks the Reasoning System for the content of the messages to send and then to the Web service invocation module to send the message.

The OWL-S VM has been implemented and tested. It is currently being extended to enforce authorization and privacy policies. Upon executing 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 cryptographic techniques (using encryption and digital signatures) can implement. We use SOAP security annotations to implement the actual message encryption or signing. Thus, Web services implementing the OWL-S VM are guaranteed to maintain secure communication with their partners.

Use Case

The representation of security capabilities and requirements in OWL-S is supported by the OWL-S Profile through two service parameters, one called securityCapability, that specifies what security requirements can be handled by the Web service, and the other called securityRequirement, that specifies what security requirements the Web service expects its clients should support. These two parameters can be used during Web service discovery in combination with the functional capabilities of the Web service. It is therefore possible to look for a Web service that “sells books and uses X.509.”

In the use case proposed in the call for position papers, the Web service has a requirement that all clients “support a reliable messaging protocol, and encrypt a specific header with WS-Security using a X.509 or user name security token in order to send an acceptable request message.” This requirement could be expressed, given the appropriate ontologies, using the securityRequirements parameter. The same approach holds for the representation of policies, with the only difference that we used Rei rather than P3P.

During discovery the capabilities of the requester would be matched against the requirements of the potential providers, and similarly the requirement of the requester would be matched against the capabilities of the providers. Only providers which match the capabilities and requirements of the requester would be selected [1].

One aspect that the use case does not address is that in interactions that extend through many message exchanges each message exchange may have different security and privacy requirements. For example, a web service may allow clients to exchange some information without encryption, while other may have to be encrypted. To address this problem we defined an encoding of security information in the inputs and outputs of the OWL-S processes description that is also reflected in the WSDL encoding . As a consequence, the encryption information is specific for each message[1].

References

[1] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan and Katia Sycara. Security For DAML Web Services: Annotation and Matchmaking. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[2] Kagal, L., Finin, T., Paolucci, M., Srinivasan, N., Sycara, K, Denker, G. "Authorization and Privacy for Semantic Web Services", IEEE Intelligent Systems, July/august 2004.

[3] Lalana Kagal *et al.*, "A Policy Based Approach to Security for the Semantic Web", InProceedings, *2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[4] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. World Wide Web Consortium (W3C) Recommendation. February 10, 2004. Available from http://www.w3.org/TR/owl-features/

[5] S. McIlraith., T.C. Son and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46--53, March/April, 2001.

[6] OWL-S Coalition. OWL-S 1.0 Draft Release. http://www.daml.org/services/owl-s/1.0/

[7] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The DAML-S Virtual Machine. In *2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[8] OASIS ServiceSecurity TC. *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0.* http://www.oasis-open.org/committees/download.php/8821/sstc-saml-sec-consider-2.0-cd-01.pdf

[9] The Universal Description, Discovery and Integration (UDDI) protocol. Version 3, 2003. http://www.uddi.org/

[1] Additional examples of OWL-S use in specifying constraints and capabilities are given at [6] and in a number of papers listed at http://www.daml.org/services/owl-s/pub-archive.html.