

Bringing Semantics to Web Services with OWL-S

DAVID MARTIN

Artificial Intelligence Center, SRI International, Menlo Park, CA, USA

MARK BURSTEIN

Intelligent Distributed Computing Department, BBN Technologies, Cambridge, MA, USA

DREW McDERMOTT

Computer Science Department, Yale University, New Haven, CT, USA

SHEILA McILRAITH

Department of Computer Science, University of Toronto, Toronto, Canada

MASSIMO PAOLUCCI*

DoCoMo Communications Laboratories Europe, Munich, Germany

KATIA SYCARA

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

DEBORAH L. McGUINNESS

Knowledge Systems, Artificial Intelligence Laboratory, Stanford University, Stanford, CA, USA

EVREN SIRIN[◇]

University of Maryland, College Park, MD, USA

NAVEEN SRINIVASAN[▽]

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Current industry standards for describing Web Services are focused on ensuring interoperability across diverse platforms, but do not provide a good foundation for automating the use of Web Services. Representational techniques being developed for the Semantic Web can be used to augment these standards. The resulting Web Service specifications enable the development of automated agents that can interpret descriptions of unfamiliar Web Services and then employ those services to satisfy user goals. OWL-S (“OWL for Services”) is a set of notations for expressing such specifications, based on the Semantic Web ontology language OWL. It consists of three interrelated parts: a profile ontology, used to describe what the service does; a process ontology and corresponding presentation syntax, used to describe how the service is used; and a grounding ontology, used to describe how to interact with the service. OWL-S can be used to automate a variety of service-related activities involving service discovery, interoperation, and composition. A large body of research has been based on OWL-S, and it has made possible the creation of a substantial set of open-source tools for developing, reasoning about, and dynamically utilizing Web Services.

Keywords: Web Services, Semantic Web, Semantic Web Services, OWL, OWL-S, Service Discovery, Service Composition

* Work performed while Paolucci was a member of the Robotics Institute, Carnegie Mellon University.

[◇] Since the writing of this paper, Sirin has moved to Clark+Parsia, LLC.

[▽] Since the writing of this paper, Srinivasan has moved to webMethods, Inc.

1 Introduction

Work on Semantic Web Services [59] lies at the intersection of two important trends in the evolution of the World Wide Web (WWW). The first trend is the rapid development of Web Service technologies, by which one agency makes available a service to others through an advertised protocol. The function of the service is often to supply information, but can also involve exchange or sale of goods or obligations. The second trend is the development of the Semantic Web, a network of sites containing computer-interpretable structured information (see, e.g., [12]).

Compiler-based tools make it easy for programmers to incorporate Web Services in their applications. To call a Web Service, a program must send it a message, or series of messages, normally encoded in XML (eXtensible Markup Language), which contain the information or request to be sent to the service; it then receives XML replies containing the returned values. The datatypes for the information passed between client and service, and its encoding using the XML-based SOAP (Simple Object Access Protocol) notation [16], can be described using WSDL (Web Services Description Language) [19]. Existing compilers for languages like Java and C# can automatically produce code to convert from a language's datatypes to their SOAP equivalents, making it almost as easy to call a Web Service as to call an ordinary subroutine.

One thing WSDL does not supply is a specification of *what happens* when a Web Service is used. The human programmer is supposed to figure that out from reading a natural-language description. Suppose you want an agent to solve problems such as the following:

- Make a periodic decision regarding the choice of which of two usual suppliers to acquire some raw materials from, while monitoring for the appearance of new alternative suppliers so that you can be informed if one offers the same materials at a lower price.
- Send a copy of your presentation to everyone attending the meeting in the room where the presentation is being given.
- Find vendors of products described using one or more controlled vocabularies (which might not be the same as the vocabularies used by the vendors to describe themselves).

Does the agent have to be human? Or can it be automated?

This is where the Semantic Web comes in. Where *interoperability* is the motivation for Web Services, *automation* of information use and *dynamic interoperability* are the objectives of the Semantic Web and Semantic Web Services. These goals are based on the idea of adopting standard languages for asserting *relationships* among entities that are declared to belong to *classes*. The information about the classes and the relationships among their members is captured in *ontologies*. The rules encoded there allow programs to make inferences about the relationships among the objects they discover at Web sites. A key step in the realization of the Semantic Web has been the development of standard languages for recording relationships and ontologies, including RDF (Resource Description Framework), RDFS (RDF Schemas), and OWL (Web Ontology Language) [56], all of which have reached final recommendation status through the standardization process at the World Wide Web Consortium (W3C).

For Web Service users, the logical next question is, Can we use these techniques to automate dealings with Web Services? The goal of the OWL-S Coalition¹ is to show that the answer is Yes. We describe the efforts of several other groups working

¹ More about the OWL-S Coalition can be found at: <http://www.daml.org/services/owl-s/>. In addition to the authors of this paper, the Coalition includes a number of other members.

on the related issues in Section 9. To move towards the realization of this vision, researchers have been developing languages, ontologies, algorithms, and architectures under the heading of *Semantic Web Services* [59]. OWL-S itself is an *OWL ontology for Services* [64], and as such provides a framework for describing both the functions and advertisements for Web Services, including a process specification language, a notation for defining process results and effects, and a vocabulary for connecting these descriptions in OWL with syntactic specifications of service message structure in WSDL (and in other existing notations).

OWL-S can be used to solve the problems presented above (although some of the Web infrastructure and reasoning tools required for complete solutions is still the subject of ongoing work). In each case, one would likely have a form-based interface between the human user and the automated agent doing the required reasoning.

- To support a regular choice between suppliers, an automated purchasing agent could use automated planning technology [63,84], to access a registry and find vendors potentially satisfying the user's objectives, then use OWL-S descriptions of those vendor's service interfaces to create an OWL-S process to poll those vendors. The agent would also periodically recompute the set of known vendors and notify the user when new vendors were found.
- To send a document to everyone currently in a particular room requires searching for a resource-management service for the building containing it, then querying it for people who have made their whereabouts in that building known (perhaps just to other people in the building)[50].
- To find a vendor of, say, "cutlery," it would be necessary to include companies that advertise the sale of "kitchenware." A software agent using OWL and OWL-S could apply ontological knowledge to broaden its search to include requests for providers of kitchenware, since this would be annotated as a *superClass* of cutlery. If a vendor used a different vocabulary where "knives" took the place of "cutlery," cross-ontological statements involving the *equivalentClass* construct of OWL would enable the agent to translate between the two.

This paper gives an overview of OWL-S, technologies based upon it, and selected research directions based upon it. It reflects the authors' design consensus as of OWL-S version 1.2, which is available on the OWL-S Web site [64].

The rest of the paper is organized as follows. In Section 2, we discuss the high-level objectives of OWL-S, and the overall structure of the framework, in terms of *profiles*, *processes*, and *groundings*. In Section 3 we discuss a central set of concepts for describing services, namely, their inputs, outputs, preconditions, and results or "IOPRs." In Section 4, we describe the structure and content of the profile ontology designed for use in advertising services. In Section 5, we present the process model ontology and a more human-readable notation, which allows the model to be viewed more easily as a recursive syntax for describing service processes rather than simply as an ontology. In Section 6 we discuss the ontology for describing groundings of services, enabling service calls to be translated into and out of message transport protocol languages such as SOAP (and its partner WSDL). In Section 7, we discuss how OWL-S can be used in various Web Service applications, focusing on service discovery and service composition. In Section 8, we survey a broad range of tools as well as research and prototype systems that have been developed based on OWL-S and extensions of OWL-S. In Section 9, we discuss related efforts to model aspects of Web Services. The paper ends with a brief summary and discussion of future directions in Section 10.

2 Design Objectives, Methodology, History

The high-level objectives of OWL-S include the following:

1. Provide a general-purpose representational framework in which to describe Web Services.
2. Support automation of service management and use by software agents.
3. Build on existing Web Services standards.
4. Build on existing Semantic Web standards.
5. Be comprehensive enough to support the entire “lifecycle” of service tasks.

OWL-S starts off as an *ontology*, that is, as a vocabulary plus axioms constraining its terms. The axioms are stated in OWL, so they talk about subclass hierarchies and how relations fit into them. OWL includes three sublanguages, OWL-Lite, OWL-DL, and OWL Full. The first two, but not the third, correspond to decidable description logics (DLs) [8]. To preserve decidability we have made an effort to keep OWL-S expressible in OWL-DL.

Nevertheless, we have discovered that for some purposes OWL is not expressive enough. To specify preconditions and effects, we need expressiveness comparable to that of first-order logic, and to specify processes effectively, we need many elements similar to those of programming languages. While many aspects of processes can be represented in OWL, and we have taken pains to make the process model ontology fit within the reasoning framework that OWL supports, we have found it necessary to allow for preconditions and effects to be expressed in logical languages that are outside of that framework.

Furthermore, it became increasingly clear that the amount of repetitive detail required to express the semantic elements of processes in terms of an ontology of classes and relations would make OWL-S unwieldy as a language for developers seeking to describe their services even as it provided precisely the details needed for computers to reason about those services using semantic descriptions. As a result, we have provided a “presentation syntax” for people to use when writing their process models, which can be expanded into OWL and associated rule representations for publication on the Semantic Web.

As an ontology, OWL-S does not make any commitment with respect to processing environment or Web Services architecture. Indeed, OWL-S has been used in very different configurations, from traditional Web Service architectures that adopt the Service Oriented Architecture (SOA) ‘triangle’ set of interactions (among a service registry, producer, and consumer), to Peer to Peer (P2P) systems [26] using a Gnutella-based architecture [70], to the multicasting-based Universal Plug and Play (UPnP) [50] systems used with wireless devices; to architectures that exploit a centralized mediator [46] [92] to manage the interaction with services and perform different forms of translation [78] between clients and services [71] [72].

OWL-S is not intended to replace existing Web Services or Semantic Web standards. Our goal is to enhance their utility by remaining compliant with existing standards and adding explicit semantics that are operational and understandable to computer programs. In our opinion, the Web Services community needs a little push to get to the point where they are comfortable with descriptions of entities outside the current world of Web Services specifications. OWL-S provides that push, right in the direction of the space the Semantic Web community already inhabits.

OWL-S has evolved through several releases. Work began in February 2001 on what was originally called “DAML-S,” (DAML stands for DARPA Agent Markup Language) because it was based on OWL’s predecessor DAML+OIL. Since then, there have been 4 releases of DAML-S followed by 4 subsequent releases of OWL-S after DAML+OIL evolved into OWL. As of this writing, the latest release is 1.2. Evolution has been rapid, driven by user feedback.

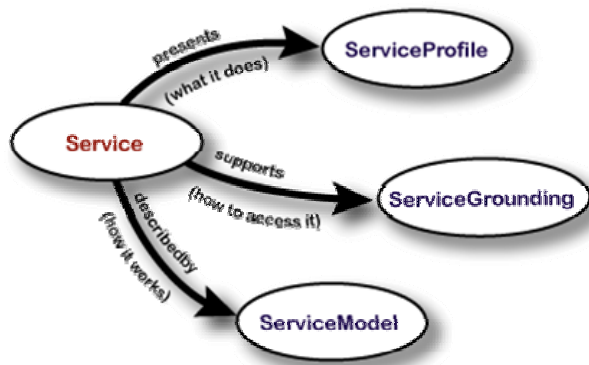


Figure 1. Top level of the service ontology.

does, the process model is used to describe *how the service is used*, and the grounding is used to describe *how to interact with the service*. The profile and process model are thought of as *abstract* characterizations of a service, whereas the grounding makes it possible to interact with a service by providing the necessary *concrete* details related to message format, transport protocol, etc. Figure 1 shows the relationships between the top-level classes of the ontology.

Each service described using OWL-S is represented by an instance of the OWL class *Service*, which has properties that associate it with a process model, groundings, and profiles for the service described. A process model provides the complete, canonical description of how to interact with the service at an abstract level, and the grounding supplies the details of how to embody those interactions in real messages to and from the service. Each profile may be thought of as a summary of salient aspects of the process model plus additional information that is suitable for purposes of advertising and selection. OWL-S allows a service to have multiple profiles so as to allow for tailoring of advertisements for different contexts. Similarly, a service may have multiple alternative groundings, which can even be added dynamically.

These notions are explained more thoroughly in the following sections. Additional details may be found in the Technical Overview at [64].

3 Inputs, Outputs, Preconditions, and Results (IOPRs)

Understanding all three components of an OWL-S service model requires understanding “IOPRs” – inputs, outputs, preconditions, and results. The *inputs* are the object descriptions the service works on; the *outputs* are the object descriptions that it produces. These descriptions are of course not known when the process is defined; all we can expect are the specification of their types (as OWL classes, at the most abstract level).

A *precondition* is a proposition that must be true in order for the service to operate effectively. An *effect* is a proposition that will become true when the service completes. These propositions are not in general statements about the values of the inputs and outputs, but about the entities referred to in the inputs and the outputs. For example, a service that requires a user to have an account and a credit card might have inputs `User` and `CreditCard` and precondition

```

hasAcctID(User, AcctID)
& validity(CreditCard, Valid)
  
```

2.1 Overall Structure of OWL-S

The OWL-S ontology includes three primary sub-ontologies: the service profile, process model, and grounding.

Roughly speaking, the profile is used to describe *what the service*

(Here `AcctID` and `Valid` are two other, local variables, to be explained below.) Note that the precondition requires the user to actually have an account with the service, with the given account ID. It also requires the credit card to have *some* validity state `Valid`; the significance of this is explained below.

In general, the effects of a service vary depending on conditions. In the case at hand, if the credit card is in fact valid, then let us suppose the service will replenish one's E-Z Pass balance to \$25. (E-Z Pass is the organization in the New York vicinity that runs the toll-booth lanes that use Radio Frequency Identification (RFID) technology to check off your car and charge your account every time you use a toll road or bridge.) If the credit card is not valid, then there is no change in the account balance. These are two separate *results*. The first might be indicated as

```
Valid = OK |->
    output(Outcome<=Success)
    & AcctBalance(25)
```

The second might be written

```
Valid = Not-OK |->
    output(Outcome<=Failure)
```

Here `OK` and `Not-OK` are values defined as elements of an OWL class `AccountValidity`.

```
<owl:Class rdf:ID="AccountValidity">
  <owl:oneOf rdf:parseType="Collection">
    <AccountValidity rdf:ID="OK"/>
    <AccountValidity rdf:ID="Not-OK"/>
  </owl:oneOf>
</owl:Class>
```

This example is overly simple, but illustrates some key points:

- The value of the output `Outcome` depends on events or tests that occur during the execution of the process, as does the effect `AcctBalance(25)`. They can't in general be predicted in advance, which is why we bundle them as results gated by conditions such as `Valid=OK`. (Note that this test is performed by the service, not the client, although *if* the client knew the credit-card status it could use that knowledge to predict what the service would do.)
- The values of variables are described as abstract OWL objects such as "Not-OK." These objects must be connected to more concrete messages in order to be sent or received by an actual programmed agent.
- The propositions referred to in preconditions and effects can refer to arbitrary facts about the world, and are not confined to talking about data structures. That's perhaps not entirely obvious in our example, which refers to statuses and balances, but it will become clearer as we go.

The last point deserves emphasis. The whole point of OWL-S is to allow agents to use Web Services to accomplish real-world goals. The vision is to have the ability to task an agent to make reservations, purchase items, schedule meetings, and make other similar arrangements. Goals are stated in terms of propositions we want to make true, constraints on the actions and resources to be used, and objective functions to be optimized. Because the goals are described in terms of real-world entities, the services must be as well, with connections down to concrete messages whose transmission will accomplish what the user wants.

Now that we've outlined IOPRs, we can get back to discussing the three components of an OWL-S service model, each of which refers to some subset of the IOPRs: the profile, the process model, and the grounding. We'll take them in that order.

4 The Service Profile

The OWL-S profile provides a set of concepts to specify capabilities of services, with the goal of supporting capability-based discovery. Specifically, the OWL-S profile allows service providers to advertise what their services do, and service requesters to specify what capabilities they expect from the services they need to use. Crucially, the profile provides an explicit description of those capabilities, so that they do not have to be extracted from incidental properties such as the name of the service, or the company that provides it. By exploiting the structure of OWL-S profiles and their references to OWL concepts, a discovery process can find those services that are most likely to satisfy the needs of a requester.

The overall structure of the OWL-S profile is shown in Figure 2, in which ovals represent existing OWL classes and arcs represent OWL properties. A profile is an instance of the class `Profile` (or an instance of a subclass of `Profile`). The principal elements that occur in a profile include the *profile type*, which is the particular subclass of `Profile` that's being instantiated, and the *inputs*, *outputs*, *preconditions*, and *results* associated with the service. (The profile is not required to list all IOPRs; that is, it is free to omit some IOPRs if they are deemed to be unimportant for purposes of advertisement and discovery.) In addition, a profile may include a *product type*, where appropriate; *service categories* that may be used to refer to existing business taxonomies that may not be codified in OWL, such as North American Industry Classification System (NAICS - <http://www.census.gov/epcd/www/naics.html>); and a variety of *service parameters* that may be used to specify additional features of the service. Subclasses of `Profile` may be created for particular domains or types of services, and specialized with appropriate properties. For example, for shipping services, there might be a `ShippingServiceProfile` subclass with the additional prop-

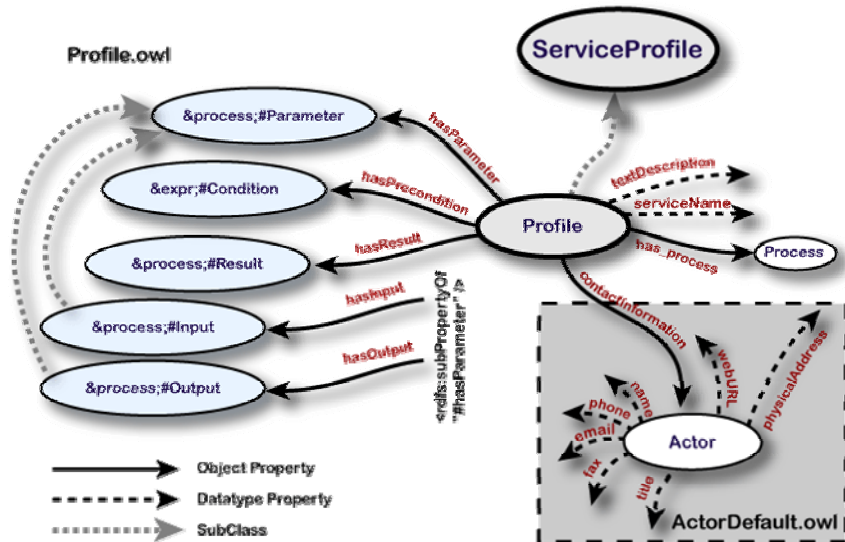


Figure 2. Top-level structure of the OWL-S profile.

erty `geographicRegionServed`, with `GeographicRegion` (from an appropriate online ontology) as its range.

Broadly speaking, the OWL-S profile describes three aspects of a service. The first one is the *functional* aspect which is represented using IOPRs. The functional aspect represents the information transformation computed by the service from the inputs that the service expects to the outputs it generates, and the transformation in

the domain from a set of preconditions that need to be satisfied for the service to execute correctly, to the effects that are produced during the execution of the service. A purchase in an e-store illustrates a typical set of these transformations: the inputs are the name of the desired product and a credit card number, the output is a receipt of purchase, the preconditions specify that the credit card is valid, and the effects specify that the credit card will be charged, and that the goods will change ownership.

The second aspect, the *classification* aspect, supports the description of the type of service as specified in a taxonomy of businesses (or other suitable domain-specific ontologies). The classification of the service is indicated by the profile type and/or the service categories. Using these elements, the provider can specify explicitly what type of service it provides, and the requester can specify the type of service with which it would like to interact.

The third aspect of the profile is the *non-functional* aspect, which makes distinctions between services that do the same thing but in different ways or with different performance characteristics. Examples of non-functional aspects of a service include the service security and privacy requirements [31], the precision and timeliness (quality of service) of the service provided, the cost structure, and aspects of provenance, such as those captured in the Proof Markup Language [74]. Since it is impossible to provide a complete set of attributes for the representation of service parameters (many of which are domain-specific), the solution adopted by OWL-S is to provide an extensible mechanism that allows the providers and requesters to define the service parameters they need.

A partial example of a profile for a fictitious airline booking service, expressed in the RDF/XML syntax of OWL, is shown in Figure 3. The profile, an instance of `FlightReservationProfile` (a subclass of `Profile`), describes inputs specifying a departure and arrival airport, and a reservation as output. The crucial aspect of this example is that the values used to specify the types of the OWL-S inputs and outputs are the URIs of concepts defined in some ontology. In this example, the ontology used is the fictitious ontology `http://fly.com/Onto`. The advantage of using concepts from Web-addressable ontologies, rather than XML schemata, is that the descriptions use terms whose provenance is globally available, which refer to entities like airports instead of, say, “three-letter codes.” This, together with the fact that these terms are arranged in a semantic abstraction hierarchy, enables the discovery process to avoid matching failures due to syntactic differences and to verify that terms used by providers and requesters are drawn from the same vocabulary. If the vocabularies are different, it may still be able to find appropriate services as long as the different terms are suitably related by their ontologies.

```

<FlightReservationProfile rdf:ID="BravoAir">
  <serviceName>Bravo Air</serviceName>
  <contactInformation rdf:resource="#BAco"/>
  <hasInput>
    <process:Input rdf:about="&bravoAirProcess;#depart">
      <process:parameterType rdf:datatype="&xsd;#anyURI">
        http://fly.com/Onto#Dep_Airport
      </process:parameterType>
    </process:Input>
  </hasInput>
  <hasInput>
    <process:Input rdf:about="&bravoAirProcess;#arrive">
      <process:parameterType rdf:datatype="&xsd;#anyURI">
        http://fly.com/Onto#Arr_Airport
      </process:parameterType>
    </process:Input>
  </hasInput>
  <hasOutput>
    <process:Output rdf:about="&bravoAirProcess;#reserve">
      <process:parameterType rdf:datatype="&xsd;#anyURI">
        http://fly.com/Onto#Reservation"
      </process:parameterType>
    </process:Output>
  </hasOutput>
  .....
</FlightReservationProfile>

```

Figure 3. A simple, partial OWL-S profile.

5 The Process Model

Once a Web agent has identified a service as likely to be relevant to its goals, it needs a detailed model of the service to determine whether the service can do the job, and, if so, what constraints must be satisfied and what pattern of interactions will be required to get it to do that job.

An OWL-S process specifies, among other things, the possible patterns of interaction with a Web Service. There are two sorts of processes that can be invoked: atomic and composite. An *atomic* process is one that has no internal structure. It corresponds to a single interchange of inputs and outputs between an agent and the service. A *composite* process consists of a set of component processes linked together by control flow and data flow structures. The control flow is described using typical

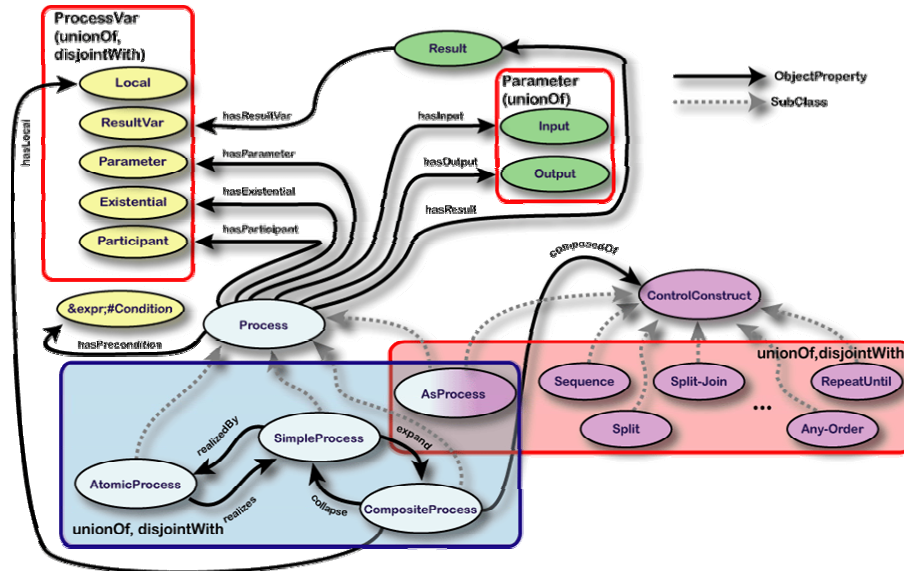


Figure 4. Top-level structure of the OWL-S process model.

programming language or workflow constructs such as sequences, conditional branches, parallel branches, and loops. Data flow is the description of how information is acquired and used in subsequent steps in this process. A third type of process, the *simple* process, can be used to provide abstracted, non-invocable views of atomic or composite processes. Simple processes are not discussed further in this paper; the reader is referred to the Technical Overview at [64] for information about them.

The concepts used to describe a process are themselves terms in the OWL-S ontology. The OWL-S definition of Process has a set of associated features (inputs, outputs, preconditions, results) linked to the Process concept by OWL properties (`hasInput`, `hasOutput`, etc.). Composite processes are related to their top-level control structure using the `composedOf` property, and other properties, such as `components`, are used to show the relationships between control structures. Other properties relate the process model to corresponding messages (referenced by the grounding), and to advertised capabilities descriptions in the corresponding OWL-S profile. Figure 4 shows some of the principal elements of the process model.

5.1 OWL-S Process Model Presentation Syntax

Machine interpretation of OWL-S process models relies on systems that read the process definitions expressed in OWL's RDF/XML serialization syntax, which is not designed for human consumption. Even simple processes can become swamped in reefs of angle brackets that make it hard for people to read and understand what each process does. For this reason, we have provided a *presentation syntax* (sometimes

called “surface syntax”) that makes OWL-S processes look like programs in a high-level language. This more procedural presentation syntax for OWL-S is easily translatable into the canonical RDF/XML syntax. Our exposition style here is to make the presentation notation primary, explaining the translations of its terms as we go.

For example, namespaces are a crucial feature of most XML applications, and OWL-S, which directly utilizes OWL, is no exception. In the presentation syntax we use the notation

```
with namespaces
(uri"http://www.daml.org/services/owl-s/1.2/congo.owl",
 service: uri"http://www.daml.org/services/owl-s/1.2/Service.owl",
 owlproc: uri"http://www.daml.org/services/owl-s/1.2/Process.owl",
 profile: uri"http://www.daml.org/services/owl-s/1.2/Profile.owl",
 rdf: uri"http://www.w3.org/1999/02/22-rdf-syntax-ns")
{
  define atomic process ExpressCongoBuy ...
  define ...
}
```

In the RDF/XML version, these declarations get attached to the outermost RDF element in the usual way:

```
<rdf:RDF
  xmlns="http://www.daml.org/services/owl-s/1.2/congo.owl"
  xmlns:service
    ="http://www.daml.org/services/owl-s/1.2/Service.owl"
  xmlns:owlproc
    ="http://www.daml.org/services/owl-s/1.2/Process.owl"
  xmlns:profile
    ="http://www.daml.org/services/owl-s/1.2/Profile.owl"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <process:AtomicProcess rdf:ID="ExpressCongoBuy" ...>
  ...
  </process:AtomicProcess>
  < ...>
  </ ...>
</rdf:RDF>
```

The `with-namespaces` expression can be used at any level, and it always gets translated into `xmlns` declarations, attached to whatever RDF description is most convenient. Once a namespace has been declared, we can then use prefixes with colons according to the XML convention. However, there are fewer places in the presentation syntax where names qualified by namespace prefixes (so-called *qnames*) are needed. For instance, reserved words of the notation are always assumed to be in the OWL-S namespace.

5.2 Atomic Processes

A process model is a description of what a client must do in order to cause a service, or a collection of services, to do something. As discussed in Section 3, all processes have inputs, outputs, preconditions, and results (IOPRs). Here is a specific example, from the “CongoBuy” scenario, an artificial example that represents a book-buying service. The complete service is described in [64].

```
with_namespaces
(uri"http://www.daml.org/services/owl-s/1.2/examples/CongoBuy.owl",
 process: uri"http://www.daml.org/services/owl-s/1.2/Process.owl",
 books: uri
  "http://www.daml.org/services/owl-s/1.2/examples/BookConcepts.owl",
  . . .
```

```

rdf:      uri"http://www.w3.org/1999/02/22-rdf-syntax-ns")
{
(define atomic process ExpressCongoBuy
  (inputs: (ISBN - books:ISBN
            SignInInfo - SignInData
            CC-Number - xsd:decimal
            CC-Type - CreditCardType
            CC-ExpirationDate - xsd:YearMonth),
  locals: (AcctID - CustomerAcctID
            CardUsed - CreditCard),
  precondition:
    (hasAcctID(SignInInfo, AcctID)
     & credit-card-number(CardUsed, CC-Number)
     & validity(CardUsed, valid))
  outputs: ...
  ...);
...}

```

Atomic processes are defined entirely by their inputs, outputs, local variables, preconditions, and results. All these fields are optional. Dashes are used to separate variable names from their conceptual types, which should be OWL concepts. For example, the input parameter ISBN is of type 'books:ISBN', which is a concept defined in the namespace 'books', an OWL domain ontology. Local variables are those bound by occurring in preconditions rather than by being passed to the process explicitly.

The results of a process are specifications of its possible outcomes, gated, as explained in section 3, by *result conditions*. (To recapitulate: the distinction between preconditions and result conditions is that preconditions are things that the client must verify before deciding to request the service, whereas result conditions are things that are checked by the service in the course of execution that impact the outcome produced.) Results consist of effect and output specifications. Effects are simply propositions which become true, while the information conveyed to output parameters is indicated by the special notation:

```
output( output-parameter <= rdf-description ).
```

Result conditions are specified by the notation $p \mid \rightarrow r$ (read "when p , then r "), which means that if p is true in the situation checked by the service before it returns an answer, then it will have result r , where r consists of a set of outputs and effects.. Note that from the perspective of a client reading this process description, and reasoning about it when interacting with the service, what it will observe is the output message that was emitted. At that point, it could infer, based on the process description, that a particular result condition p had obtained, and that the listed effects of the process for that result condition should be true.

Our Congo example might have the following outputs and conditional results:

```

define atomic process ExpressCongoBuy
  (...
  precondition:
    (hasAcctID(SignInInfo, AcctID)
     & credit-card-number(CardUsed, CC-Number)
     & validity(CardUsed, valid))
  outputs (Status - CongoBuyOutputType),
  result:
    (forall (book - books:Book)
     (hasISBN(book, ISBN) & inStockBook(book)
      |-> output(Status <= OrderShippedAcknowledgement)
           & shippedTo(addressOf(AcctID), book))

```

```

&
(hasISBN(book, ISBN) & ~inStockBook(book)
|-> output(Status <= NotifyOutOfStockBook(ISBN))))

```

Here, two result conditions are described, one where the book is in stock, and one where it is not (`~inStockBook(book)`). In the first case, the output `OrderShippedAcknowledgement` is produced, which is the presentation syntax version of an RDF expression representing the meaning of the associated WSDL output message. When the client sees this message, it is licensed to infer that the associated `shippedTo` expression is true, and also that the condition `inStockBook` was true when the service executed. In the other case, a different message would have been sent by the service, whose meaning is captured by the OWL concept `NotifyOutOfStockBook`.

The equivalent representation of this process in OWL's RDF/XML syntax is as follows:

```

<process:AtomicProcess rdf:ID="ExpressCongoBuy">
  <process:hasInput>
    <process:Input rdf:ID="ExpressCongoBuyBookISBN">
      <process:parameterType rdf:datatype="xsd:anyURI"
        &profileHierarchy;#ISBN
      </process:parameterType>
    </process:Input>
    ...
  <process:hasPrecondition>
    ...
  </process:hasPrecondition>
  ...
</process:AtomicProcess>

```

In the RDF/XML version, instead of results being produced by a recursive grammar, each result is a separate object, with associated when-conditions indicated using the `inCondition` property. For reasons that will become clear in a moment, we omit the details of these properties, but for the RDF/XML equivalent of `output` we can be a bit more complete:

```

<process:AtomicProcess rdf:ID="ExpressCongoBuy">
  ...
  <process:hasResult>
    <process:Result>

      <process:inCondition>
        ...
      </process:inCondition>
      <process:hasEffect>
        ...
      </process:hasEffect>
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#ExpressCongoBuyStatus"/>
          <process:valueData
            rdf:resource="#OrderShippedAcknowledgement"/>
          </process:OutputBinding>
        </process:withOutput>
      </process:Result>
    </process:hasResult>
  </process:AtomicProcess>

```

This is considerably more verbose than `Status <= OrderShippedAcknowledgement`, but the reduction of the process notation to OWL descriptions allows us to be clear about how entities in the process model are linked to entities in the profile (Sec-

tion 4) and the grounding (Section 6). It also allows process descriptions to be serialized in the standard RDF/XML notation of OWL.

We have left blanks in the RDF wherever the presentation syntax includes a logical formula. There are many alternative approaches to expressing formulas and planting them inside RDF, including N3 [11], RuleML [79], SWRL [42], SWRL-FOL [73], and SPARQL [75]. Some of these use the idea of “layering” the notation “on top of” RDF by encoding its syntax in RDF, but there is widening acceptance of the idea that formulas can be embedded as (string or quoted-XML) constants. In this paper we will avoid the details of formula translations.

The mention of formula representation brings us to the key question of what sorts of inference we expect the OWL-S process models to support. Some of these questions depend on the interpretation of composite processes, which we will discuss shortly. But there are plenty of tasks that can be supported using purely atomic models of processes. Given a set of goals and a set of process specifications, one can find or check a described sequence of process instances to determine if it can accomplish those goals. Finding one is the classic *AI planning* problem, which in this context is called *dynamic service composition* (see Section 7). Another form of inference we expect OWL-S to be useful for is *plan recognition*: given one or more actions by an agent, infer what goals the agent might have.

5.3 Composite Processes

The possibilities for reasoning about processes broaden considerably when we allow processes to have internal structure. Such processes are called *composite processes*:

```
define composite process P
  (inputs: (...))
  (outputs: (...))
  (preconditions: (...))
  (results: (...))
  )
  {
    ---body---
  }
```

In the simplest case, the *body* of a composite process consists of a *sequence* of actions, separated by semicolons. The simplest sort of action is to execute another process. This kind of act is called a *perform act*.

```
define composite process P(inputs-outputs-preconditions-results)
  {
    perform Q(V <= E, ...);
    perform R(X <= F, ...);
    ...
  }
```

This notation indicates that the first step of the composite process *P* is to perform the process *Q* (which may or may not be atomic) with some specific parameters. Input parameter *V* of process *Q* should get value *E*, and so forth. Inputs to these steps may be the same as or derived from an input to the composite process, they may be constant values, or, for steps other than the first, they may be the result of an earlier step in the process. Here is a very simple composite process in our presentation syntax. The XML/RDF serialization of this simple process would take nearly a full page to show, but see the OWL-S Website [64] for examples.

```

define composite process BuyBook
  (inputs: (ISBN - books:ISBN, user - access:UserID))
  { perform Login(UID <= user);
    perform requestBuyBook(BookID <= ISBN) }

```

If a planning system is given a set of composite processes describing standard ways for interacting with Web Services, it can generate a set of instances of those processes that will accomplish its objectives. This paradigm is called *hierarchical planning*; we discuss related work in this area in Section 0.

In solving the problem of *nondeterminism reduction*, an inference engine verifies that in a particular circumstance, there is a set of choices of actions and their orderings that will allow a plan with underdetermined structure to be executed successfully [77]. Nondeterminism reduction is often more efficient than planning, because the search space, i.e. the number of process choices the planner must make, is significantly reduced. That is, all the possibilities are laid out in advance and the system must merely narrow them to the point that all remaining execution traces result in a situation satisfying the goal. There are a couple of OWL-S control constructs for expressing nondeterminism. One is the *choice* construct, which appears in the presentation syntax as:

$$P_1 \text{ ;? } P_2 \text{ ;? } \dots \text{ ;? } P_k$$

and is executed properly when one of the P_j is executed. Another sort of nondeterminism is embodied in the *any order* construct:

$$P_1 \text{ ||; } P_2 \text{ ||; } \dots \text{ ||; } P_k$$

which means that all of the P_j are to be executed in *some* order. Other constructs in OWL-S include *conditional execution*:

$$\text{If } E \text{ then } A \text{ else } B$$

split-join:

$$P_1 \text{ ||> } P_2 \text{ ||> } \dots \text{ ||> } P_k$$

and *split*:

$$P_1 \text{ ||< } P_2 \text{ ||< } \dots \text{ ||< } P_k$$

The last two indicate parallel (interleaved) execution of the P_j . The difference between them is that *split-join* is finished when all the P_j finish, whereas *split* terminates immediately, after the subprocess instances are spawned.

5.4 The Semantics of OWL-S

OWL-S is an OWL-DL ontology. Since OWL has a well-defined semantics, so too does OWL-S. However, this is not the complete story.

OWL-DL is a reasonably expressive language, whose expressive power was limited *by design* in order to balance expressiveness against the desire for decidability and the computational complexity of the description logic inferences it must support. As a result, many sophisticated definitions and inter-relationships between OWL concepts cannot be described within the OWL language. While the OWL-S process model sub-ontology has a well-defined OWL semantics, if you actually “do the math” you will discover that the meaning of “Split” (or “||<” in the presentation syntax) is that it is a class with certain relationships to other classes, such as `ControlConstruct`, `Sequence`, `SplitJoin`, etc. The formal representation of

the differences between these terms, let alone what it means to execute them, is missing.

We have explored several solutions to this problem. The intended interpretation of the process vocabulary was defined in written form in the Technical Overview of OWL-S [64]. It has also been defined by translation to first-order logic using the situation calculus, a language for reasoning about action and change (e.g., [77]); by using Petri nets [61]; and by using an execution semantics that incorporates subtype polymorphism [5]. More recently, the semantics of OWL-S was described in the Process Specification Language (PSL) (ISO 18629) a formally axiomatized process ontology [40]. The existence of these specifications makes it possible to use automated systems for reasoning about processes that are built around first-order logic, Petri nets, or other popular formal frameworks.

6 The Grounding

The grounding ontology of OWL-S is used to specify how the abstract information exchanges that are detailed by atomic-process descriptions are realized by concrete information exchanges between the agent requesting a service and the deployed Web Service it has chosen to use. A grounding maps each OWL-S atomic process (in a collection of processes associated with a service) to a WSDL operation (defining input and output messages), and relates each OWL-S process input and output to elements of the XML serialization of the input and output messages of that operation. The purpose of this mapping is to enable the translation of semantic inputs generated by an agent into the appropriate WSDL messages for transmission to the service, and the translation of service output messages back into appropriate semantic descriptions (i.e., OWL descriptions) for interpretation by the agent.² It should be noted that a grounding can be supplied at runtime, so as to allow for dynamic binding.

We emphasize that an OWL-S/WSDL grounding involves a complementary use of the two languages in a way that is in accord with the intentions of the authors of WSDL. Both languages are required for the full specification of a grounding, because the two languages do not cover the same conceptual space. WSDL, by default, specifies message types using XML Schema, whereas OWL-S allows for the definition of abstract types as (description logic-based) OWL classes. Thus, it is natural that an OWL-S/WSDL grounding uses OWL classes as the abstract types of message parts declared in WSDL, and then relies on WSDL binding constructs to specify the formatting of the messages.

² OWL-S allows for groundings to other invocation frameworks besides WSDL; for example, at least one project has utilized a grounding that bottoms out in UPnP. Here, we discuss only the WSDL grounding since that is included with the online releases, and is by far the most widely used.

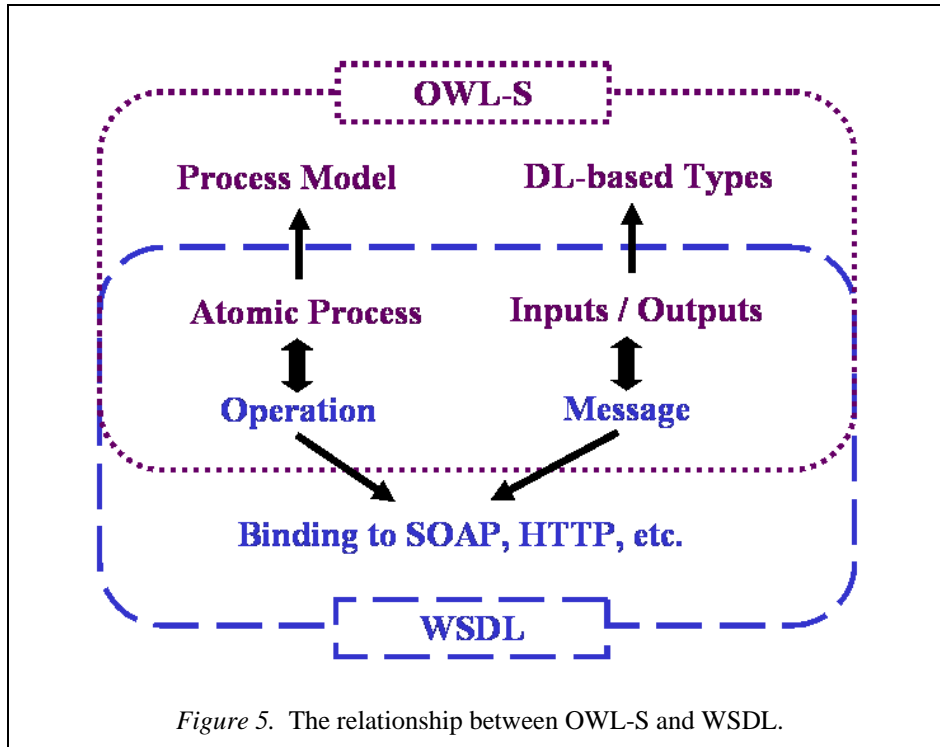


Figure 5. The relationship between OWL-S and WSDL.

Figure 5 shows the overlapping relationship between OWL-S and WSDL. WSDL *operations* correspond to OWL-S *processes* (and here we will confine ourselves to *atomic processes*). The *inputs* and *outputs* of an OWL-S process correspond to *messages* in WSDL. The grounding describes the mapping between the DL-based objects referred to in process I/O specs and the complex datatypes referred to in WSDL message definitions.

On the WSDL³ side, the most abstract description of a Web Service is in terms of *messages*, *operations* and *port types*. For a service supplying information about book availability, the WSDL port type is of the “request-response” variety, meaning that the client sends one message and receives one reply (just like an OWL-S atomic process).

```

<portType name="CongoCheckPortType">
  <operation name="CheckBook"
    owl-s:owl-s-process
    ="congoOwl:CongoCheck">
    <input message="CongoCheckInput" />
    <output message="CongoCheckOutput" />
  </operation>
</portType>

```

The bold-face attributes are extensions to WSDL that were required to connect it to OWL-S.⁴ The `input` and `output` attributes declare the presence and types of the request to and response from the Web Service.

³ The example that follows uses WSDL 1.1, as do existing tools; when the WSDL 2.0 standard is finalized, we will provide grounding constructs for mapping to its novel elements.

⁴ Fortunately, WSDL is extensible at various points, so almost all the changes we require are just a matter of defining new namespaces. For the exceptions, see [47].

The messages are described in `message` elements:

```

<message name="CongoCheckInput">
  <part name="BookISBN"
        type="xsd:string"
        owl-s:owl-s-parameter
        ="CongoOwl:In-BookISBN"/>
  <part name="AcctID"
        type="xsd:string"
        owl-s:owl-s-parameter
        ="CongoOwl:In-AcctID"/>
</message>
<message name="CongoCheckOutput">
  <part name="Availability"
        type="xsd:Boolean"
        owl-s:owl-s-parameter
        ="CongoOwl:Out-Avail"/>
</message>

```

Here we've chosen the simple type `xsd:string` for both inputs to the process; space doesn't permit exploration of more realistic datatypes.

These snippets of XML just displayed are WSDL, not RDF, as extended to indicate the correspondence of WSDL message parts with OWL-S Parameters. On the OWL-S side, we must also specify the *grounding* of the Web Service, which tells OWL-S tools which messages the inputs and outputs of a process correspond to, and how to translate the values:

```

<WsdAtomicProcessGrounding rdf:ID="CongoCheckGrounding">
  <owlsProcess rdf:resource="&congo;#congoCheck"/>
  <wsdlOperation>
    <WsdOperationRef>
      <portType>
        <xsd:uriReference
          rdf:value
            ="&congoWsd;#CongoCheckPortType"/>
      </portType>
      <operation>
        <xsd:uriReference
          rdf:value="&congoWsd;#CheckBook"/>
      </operation>
    </WsdOperationRef>
  </wsdlOperation>

  <wsdlInputMessage
    rdf:resource
      ="&congoWsd;#CongoBuyInput"/>
  <wsdlInput>
    <wsdlInputMessageMap>
      <owlsParameter rdf:resource="&congo;#In-BookISBN"/>
      <wsdlMessagePart>
        <xsd:uriReference
          rdf:value="&congoWsd;#BookName"/>
      </wsdlMessagePart>
    </wsdlInputMessageMap>
  </wsdlInput>
  <wsdlInput>
    <wsdlInputMessageMap>
      <owlsParameter rdf:resource="&congo;#AcctID"/>
      ...
    </wsdlInputMessageMap>
  </wsdlInput>

  <wsdlOutputMessage rdf:resource

```

```

                                ="&congoWsd1#CongoCheckOutput" />
<wsdlOutput>
  <wsdlOutputMessageMap>
    <owlsParameter rdf:resource="&congo#Out-Avail" />
    <wsdlMessagePart>
      ...
    </wsdlMessagePart>
  </wsdlOutputMessageMap>
</wsdlOutput>
</Wsd1AtomicProcessGrounding>

```

Although the RDF-based OWL-S is more verbose than WSDL, it should be clear that it expresses related but complementary information. The `wsdlOperation` property of a `Wsd1AtomicProcessGrounding` object specifies the portType/operation pair from the WSDL spec. The `wsdlInputMessage` and `wsdlOutputMessage` properties specify how the atomic process maps into a request/response pattern. The `wsdlInput` and `wsdlOutput` properties specify "message maps," which are just associations between OWL-S parameters and WSDL message parts. This information allows an OWL-S agent to find in the WSDL spec the information about the concrete embodiment of the abstract parameters.

Further details about OWL-S groundings may be found in [47] and in the Technical Overview [64].

7 Technology and Research Directions Based On OWL-S

OWL-S has been employed in a wide range of research efforts aimed at achieving the kind of automation we described in Section 1. In this section, we describe some of the explorations that have taken place.

7.1 Architecture and Service Enactment

In this section, we describe how OWL-S can be used in a typical Web Service interaction in which the service requester formulates a request based on its goals, queries a registry such as UDDI to find a provider, and finally interacts with the provider. The participants and their roles are shown in Figure 6. The main steps in this process are:

1. **Formulation of a request:** The client has a goal that needs to be solved, and it would like to find a Web Service that can satisfy the goal. To find such a Web Service, the client expresses its goal as an OWL-S profile. The translation of a goal into a profile is an abstraction process from a goal to the capabilities that are required to satisfy that goal. Such capabilities are expressed in terms of the results and information produced as well as the quality of service that the client expects from a provider. An algorithm for generating these abstractions is described in [71].
2. **Service Request:** The client uses the service profile that it constructed to query a registry that contains a collection of offered services.
3. **Matching:** The registry compares the request to its library of service profiles and returns (URI references for) candidates that might achieve that goal. The matching process relies on the relations between the terms used to specify the advertisements and those in the request, both of which are defined using OWL ontologies ([43] [45] [67]).

4. **Selection:** The client analyzes the candidates returned by the registry and selects a Web Service to interact with. Since it is possible that none of the services found will precisely satisfy all the requirements and objectives of the requester, the selection process may require consideration of a number of different trade-offs [9].
5. **Invocation:** The client determines how to make a request to the selected service by unifying its goals and preferences with the effects specified in the service process model, to determine a semantically valid set of inputs to the server. These inputs are then mapped onto a WSDL input message pattern using the OWL-S service grounding, and the resulting message is sent.
6. **Reply:** The service receives the request, and determines whether it can perform the request. It may acknowledge the request, send an error, request additional information, or (generally, on completion) send a reply stating the service results.
7. **Reply Interpretation:** Upon receiving the reply, the client parses the reply, in accordance with its WSDL specification, and uses the grounding specification to map it into the abstract outputs specified in the process model [65]. If the invocation of the service is successful, the effects of the service hold true in the client's world. This additional knowledge may satisfy preconditions for other services.
8. **Goal satisfaction:** The client uses the instantiated outputs to decide whether its goal has been achieved successfully, or whether there is a need for additional interactions with the Web Service or for searching for another Web Service [89].

When the process model of a service is a composite process, steps 5, 6 and 7 may be repeated a number of times, since the interaction between the client and the server may require numerous information exchanges. For example, any interaction with a B2C Web site such as Amazon's requires the buyer to fill out multiple forms specifying such things as which book to buy, selecting the appropriate edition, specifying where to ship the book, describing which credit card to use, etc. The client may also need to interact with multiple Web Services concurrently to find out which one will provide the best result, or with the best quality of service. For more information about the semantics and invocation of composite processes refer to [65].

This interaction process is consistent with the Service Oriented Architecture (SOA) interaction protocols that are used with UDDI, WSDL and SOAP. It reflects the similar roles played by the service profile representation of Web Services in OWL-S and the UDDI representations of Web Services [68][87].

The difference here is that UDDI interactions require human software developers do the job of interpreting the registry results and creating the calls to the services. Figure 6 shows how the process is implemented using OWL-S for more dynamic Web Services interactions. An application using OWL-S should not require a programmer to query UDDI, read the WSDL models found there, and implement those interfaces. The client software agent is responsible for the interaction with the OWL-S registry, for the determination of which candidate services are most appropriate, for the determination of the information required to invoke each service, and for the interpretation and response to messages returned by the service. Variations on this execution model for OWL-S have been implemented by a number of academic and industry researchers (e.g. [65], [84]).

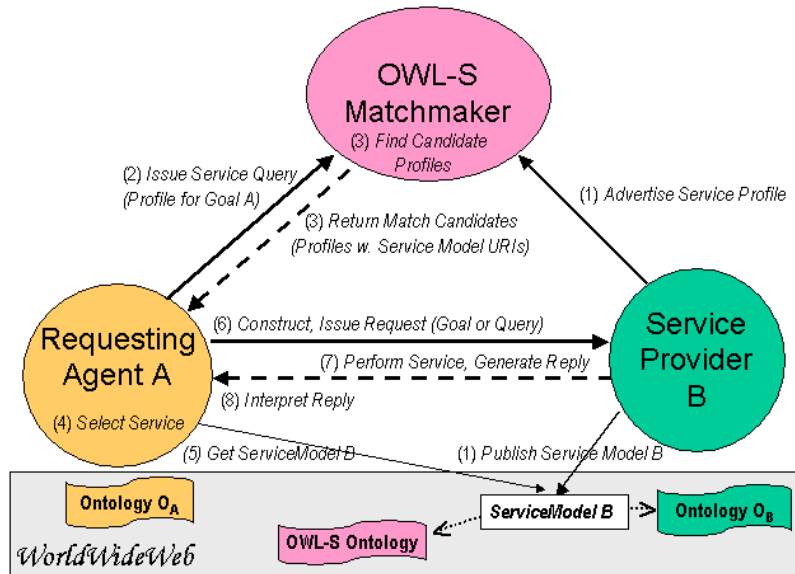


Figure 6. Typical OWL-S Semantic Web Service interaction model.

7.2 Ontology-Based Service Discovery

The first few phases of the process described in the previous section are areas where Semantic Web Service discovery based on the OWL-S profile sub-ontology may be expected to provide value-added over existing Web Service discovery standards. Capability-based matching enables agents to find services that aptly match their requirements for particular results, while semantic matching allows flexibility in identifying the service parameters needed (and available to the client) to specify the criteria of the requested service.

Universal Description, Discovery and Integration (UDDI) provides a number of means of searching its registry of services; services can be searched by name, by business, by bindings or by tModels. For example, it is possible to look for services that adhere to Rosetta Net specifications or services that sell new cars. Unfortunately, the search mechanism supported by UDDI is limited to keyword matches and does not support any inference based on the taxonomies referred to by the tModels. For example a car selling service may describe itself as “New Car Dealers” which is an entry in the North American Industry Classification System (NAICS), but a search for “Automobile Dealers” services will not identify the car selling service despite the fact that “New Car Dealers” may be regarded as a subtype of “Automobile Dealers”. Since OWL-S uses OWL ontologies to describe functional and non-functional properties it overcomes the limitations of syntactic search by providing semantic search based on subsumption relations.

Broadly speaking, there are two ways to represent capabilities of Web Services. The first approach provides an extensive class hierarchy with each class representing a set of similar services. Using such a hierarchy, services may be defined as instances

of classes that represent their capabilities. Amazon, for example, may advertise itself as a member of an OWL class for bookselling services. (For advertising purposes, such a class would be a subclass of `Profile`.) The second way to represent a service's capabilities is to provide a generic description of its functionality in terms of the state transformations that it produces. For example, Amazon may specify that it provides a service that takes as input a book title, author, address and a valid credit card number, and produces a state transition such that the book is delivered to address, the credit card is charged the book price, and the book delivered changes its ownership. Despite their differences, both ways to represent capabilities use domain-specific ontologies to provide the connection between what the Web Service does and the general description of the environment in which the Web Service operates.

There are trade-offs between these two styles of representation in service discovery. The use of an explicit ontology of capabilities facilitates the discovery process since the matching process is reduced to subsumption tests between the capabilities expressed in service profiles and those expressed in a service discovery query. Both descriptions reference concepts in the ontologies of the domain the service covers. On the other hand, enumerating all possible capabilities even in restricted domains may be difficult. For example, consider the problem of representing translation services from a source language LS to a target language LT. Assuming n possible languages, there are n^2 possible types of translation services. A service taxonomy might have different classes of service for each pair of languages that could be translated, or it might just represent "translation services" as one general category, with explicit properties that allow particular services to describe the languages that they can translate from and translate to. This latter approach is consistent with describing the capability in terms of a state transformation. It distinguishes the translators by describing how they produce different kinds of results. OWL-S provides support for both styles of service description.

A number of capability matching algorithms using OWL-S service profile descriptions have been proposed. In general, they exploit one of the two views of the capabilities described above. Matching algorithms, such as described in [43], assume the availability of ontologies of functionalities to express capabilities. Matching between a request and the available advertisements is reduced to their subsumption relation. Different degrees of match are detected depending on whether the advertisement and the request describe the same capability or whether one subsumes the other.

Other matching algorithms, such as in [67], [87], and [36], assume that capabilities are described by the state transformation produced by the Web Service. Paolucci et al. [67] describe a matchmaking algorithm that compares the state transformations described in the request to the ones described in the advertisements. More specifically, the subsumption relation between input and output types are examined to generate flexible matches, which are beyond the abilities of existing text-based matching engines. The algorithm [67] performs two matches, one comparing outputs and another comparing inputs. If the output required by the requester is of a kind covered (subsumed) by the advertisement, then the inputs are checked. If the inputs specified in the request are subsumed by the input types acceptable to the service, then the service is a candidate to accomplish the requester's requirement. Numerous extensions have been proposed to improve the matchmaking algorithms by exploiting more features of subsumption relations [43]. Instead of looking at only the parameter types, these matching algorithms treat the whole OWL-S profile description as one OWL concept and go beyond equality and subsumption matching. Researchers have also pointed out the need to have a ranking mechanism to evaluate the matches produced. Benatallah et al. [10] describe one such ranking method based on the algorithms derived from hypergraph theory. Cardoso et al. [18] compute the syntactic, operational

and semantic similarities expressed in OWL-S profiles to generate a ranking function.

Using OWL-S for Web Service discovery has not been limited to matching algorithms. Several different researchers have also investigated how OWL-S can be integrated into existing Web Service discovery architectures. The OWL-S/UDDI matchmaker [87] integrates OWL-S capability matching into the UDDI registry. This integration is based on the mapping of OWL-S service profiles into UDDI Web Service representations [68]. Akkiraju et al. [1] improved this coupling by providing additional extensions to the UDDI inquiry API and enhancing the service discovery of UDDI by performing semantic matching and automatic service composition using planning algorithms. Other efforts show how to integrate OWL-S with different network protocols and P2P infrastructure. Specifically, Paolucci et al. [70] presented an approach for using OWL-S based discovery in Gnutella P2P environment, Elenius and Ingmarsson [26] achieved similar functionality in Juxtapose (JXTA) P2P infrastructure and Masuoka et al. [50] presented discovery methods in pervasive environments by utilizing the UPnP protocol.

7.3 Composition

The task of automated Web Service composition was introduced in Section 5.2. Automated composition is the process of automatically selecting, combining, integrating and executing Web Services to achieve a user's objective. "Make the travel arrangements for my WWW2005 conference trip" or "Buy me an Apple iPod at the best available price" are examples of possible user objectives addressed by composition. Industrial standards provide tools for specifying a manually generated composition so that it can be executed automatically; WS-BPEL (Web Service Business Process Execution Language) [4] is an example of such a tool. Human beings perform manual Web Service composition by exploiting their cultural knowledge of what a Web Service does (e.g., that `www.apple.com` will debit your credit card and send you an iPod), as well as information provided on the service's Web pages, in order to compose and execute a collection of services to achieve some objective. To automate Web Service composition, all this information must be encoded in a computer-interpretable form. None of the existing industrial standards for Web Service description encode this level of detail

One key advantage of automated Web Service composition is that it allows the flexibility to change the composition structure on the fly to adapt to changing conditions. For example, if a Web Service that is needed within a composition is no longer available, or it does not have the product that the client needs, the overall composition does not fail; rather it can be modified by introducing other Web Services that solve the problem at hand (e.g., by dynamically discovering services that are substitutable to the one that does not have the required product). As a result, OWL-S has the potential to support a form of dynamic Web Service.

Automated Web Service composition is akin to both an AI planning problem and a software synthesis problem, and draws heavily on both of these areas of research. There are several different approaches to Web Service composition based on OWL-S descriptions. For example, the OWL-S process model has been used to construct high-level generic procedures or plans that approximately describe how to achieve some objective. Then these high-level plans are expanded and refined using automated reasoning machinery. This is the task of nondeterminism reduction discussed previously. The first such system was the Golog based system [58, 59]. This system interleaves the execution of information-providing services with the simulation of world-altering ones to generate a sequence of Web Services customized to user's

preferences and constraints. In a similar spirit, several other researchers [35, 84] have used the paradigm of Hierarchical Task Network (HTN) planning [63] to perform automated Web Service composition. HTN planning has also been used in [69, 84] to compose Web Services in the travel domain and in the organization of a B2B supply chain. Other planning techniques that have been applied to the composition of OWL-S services, range from simple classical STRIPS-style planning [83], to extended estimated-regression planning [51], to “Planning as Model Checking” [85] that deals with non-determinism, partial observability, and complex goals. Some of these systems limit themselves to dealing only with atomic processes, since operator-based planning systems are unable to represent composite processes. In [60], the authors propose a way of compiling a large class of composite process specifications into atomic process specifications.

There are many systems that deal with a relaxed service composition problem where preconditions and results (effects) are ignored. Constantinescu et al. (e.g., [21]) presents a system that constructs chains of services based on partial matches of input/output types determined at runtime. These have great utility for information integration. Web Service Composer [85] uses an interactive composition method based on parameter types to put services together and filters the potential matches using information from the compositional context. An improved version of this approach is an end-user interactive composition system, STEER [49]. STEER lets fairly unsophisticated users find, select, compose and execute local, pervasive and remote Web Services to achieve common tasks. The system combines services that adhere to different ontologies by utilizing semantic mappings and/or transformation functions which are also published as Web Services.

Another OWL-S based approach for composition employed in the Semantic Discovery Service (SDS) [45] is to augment the standard WS-BPEL execution engine with an OWL-S ontology to enable run-time discovery and binding of services. The SDS system performs semantic integration and interoperation in the context of dynamic customization of existing Web Service compositions (WS-BPEL workflows). Customizing constraints provided by the user enable the dynamic binding of user-customized services in the workflow. However, this dynamic binding sometimes requires further semantic integration. OWL-S is used to describe the services, the customizing user constraints, and to perform Web Service discovery and semantic integration. Certainly we would like everyone to do composition natively in OWL-S, but we also concede that many will stick with IBM and Microsoft tools for a variety of reasons. By demonstrating that WS-BPEL can be augmented with OWL-S descriptions to perform dynamic service binding, we demonstrate the simple value-added of using an ontology such as OWL-S, either natively or to augment existing software.

The SDS system was more recently extended to add an explanation facility that enables rich explanations of the system’s success or failure at creating an integrated composition [55]. The explainable system integrates Inference Web [57] with the existing discovery system to add explanation, abstraction, and browsing capabilities along with a provenance registry infrastructure. Composition steps are registered using the Inference Web and both successful and failed composition efforts are explained, providing transparency and audit information. The addition of this explanation facility further underlines the value-added of semantic technologies for Web Service applications. It is also an example of where additional non-functional parameters are used in describing Web Services. The information is used to provide more meaningful explanations.

While all of this work is promising, we are still some distance from the goal of fully automated Web Service composition. With adoption of approaches to Web Service description such as OWL-S and advances in planning-related and program syn-

thesis technologies, we believe that broad-scale automated Web Service composition is well within reach.

8 Tools and Extensions

The release of OWL-S has stimulated a lot of interesting work in the Semantic Web Services area. Applications from industry have begun to emerge as the need for extra semantics has become increasingly evident. As a result, a wide variety of OWL-S applications and tools are now available. OWL-S has also been extended by researchers to support various application specific requirements. In this section, we summarize other applications, tools and extensions that build on or support OWL-S.

The OWL-S Editor [27], a plug-in to the popular ontology development tool, Protégé [34], provides a comprehensive set of capabilities for creating and maintaining OWL-S service descriptions. Among other things, the editor allows the user to build complex process models using a GUI. Another OWL-S editor [80] is provided by University of Malta and provides functionality to create, validate and visualize OWL-S services. DINST [33] is another graphical tool to visually build OWL-S services. The service creation process sets up a layered service ontology where OWL-S is used as an upper service ontology.

CMU OWL-S Development Environment (CODE) [88] is Eclipse-based and supports a Semantic Web Service developer through the whole process from the Java generation, to the compilation of OWL-S descriptions, to the deployment and registration with UDDI. CODE includes modules such as WSDL2OWL-S translator, an Eclipse-based editor for creating and editing OWL-S profile, process model and grounding, and the OWL-S Virtual Machine [65]. Mindswap OWL-S API [86] provides a Java API for programmatic access to read, execute and write OWL-S service descriptions. The API provides functionality for parsing, serializing, validating, reasoning, and executing services for various different versions of OWL-S.

There are also tools that primarily focus on the generation of OWL-S descriptions from WSDL files. Besides the WSDL2OWL-S [88] converter, another such tool is Assam (Automated Semantic Service Annotation with Machine learning) [41] which assists the user to semantically annotate (legacy) WSDL services. The tool uses a combination of different machine learning techniques such as iterative relational classification and ensemble learning to semi-automate the annotation process. OntoLink (Ontology-Linker) [49] is another tool that helps users to generate executable OWL-S descriptions from WSDL files. The emphasis of the tool is to semi-automate the generation of *grounding specifications* by letting the user define mappings between ontologies and XML schema definitions through a GUI. An XSLT transformation is automatically generated based on the mappings the user defined and the resulting OWL-S grounding is directly executable. OntoLink's other functionality is to specify semantic and syntactic mappings/transformations between concepts defined in different ontologies (in a similar fashion) so that services defined using disjoint ontologies can interoperate together. Mappings between ontologies are generated using XSLT and automatically wrapped inside a so-called *translator service*. These translator services transform the output of one service to a compatible format that another service can consume so that two previously incompatible services can be composed together.

In addition to developing tools for OWL-S, researchers have proposed several extensions to the OWL-S framework. For example, Denker et al. [24] present security annotations for OWL-S services to enable brokering between agents and services. A set of ontologies that describe credentials, security mechanisms and privacy options has been used for this purpose. Kagal et al. [31] extends this framework to express

such annotations in Rei [32], a logic-based language for defining rules and constraints over domain-specific ontologies. The OWL-S Matchmaker system [67] is also extended for policy matching and the OWL-S Virtual Machine (VM) [65] has been extended to enforce policies and security mechanisms. Similar extensions to OWL-S have been proposed in [87] to support policy and contract management using KAoS [92] services and tools. A policy management framework is used as an authorization service in grid computing environments, as a distributed policy specification and enforcement capability for a semantic matchmaker, and as a verification tool for service composition and contract management. Other kinds of extensions to OWL-S include augmenting the process ontology to model WS brokers [71], extensions to describe concrete application servers which facilitate reusing and combining Semantic Web software modules (e.g. ontology stores, reasoners, etc.), adaptations of OWL-S for describing Web Services in the bioinformatics domain [98], enriching OWL-S with speech-acts to describe agent based Web Services [39], and modeling processes for learning software to assist and train US Airforce Planners in a new large DARPA integrated learning effort [30].

The OWL-S Web site [64] includes pages listing related publications, tools, and use cases from the community at large. In addition, a number of open-source tools are hosted at www.semwebcentral.org. Discussion of issues related to OWL-S are conducted on the public mailing list of the W3C's Semantic Web Services Interest Group [81].

9 Related Work

In this section we review related work and compare OWL-S to alternative approaches to linking Web Services and the Semantic Web. The technologies discussed here are: WS-BPEL, CDL, ebXML, Grid services, SWSF, WSMO, and WSDL-S.

The Web Services Business Process Execution Language (WS-BPEL) [4] enables the specification of executable business processes (including Web Services) and business process protocols in terms of their execution logic or control flow. Executable business processes specify the behavior of individual participants within Web Service interactions and can be invoked directly, whereas business process protocols (also called abstract processes) abstract from internal behavior to describe the messages exchanged by the various Web Services within an interaction.

Abstract processes only consider protocol-relevant data and ignore process-internal data and computation. The effects of such computation on the business protocol are then described using non-deterministic data values. Executable processes, on the other hand, are described using a process description language which deals with both protocol-relevant and process-internal data. WS-BPEL also defines several mechanisms for recovery from faults, such as catching and handling of faults, and compensation handlers which specify compensatory activities in the event of actions that cannot be explicitly undone.

The area of greatest overlap between WS-BPEL and OWL-S is in the process model, but there are some crucial differences. OWL-S's process notation includes preconditions and results, which enable automated tools to select and compose Web Services. The WS-BPEL notation includes complex control structures and exception-recovery protocols. It would be fairly easy to add such features to OWL-S, but we are reluctant to do so without further research on how to automate reasoning about them.

The OWL-S process model and WS-BPEL attempt to cover similar territory, but in complementary ways. The emphasis in OWL-S is on making process descriptions computer-interpretable — described with sufficient information to enable *automation*

of a variety of tasks including Web Service discovery, invocation, and composition. WS-BPEL provides a language primarily intended for *manually* constructing processes and protocols. The two design goals are not really incompatible. For instance, recent work [45] has shown how WS-BPEL can use OWL-S descriptions of services to augment its functionality to include tasks such as dynamic partner binding and semantic integration.

ebXML (Electronic Business using eXtensible Markup Language) [25] addresses the broad problem of B2B interaction from a workflow perspective. Business interactions are described through two views: a Business Operational View (BOV) and a Functional Service View (FSV). The BOV deals with the semantics of business data transactions, which include operational conventions, agreements, mutual obligations and the like between businesses. The FSV deals with the supporting services: their capabilities, interfaces and protocols. Although ebXML does not restrict the means of B2B interaction to Web Services, their focus is essentially the same as that of OWL-S.

ebXML enables Web Services to describe the business processes they support and the services they offer using Collaboration Protocol Profiles (CPP). A business process in ebXML is considered to be a set of business document exchanges between a set of Web Services. This is akin to the Web Services message exchange as commonly described in Web Services standards. CPPs contain industry classification, contact information, supported business processes, interface requirements etc. They are registered within an ebXML registry (similar to a UDDI registry), in which other Web Services and their business processes can be discovered. However, ebXML's scope does not extend to the format in which the business documents are specified. This is left to the interacting Web Services to agree upon a priori by the creation of a Collaboration Protocol Agreement. Since OWL-S provides a language for the description of the behavior of Web Services, its scope is complementary to that of ebXML. In fact, OWL-S descriptions could themselves be used within ebXML to describe the business processes of interacting Web Services.

The **Web Service Choreography Description Language (CDL)** describes Web Service interactions in terms of their externally observable behavior, typically exposed through message exchanges. Each participant in an interaction specifies an interface, describing the temporal ordering and logical dependence of messages it sends and receives. In addition, a global model can be specified that describes the collective message exchange of interacting Web Services. Unlike WS-BPEL, CDL does not describe the executable details of individual Web Services. It focuses on the problem of collaboration (message exchange between distributed peers) rather than orchestration (creation of executable Web Services). Consequently, it does not assume the presence of a central process managing the interactions between Web Services. For the same reason, CDL control flow constructs are considerably simpler than those of WS-BPEL. CDL documents are formal specifications intended for explication, (automated) verification, and conformance testing, although they can be used to automatically generate code skeletons.

CDL corresponds most closely to the OWL-S process model. Both share the goals of describing the message exchange between participating Web Services. However, unlike CDL, OWL-S markup is also meant to support execution and the generation of executable service compositions (e.g., using planning techniques). Also, CDL specifically eschews any description of the *significance*, e.g., the business significance, of interactions whereas OWL-S is specifically intended to support the description of everything from the “real world” preconditions and results of an invocation to the classification of services by their primary purpose.

Grid services. A Grid is a system designed “to coordinate resources that are not subject to centralized control using standard, open, general purpose protocols and

interfaces to deliver nontrivial quality of service” [28]. The resources available on a Grid are modeled as *Grid services* with state. The Open Grid Services Infrastructure (OGSI) uses extended WSDL constructs and XML schemas to introduce the notion of stateful Web Services along with standard operations for creating and destroying Web Services, and for representing, querying, and updating metadata associated with a Web Service. In addition, OGSI provides mechanisms for references to instances of Web Services and asynchronous notification of Web Service state changes. OGSI also defines an XML Schema for describing faults that take place during WSDL operations.

The OGSI framework has been recently refactored to define a family of related specifications that could be adopted on a piecemeal basis. This refactoring together with some extensions to keep pace with changes in Web Services standards forms the WS-Resource Framework (WSRF). Although OGSI and WSRF go beyond WSDL with the specification of several kinds of specialized port types and fault messages, they do not attempt to describe the behavior of Grid or Web Services as OWL-S does. In this sense, OWL-S is complementary to both of these specifications.

The **Semantic Web Services Framework (SWSF)** [82], published online recently by the Semantic Web Services Initiative, builds loosely on OWL-S to provide a more comprehensive framework, in the sense of defining a larger set of concepts. It also builds on a mature pre-existing ontology of process modeling concepts, the Process Specification Language (PSL). SWSF specifies a Web-oriented language, SWSL, with a logic programming layer and also a first-order logic layer. It uses SWSL to define an ontology of service concepts (SWSO), and takes advantage of SWSL’s greater expressiveness (relative to OWL) to more completely axiomatize the concepts. Since SWSO is completely axiomatized in first-order logic, it avoids the need to use hybrid (DL-based and FOL-based) reasoning as is the case with OWL-S.

The **Web Services Modeling Ontology (WSMO)** [96] shares with OWL-S and SWSF the vision that ontologies are essential to support automatic discovery, interoperation, composition, etc. of Web Services. Like SWSF, the WSMO effort defines an expressive Web-oriented language, WSML [23], which provides a uniform syntax for subdialects ranging from description logic to first-order logic. Like OWL-S, WSMO declares inputs, outputs, preconditions, and results (although using somewhat different terminology) associated with services. Unlike OWL-S, WSMO does not provide a notation for building up composite processes in terms of control flow and data flow. Instead, it focuses on specification of internal and external choreography, using an approach based on abstract state machines. Other distinguishing characteristics include WSMO’s emphasis on the production of a reference implementation of an execution environment, WSMX, and on the specification of mediators --- mapping programs that solve the interoperation problems between Web Services.

In WSMO’s approach, mediators perform tasks such as translation between ontologies, or between the messages that one Web Service produces and those that another Web Service expects. WSMO includes a taxonomy of possible mediators that helps to classify the different tasks that mediators are supposed to solve. The definition of mediators in WSMO calls attention to some important translation tasks associated with Web Services. Not surprisingly, these same translation tasks are needed in support of interactions with OWL-S described Web Services. Some OWL-S based systems also make use of mediator components. However, rather than requiring the existence of a distinguished type of entity in the Web Services infrastructure, OWL-S takes the view that mediators are services (a view that recently has been espoused by many WSMO researchers) and as such these mediation services can use the mechanisms provided by OWL-S for discovery, invocation and composition. In addition, mediators could be constructed through OWL-S –enabled composition [69, 72].

WSDL-S [2] is a small set of proposed extensions to WSDL by which semantic annotations may be associated with WSDL elements such as operations, input and output type schemas, and interfaces. WSDL-S aims to support the use of “semantic concepts analogous to those in OWL-S while being agnostic to the semantic representation language” [2]. The way in which WSDL-S allows one to specify a correspondence between WSDL elements and semantic concepts is very similar to how the OWL-S grounding works; indeed, something very much like OWL-S declarations could be used as the referents of the WSDL-S attributes. The most notable difference between WSDL-S and the OWL-S grounding is that with WSDL-S the correspondence must be given in the WSDL spec, whereas with OWL-S it is given in a separate OWL document. (The OWL-S grounding also proposes some WSDL extension elements, but they are regarded as optional for most purposes.) Thus, the aims of WSDL-S are compatible with those of OWL-S, but WSDL-S focuses on the practical advantages of a more lightweight, incremental approach. (We note that WSDL-S is associated with an execution environment and tools known as METEOR-S, but comparison with tools and environments is beyond the scope of this paper.)

10 Conclusions and Future Directions

OWL-S consists of three parts:

- A vocabulary for describing Web Service *profiles*, which are high-level descriptions of Web Services suitable for advertising them and supporting reasoning about their applicability to a user’s aims.
- An ontology and presentation language for specifying *process models* in detail. The models describe services primarily in terms of IOPRs -- inputs, outputs, preconditions, and results. The preconditions and effects (conditioned by result conditions) are fine-grained and precise enough to be matched to the user’s formally specified goals and descriptions of available resources. *Composite* processes add information about process structure, behavior, and interactions.
- An ontology for describing *groundings* of Web Services, which map their inputs and outputs to messages at the communication-protocol level. Although other protocols have been used, SOAP messaging, as described by WSDL, has been the most popular choice for use with OWL-S, and the ontology includes built-in vocabulary for describing mappings to WSDL ports and messages.

Further information on OWL-S is available at the Web site [64]. In addition to the OWL ontology files, the release site includes examples and other forms of documentation, including a comprehensive technical overview, a tutorial “walk-through” of a simple example, additional explanatory material regarding the grounding and the use of profile-based class hierarchies, and information about the status of this work. A subset of this material has been presented to the W3C consortium as a member submission [48].

We measure the success of OWL-S by its ability to support research into automated reasoning about Web Services and how they can achieve users’ goals. We are encouraged by the number of projects, tools, and extensions that are building on OWL-S (including a great many efforts that have been carried out by non-members of the OWL-S Coalition).

There is, of course, much that remains to be done. One obvious gap is the lack of exception-handling machinery in the process model. Another is the absence of ways

of describing security and quality-of-service specifications. (Promising work on these things is underway, but has not yet been incorporated into OWL-S releases.)

For OWL-S to become widely used it must become more tightly integrated with commercial Web Service standards and the tools supporting it must be made easier to use for non-expert developers. Further work on algorithms that effectively support discovery, selection, and composition of services in large-scale, complex settings is also needed. In addition, it will be important to conduct empirical evaluation of the applicability and benefits of OWL-S in developing and managing service-based systems in businesses and other settings.

As in other areas of the Semantic Web, we need more domain level ontologies. There is a need to develop specializations of the profile for a variety of domains, and for broad categories of services. In addition, for services that sell goods, ontology modules are needed for the specification of cost models, negotiations, contracts, and guarantees. To encourage the participation of communities of interest in developing these extensions, we plan to investigate the use of an open-source style of evolution.

OWL-S provides a representational framework that provides for fuller specification of the capabilities and behavior of Web Services, so as to support greater automation of service-related activities such as discovery and composition. We believe that the need to better support such activities in the world of Web Services will lead to significant further evolution of OWL-S and related approaches.

Acknowledgments

Numerous people have contributed to OWL-S, many of whom are not included in the authorship of this paper. In particular, we wish to acknowledge the contributions of A. Ankolekar, G. Denker, D. Elenius, J. Hobbs, L. Kagal, O. Lassila, S. Narayanan, B. Parsia, T. Payne, M. Sabou, C. Schlenoff, M. Solanki, R. Washington and H. Zeng. We also wish to thank I. Horrocks, P. Patel-Schneider and M. Uschold for helpful comments at various stages of the development of this ontology. We are indebted to the W3C for providing the use of the public-sws-ig mailing list for discussion of OWL-S issues, and to the many interested researchers and users who have taken the trouble to share their suggestions and questions on the list. We gratefully acknowledge funding from the DARPA DAML program, without which OWL-S would not have been possible, and special thanks go to J. Hendler, M. Burke and M. Greaves (DAML program managers) for their tremendous support of this effort.

References

1. R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder, "A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI", in *Proceedings of IJCAI Information Integration on the Web Workshop*, Acapulco, Mexico, August 2003.
2. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma, "Web Service Semantics - WSDL-S", Technical Note, Version 1.0, April, 2005. Available at <http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html>.
3. J. L. Ambite (Ed.), *Proceedings of the ICAPS2003 Workshop on Planning for Web Services*, 2003.
4. T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte (Ed.), I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services", Version 1.1, 2003, at <http://www.ibm.com/developerworks/webservices/library/ws-bpel/>.
5. A. Ankolekar, F. Huch and K. Sycara, "Concurrent Execution Semantics of DAML-S with Subtypes", in *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, 2002.

6. A. Ankolekar, M. Paolucci, and K. Sycara, "Towards a Formal Verification of OWL-S Process Models", in *Proceedings of the International Semantic Web Conference (ISWC 2005)*, 2005.
7. Apache Axis, at <http://ws.apache.org/axis/>.
8. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider *The Description Logic Handbook; Theory, Implementation, and Applications*. Cambridge University Press, 2003.
9. W.-T. Balke and M. Wagner, "Towards personalized selection of Web Services", in *Proceedings of the International World Wide Web Conference*, Budapest, Hungary, 2003.
10. B. Benatallah, M. Hacid, C. Rey and F. Toumani, "Request Rewriting-Based Web Service Discovery", in *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp 335-350, October 2003.
11. T. Berners-Lee, "Primer: Getting into RDF and the Semantic Web using N3", at <http://www.w3.org/2000/10/swap/Primer.html>, 2000.
12. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", *Scientific American*, May 2001.
13. A. Bernstein and M. Klein, "High Precision Service Retrieval", in *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, Sardegna, 2002.
14. P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso, "Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking", in *Proceedings of the 17th Int. Joint Conference on Artificial Intelligence Conference (IJCAI'01)*, 2001.
15. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, H. Stuckenschmidt: "C-OWL: Contextualizing Ontologies", *Proceedings of the International Semantic Web Conference*, pp. 164-179, October 2003.
16. D. Box, D. Ehnebuske, . Kakivaya, A. Layman, N; Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, *Simple Object Access Protocol (SOAP) 1.1*. W3C Tech Report, 2000, at <http://www.w3.org/TR/SOAP>.
17. M. Burstein, C. Bussler, T. Finin, M. Huhns, M. Paolucci, A. Sheth, S. Williams and M. Zaremba , "A Semantic Web Services Architecture" *IEEE Internet Computing*, September-October 2005.
18. J. Cardoso and A. Sheth, "Semantic E-workflow Composition", *Journal of Intelligent Information Systems*, Vol. 21, No. 3, pp. 191-225, 2003.
19. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", 2001, at <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>.
20. S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello, "Logic Based Approach to Web Services Discovery and Matchmaking", in *Proceedings of the E-services Workshop at 5th International Conference on Electronic Commerce (ICEC'03)*, Pittsburgh, USA, 2003.
21. I. Constantinescu, B. Faltings, and W. Binder, "Large Scale, Type-Compatible Service Composition", in *Proceedings of the 2nd International Conference on Web Services (ICWS 2004)*, San Diego, USA, July 2004.
22. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P.F. Patel-Schneider, and L. A. Stein, "Web Ontology Language (OWL) W3C Reference version 1.0", 18 August 2003, at <http://www.w3.org/TR/2002/WD-owl-ref-20021112>.
23. J. De Bruijn, H. Lausen, A. Polleres, and D. Fensel, "The Web Service Modeling Language WSM: An Overview," *DERI Technical Report 2005-06-16*, 2005, at <http://www.wsmo.org/wsml/wsml-resources/deri-tr-2005-06-16.pdf>
24. G. Denker, L. Kagal, T. Finin, M. Paolucci, N. Srinivasan and K. Sycara, "Security For DAML Web Services: Annotation and Matchmaking", in *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp. 335-350, October 2003.
25. Electronic Business using eXtensible Markup Language (eXML) Web site, at <http://www.ebxml.org/>.
26. D. Elenius, M. Ingmarsson, "Ontology-based Service Discovery in P2P Networks", in *Proceedings of the MobiQuitous'04 Workshop on Peer-to-Peer Knowledge Management (P2PKM 2004)*, Boston, USA, August 2004.
27. D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, R. Senanayake, "The OWL-S Editor -- A Development Tool for Semantic Web Services", in *Proceedings of the 2nd European Semantic Web Conference*, May 2005.

28. I. Foster, "What is the Grid? A Three Point Checklist", July 20, 2002, at <http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf>
29. E. Friedman-Hill, "Jess, the Rule Engine for the Java Platform", at <http://herzberg.ca.sandia.gov/jess/>.
30. GILA project. DARPA Integrated Learning Program. Available at <http://www.atl.external.lmco.com/news/2006/050906.php>.
31. L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara, "Authorization and Privacy for Semantic Web Services", *IEEE Intelligent Systems*, Vol. 19, No. 4, pp. 50-56, July 2004.
32. L. Kagal, T. Finin, and A. Joshi, "A Policy Based Approach to Security on the Semantic Web", in *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, FL, October 2003.
33. M. Klein and B. König-Ries, "A Process and a Tool for Creating Service Descriptions based on DAML-S", in *Proceedings of 4th VLDB Workshop on Technologies for E-Services (TES'03)*, Berlin, Germany, September 2003.
34. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", in *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan (2004).
35. U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler, "Information Gathering During Planning for Web Service Composition", in *Proceedings of the Third International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
36. T. Di Noia, E. Di Sciacio, F. M. Donini and M. Mongiello, "Semantic Matchmaking in a P-2-P Electronic Marketplace", SAC 2003, pp. 582-586, 2003.
37. R. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence* 2, pp. 189-208, 1971.
38. F. Gandon and N. Sadeh, "Semantic Web Technologies to Reconcile Privacy and Context Awareness", *Web Semantics Journal*, Vol. 1, No. 3, 2004.
39. N. Gibbins, S. Harris, and N. Shadbolt, "Agent-based Semantic Web Services", in *Proceedings of the 12th International WWW Conference (WWW2003)*, 2003.
40. M. Gruninger, "Applications of PSL to Semantic Web Services", in *Proceedings of the Workshop on Semantic Web and Databases. Very Large Databases Conference*, Berlin 2003.
41. A. Heß, E. Johnston and N. Kushmerick, "ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services", in *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
42. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", W3C Member Submission, May 2004, at <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.
43. L. Li and I. Horrocks, "A Software Framework for Matchmaking Based on Semantic Web Technology", in *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331-339, ACM, 2003.
44. T. W. Malone, K. Crowston, B. P. Jintae Lee, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell, "Tools for Inventing Organizations: Toward a Handbook of Organizational Processes", *Management Science*, Vol. 45, No. 3, pp. 425-443, March, 1997.
45. D. Mandell and S. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation", in *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, pp. 227--241, 2003. Code for the working system is available at: <http://projects.semwebcentral.org/>
46. D. Martin, A. Cheyer, and D. Moran, "The Open Agent Architecture: A Framework for Building Distributed Software Systems," *Applied Artificial Intelligence*, Vol. 13, Nos. 1 and 2, 1999, pp. 91-128.
47. D. Martin, M. Burstein, O. Lassila, M. Paolucci, T. Payne, S. McIlraith, "Describing Web Services using OWL-S and WSDL", in Release 1.2 of OWL-S, at <http://www.daml.org/services/owl-s/1.2/owl-s-wsdl.html>
48. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, "OWL-S: Semantic

- Markup for Web Services”. W3C Member Submission, November 2004, at <http://www.w3.org/Submission/2004/07/>.
49. R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin, “Ontology-Enabled Pervasive Computing Applications”, in *IEEE Intelligent Systems*, Vol. 18, No. 10, pp. 68-72, 2003.
 50. R. Masuoka, B. Parsia and Y. Labrou, “Task Computing - The Semantic Web meets Pervasive Computing”, in *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, Sanibel Island, FL, USA, October 2003.
 51. D. McDermott, “Estimated-Regression Planning for Interactions with Web Services”, in *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS2002)*, Toulouse, France, April 2002.
 52. D. McDermott, “The Planning Domain Definition Language Manual”, *Yale Computer Science Report 1165 (CVC Report 980003)*, 1998.
 53. D. McDermott, “Presentation Syntax for OWL-S,” in Release 1.2 of OWL-S, at <http://www.daml.org/services/owl-s/1.2/owl-s-gram/owl-s-gram-htm.html>.
 54. D. McDermott and D. Dou, “Representing Disjunction and Quantifiers in RDF”, *Proceedings of the First International Semantic Web Conference (ISWC2002)*, 2002.
 55. D. L. McGuinness, D. Mandell, S. McIlraith, P. Pinheiro da Silva, “Explainable Semantic Discovery Services”, *Stanford Networking Research Center Project Review*, Stanford, CA, February 2005.
 56. D. L. McGuinness and F. van Harmelen, “OWL Web Ontology Language Overview”, World Wide Web Consortium (W3C) Recommendation. February 10, 2004, at <http://www.w3.org/TR/owl-features/>
 57. D. L. McGuinness and P. P. da Silva, “Explaining Answers from the Semantic Web: The Inference Web Approach”, *Journal of Web Semantics*, Vol. 1, No. 4, pp. 397-413, October 2004.
 58. S. McIlraith and T. Son, “Adapting Golog for Composition of Semantic Web Services”, in *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pp. 482-493, 2002.
 59. S. McIlraith., T.C. Son and H. Zeng, “Semantic Web Services”, *IEEE Intelligent Systems, Special Issue on the Semantic Web*, Vol. 16, No. 2, pp. 46-53, March/April, 2001.
 60. S. McIlraith and R. Fadel, “Planning with Complex Actions”, in *Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning (NMR2002)*, pages 356-364, April, 2002.
 61. S. Narayanan and S. McIlraith, “Simulation, Verification and Automated Composition of Web Services”, in *Proceedings of the Eleventh International World Wide Web Conference (WWW2002)*, Honolulu, Hawaii, May 2002
 62. D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, “SHOP: Simple Hierarchical Ordered Planner”, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp.968—973, 1999.
 63. D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, “SHOP2: An HTN Planning System”, *Journal of Artificial Intelligence Research*, 2003.
 64. D. Martin, M. Burstein, D. McDermott, D. McGuinness, S. McIlraith, M. Paolucci, E. Sirin, N. Srinivasan, K. Sycara, “OWL-S 1.2 Release”, At <http://www.daml.org/services/owl-s/1.2/>.
 65. M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara, “The DAML-S Virtual Machine”, in *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp 335-350, October 2003.
 66. M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura, “Toward a Semantic Choreography of Web Services: from WSDL to DAML-S”, in *Proceedings of ICWS03*, 2003.
 67. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, “Semantic Matching of Web Services Capabilities”, in *Proceedings of the First International Semantic Web Conference (ISWC2002)*, 2002.
 68. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, “Importing the Semantic Web in UDDI”, in *Proceedings of E-Services and the Semantic Web (ESSW02)*, 2002.
 69. M. Paolucci, K. Sycara, and T. Kawamura, “Delivering Semantic Web Services”, in *Proceedings of the Twelfth World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003, pp 111- 118.

70. M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan, "Using DAML-S for P2P Discovery", in *Proceedings of the First International Conference on Web Services (ICWS'03)*, Las Vegas, USA, June 2003, pp 203- 207.
71. M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara, "Untangling the Broker Paradox in OWL-S", in *Proceedings of AAAI Spring Symposium on Semantic Web Services*, 2004.
72. M. Paolucci, N. Srinivasan and K. Sycara, "Expressing WSMO Mediators in OWL-S", in *Proceedings of the Semantic Web Services Workshop (SWS2004)* at the Third International Semantic Web Conference (ISWC 2004). CEUR Workshop Proceedings, Vol. 119, paper 10. Online proceedings available at <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-119/>.
73. P. F. Patel-Schneider, "A Proposal for a SWRL Extension Towards First-Order Logic", W3C Member Submission, April 2005, at <http://www.w3.org/Submission/2005/SUBM-SWRL-FOL-20050411/>.
74. Paulo Pinheiro da Silva, Deborah L. McGuinness and Richard Fikes. A Proof Markup Language for Semantic Web Services. *Information Systems*, Vol. 31, Nos. 4-5, June-July 2006, Pages 381-395.
75. E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF", at <http://www.w3.org/TR/rdf-sparql-query/2005/>.
76. J. Rao, P. Kungas, and M. Matskin, "Composition of Semantic Web Services using Linear Logic theorem proving", *Information Systems Journal - Special Issue on the Semantic Web and Web Services*, 2004.
77. R. Reiter *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, Mass.: The MIT Press, 2001.
78. D. Roman, H. Lausen, U. Keller, "Web Service Modeling Ontology", at <http://www.wsmo.org/TR/d2/v1.1/#mediators>.
79. The Rule Markup Initiative, at <http://www.dfki.uni-kl.de/ruleml/>.
80. J. Scicluna, C. Abela, and M. Montebello, "Visual Modelling of OWL-S Services", in *Proceedings of the IADIS International Conference WWW/Internet (ICWI2004)*, Madrid, Spain, October 2004.
81. Semantic Web Services Interest Group (a W3C activity) home page, at <http://www.w3.org/2002/ws/swsig/>.
82. Semantic Web Services Framework, version 1.0, at <http://www.daml.org/services/swsf/1.0/>.
83. M. Sheshagiri, M. desJardins, and T. Finin, "A Planner for Composing Services Described in DAML-S", in the *Proceedings of AAMAS'03 Workshop on Web Services and Agent-based Engineering*, 2003.
84. E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN Planning for Web Service Composition using SHOP2", *Journal of Web Semantics*, Vol 1, Nol. 4, pp. 377-396, 2004.
85. E. Sirin, B. Parsia, and J. Hendler, "Filtering and Selecting Semantic Web Services with Interactive Composition Techniques", *IEEE Intelligent Systems*, Vol. 19, No. 4, pp. 42-49, 2004.
86. E. Sirin and B. Parsia, "The OWL-S Java API", Poster, in *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
87. N. Srinivasan, M. Paolucci, K. Sycara, "An Efficient Algorithm for OWL-S Based Semantic Search in UDDI", *SWSWPC 2004*: 96-110.
88. N. Srinivasan, M. Paolucci, and K. Sycara, "CODE: A Development Environment for OWL-S Web Services", at <http://projects.semwebcentral.org/projects/owl-s-ide/>
89. Sycara, K., Paolucci, M. Ankolekar, A., Srinivasan, N. "Automated Discovery, Interaction and Composition of Semantic Web Services", *Journal of Web Semantics*, Vol. 1, No. 1, December 2003, pp. 27-46.
90. P. Traverso and M. Pistore, "Automated Composition of Semantic Web Services into Executable Processes", in *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, Nov. 2004.
91. "The Universal Description, Discovery and Integration (UDDI) Protocol", Version 3, 2003, at <http://www.uddi.org/>
92. A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, S. Aitken, "KAoS Policy Management for Semantic Web Services", *IEEE Intelligent Systems*, Vol. 19, No. 4, pp. 32-41, July 2004.

93. Web Services Choreography Working Group home page, at <http://www.w3.org/2002/ws/chor/>.
94. Web Services Description Working Group home page, at <http://www.w3.org/2002/ws/desc/>.
95. “Web Services Architecture” (Working Group Note), February, 2004, at <http://www.w3.org/TR/ws-arch/>.
96. Web Service Modeling Ontology (WSMO) Web site, at <http://www.wsmo.org/>.
97. H. Wong and K. Sycara, “A Taxonomy of Middle-Agents for the Internet,” in *Proceedings of the 5th International Conference on Multi-Agent Systems (ICMAS 2000)*, AAAI Press, 2000, pp. 465–466.
98. C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood, “A Suite of DAML+OIL ontologies to describe Bioinformatics Web Services and Data”, in *International Journal of Cooperative Information Systems*, Vol. 12, No. 2, pp. 197–224, 2003.
99. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau, “Automating DAML-S Web Services Composition Using SHOP2”, in *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, 2003.