World Wide Web (2007) 10:243–277
DOI 10.1007/s11280-007-0033-x

Bringing Semantics to Web Services with OWL-S

David Martin · Mark Burstein · Drew McDermott · Sheila McIlraith · Massimo Paolucci · Katia Sycara · Deborah L. McGuinness · Evren Sirin · Naveen Srinivasan

Received: 17 July 2005 / Revised: 2 March 2006 /
Accepted: 30 March 2007 / Published online: 3 July 2007

Abstract Current industry standards for describing Web Services focus 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 software programs that can interpret descriptions

This work was performed while Paolucci was at the Robotics Institute, Carnegie Mellon University, Sirin was at the University of Maryland, College Park, and Srinivasan was at the Robotics Institute, Carnegie Mellon University.

D. Martin (✉)
Artificial Intelligence Center, SRI International, Menlo Park, CA, USA
e-mail: martin@ai.sri.com

M. Burstein
Intelligent Distributed Computing Department, BBN Technologies, Cambridge, MA, USA

D. McDermott
Computer Science Department, Yale University, New Haven, CT, USA

S. McIlraith
Department of Computer Science, University of Toronto, Toronto, Canada

M. Paolucci
DoCoMo Communications Laboratories Europe, Munich, Germany

K. Sycara
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

D. L. McGuinness
Knowledge Systems, Artificial Intelligence Laboratory, Stanford University, Stanford, CA, USA

E. Sirin
Clark+Parsia, LLC, Arlington, VA, USA

N. Srinivasan
webMethods, Inc., Fairfax, VA, USA

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 on OWL-S has led to the creation of many 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

1 Introduction

An important trend in the evolution of the World Wide Web (WWW) has been the development of standardized Web Service technologies, by which a service provider can make a service available to consumers through an advertised protocol. The function of a Web Service is often to supply information, but can also involve exchange or sale of goods or obligations. Web Services provide a basis for interoperability between service providers and consumers, based on the reliable exchange of messages.

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 a series of messages, normally encoded in XML (extensible markup language), which contain(s) the information or request to be sent to the service; the program then receives XML replies containing the returned values. The datatypes for the information passed between the service consumer and provider, and its encoding using the XML-based simple object access protocol (SOAP) [12], can be described using the Web Services description language (WSDL) [15]. Existing compilers for languages like Java [33] and C# [42] can automatically produce code to convert from the datatypes of a language to their SOAP equivalents, making it almost as easy to call a Web Service as to use a standard library component.

One thing WSDL does not supply, however, is a specification of *what happens* when a Web Service is used. A human programmer is supposed to figure that out from reading a natural-language description. Suppose you want an automated software agent [60] to solve problems such as the following:

- Given Web Service interfaces to access the current prices of two parts suppliers, make a periodic decision as to which of those suppliers to acquire 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.
- Given Web Service interfaces to a variety of networked capabilities that are available in a meeting room, send a copy of your slides to everyone attending an invited presentation that you are giving in that room.
- Given Web Service interfaces for the catalogs of various online vendors, find products described using one or more controlled vocabularies (which might not be the same as the vocabularies used in the vendors' catalogs).

Without a description of what happens when a Web Service is used—a description that can be processed by computer software—a human must always be involved in solving such

problems. Thus, the question arises of how to provide software agents with the information they need about Web Services to solve problems such as these.

A second trend in the evolution of the Web, known as the *Semantic Web*, can provide some answers. The Semantic Web is a set of technologies for representing, and publishing on the Web, computer-interpretable structured information [10]. *Semantic Web Services* [56] is a research field that endeavors to apply these technologies to the description and use of Web Services. Whereas *interoperability* is the primary motivation for Web Services, *automation* of information use and *dynamic interoperability* are the primary objectives of 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 definitions encoded there allow programs to make inferences about the relationships among the objects that 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 the resource description framework (RDF), RDF schemas (RDFS), and the Web ontology language (OWL) [52], all of which have completed the standardization process at the World Wide Web Consortium (W3C).

The central question facing Semantic Web Services, then, is: "Can we use Semantic Web techniques to automate dealings with Web Services?" The goal of the OWL-S Coalition [61] is to show that the answer is *yes*. To move towards the realization of this vision, members of this coalition and other Semantic Web Service researchers have been developing languages, ontologies, algorithms, and architectures. In this paper, we describe OWL-S version 1.2 [46], an ontology for services expressed in OWL.[1] It 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 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 are still the subject of ongoing work). In each case, one would likely have a form-based interface between the human user and the automated software agent doing the required reasoning.

- To support a regular choice between suppliers, a purchasing software agent could first use automated planning technology [58, 80] to access a registry and find vendors potentially satisfying the user's objectives, then use OWL-S descriptions of those vendors' 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) [48].
- 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, as 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

[1] Prior to version 1.0, this work was known as DAML-S, so named for DAML+OIL, the precursor of OWL.

statements involving the *equivalentClass* construct of OWL would enable the agent to translate between the two.

This paper describes OWL-S and selected tools, technologies and research directions based upon it. 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 *service profiles, process models*, and *groundings*. In Section 3, we discuss key concepts for describing services: their inputs, outputs, preconditions, and results. 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. The latter 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 and prototypes 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 directions for future work in Section 10.

2 Overview of OWL-S

In this section, we provide some context for understanding OWL-S. After presenting the objectives that motivate this work, we give a brief overview of its logic foundations, ways in which it can be used, and its overall structure.

The principal high-level objectives of OWL-S are (1) to provide a general-purpose representational framework in which to describe Web Services; (2) to support automation of service management and use by software agents; (3) to build, in an integral fashion, on existing Web Service standards and existing Semantic Web standards; and (4) to be comprehensive enough to support the entire lifecycle of service tasks.

2.1 Logic foundations

OWL-S starts off as an *ontology*, i.e. a vocabulary plus axioms constraining its terms [51]. The axioms are stated in OWL, so they involve things like class descriptions, subclass hierarchies, and definitions of the kinds of relationships that may hold between different classes. OWL includes three sublanguages: OWL-lite, OWL-DL, and OWL full. The first two, but not the third, correspond to decidable description logics (DLs) [5]. *Decidability*, in this context, means that certain fundamental questions about an ontology are guaranteed to be answerable. An example of this is the question of *subsumption* [5], i.e. whether one particular class A is a superclass of another class B (in other words, whether A *subsumes* B). To preserve decidability, we had a design objective to keep OWL-S expressible in OWL-DL.

Nevertheless, we have discovered that for some purposes OWL-DL is not expressive enough. For example, to specify the preconditions and results of processes, we need expressiveness comparable to that of first-order logic [26]. Thus, while many aspects of processes *can* be represented in OWL-DL (and are in OWL-S), we have found it necessary to allow for preconditions and results to be expressed in other logical languages.

In addition, we have found that the language elements and syntax provided by OWL-DL are not particularly well suited to provide process descriptions that can be easily comprehended by humans. For this reason, we have provided a "presentation syntax" for

developers to use when writing their process models, which can be expanded into OWL and associated rule representations for publication on the Semantic Web.

2.2 Applicability

As an ontology, OWL-S does not make any assumption with respect to the processing environment or Web Service 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 [11] (among a service registry, producer, and consumer), to peer-to-peer (P2P) systems [23] using a Gnutella-based architecture [67], to multicast-based universal plug and play (UPnP) [48] systems used with wireless devices, to architectures that exploit a centralized mediator [44, 87] to manage the interaction with services and perform different forms of translation between service consumers and providers [68, 69].

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 is operational and understandable to computer programs. In our opinion, the Web Services community needs a little push to get to the point where it feels comfortable with descriptions of entities outside current Web Service specifications. OWL-S provides that push, right in the direction of the space that the Semantic Web community already inhabits.

2.3 Overall structure of OWL-S

The OWL-S ontology includes three primary sub-ontologies: the service profile, process model, and grounding. The service profile is used to describe *what the service 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 service 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. In this figure, an oval represents an OWL class, and an arc represents an OWL property. For example, the *presents* property represents a relationship that can hold between a *service* and a *profile*.

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 (an instance of the class

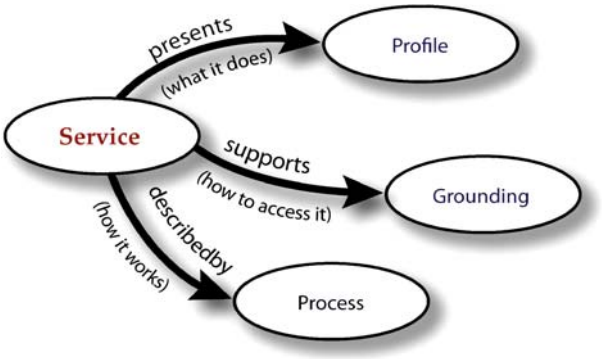

Figure 1 Top level of the service ontology.

process), one or more groundings (each an instance of the class *grounding*), and optionally one or more profiles (each an instance of the class *profile*).[2] 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 service profile may be thought of as a summary of salient aspects of the process model plus additional information that is suitable for the 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 OWL-S technical overview [46].

3 Inputs, outputs, preconditions, and results

Understanding all three components of an OWL-S service model requires understanding inputs, outputs, preconditions, and results. The *inputs* are the object descriptions that 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 is 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. *Results* consist of effect and output specifications. An *effect* is a proposition that will become true when the service completes. In general, these propositions are not 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))
```

(AcctID and Valid are local variables. This concept will be explained in Section 5.2). The precondition requires the user to actually have an account with the service, with the account identifier AcctID. It also requires the credit card to have a known status Valid.

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.[3] 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)
```

[2] For ease of exposition, Figure 1 presents a slight simplification. In particular, it omits an organizational layer of classes named ServiceProfile, ServiceGrounding, and ServiceModel.

[3] 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.

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 cannot 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 provider, not the service consumer, although if the consumer knew the credit-card status, it could use that knowledge to predict what the provider 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 implemented software 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. Although this is not entirely obvious in the example above, which refers to statuses and balances, it will be discussed further in Section 5.

The last point deserves emphasis. The purpose of OWL-S is to allow software agents to use Web Services to accomplish real-world goals. The idea is to have the ability to ask a software 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.

Inputs, outputs, preconditions, and results are referred to, in different ways, by each of the three components of an OWL-S service model: the service profile, the process model, and the grounding. We will discuss them in more detail in that order.

4 The service profile subontology

The OWL-S profile provides a set of concepts to specify capabilities of services [11], 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 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 defined in a profile are the

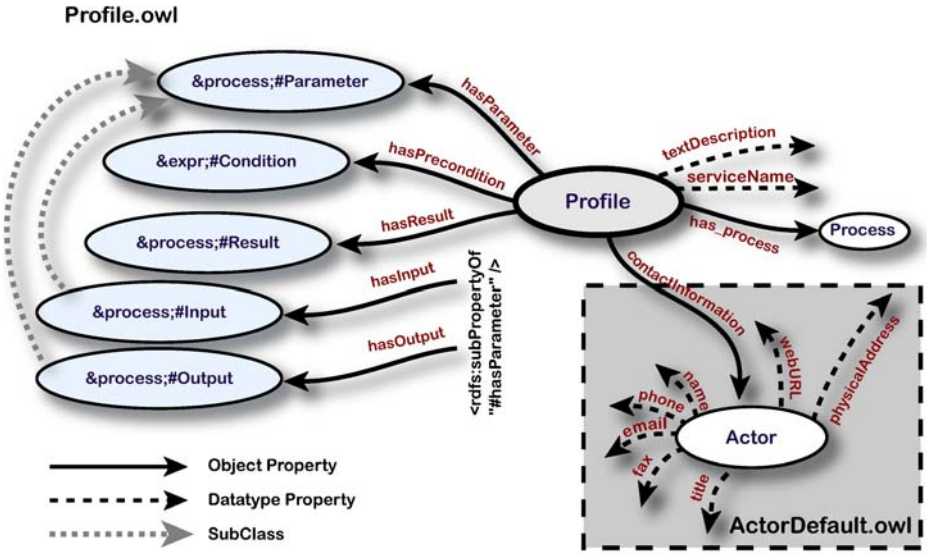

Figure 2 Top-level structure of the OWL-S profile.

profile type, which is the particular subclass of Profile that is being instantiated, and the inputs, outputs, preconditions, and results associated with the service. (The profile is not required to list all inputs, outputs, preconditions, and effects; it can omit some if they are deemed to be unimportant for the sake 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 the North American Industry Classification System (NAICS) [59]; 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 service, and specialized with appropriate properties. For example, for shipping services, there might be a ShippingServiceProfile subclass with the additional property geographicRegionServed, with GeographicRegion (from an appropriate online ontology) as its range.

Broadly speaking, an OWL-S service profile describes three aspects of that service. The first one is the *functional aspect*, which is represented using inputs, outputs, preconditions, and effects. The functional aspect represents (1) the information transformation computed by the service, from the inputs that the service expects to the outputs it generates; and (2) 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 online 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 service provider can specify explicitly what type of service it provides, and the service consumer 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 security and privacy requirements, the precision and timeliness (Quality of Service, or QoS) of the service provided, the cost structure, and aspects of provenance such as those captured in the Proof Markup Language [72]. As 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 consumers to define the service parameters they need.

```
<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>
```

Listing 1. A simple, partial OWL-S profile.

A partial example of a profile for a fictitious airline booking service, expressed in the RDF/XML syntax of OWL, is shown in Listing 1. The profile, an instance of FlightReservation-Profile (a subclass of Profile), describes inputs specifying the departure and arrival airports, 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 uniform resource identifiers (URIs) of concepts defined in some ontology. The ontology used in this example is the fictitious ontology http://fly.com/Onto. The advantage of using concepts from Web-addressable ontologies, rather than XML schemas, is that the descriptions use terms whose provenance is globally available, which refer to entities such as 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 the terms used by service providers and consumers are drawn from the same vocabulary. If the vocabularies are different, the discovery process may still be able to find appropriate services as long as the different terms are suitably related by their ontologies.

5 The process model subontology

Once a software agent has identified a service as likely to be relevant to its goals, it needs a detailed model of the service to determine whether this service can meet the agent's needs, and, if so, what constraints must be satisfied and what pattern of interactions are required to make use of the service.

An OWL-S process specifies, among other things, the possible patterns of interaction with a Web Service. Two sorts of processes can be invoked: atomic and composite processes. An *atomic process* is one that has no internal structure. It corresponds to a single interchange of inputs and outputs between a consumer and a provider. 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 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 [46].

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. In addition, there are properties that relate the process model to corresponding messages (referenced by the grounding), and to advertised capabilities descriptions in the corresponding OWL-S profile. Figure 3 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 also 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. In the following discussion of the process model, features are introduced using the presentation syntax, with illustrations of how that can be translated into the RDF/XML syntax.

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",
```

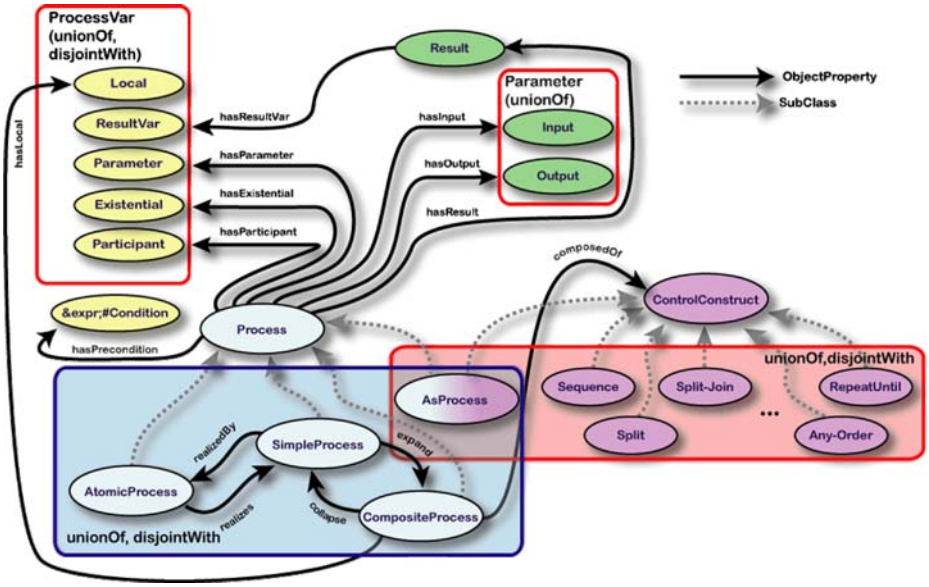

Figure 3 Top-level structure of the OWL-S process model.

```
    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 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 service consumer must do in order to cause a service, or a collection of services, to do something. Here is a specific, although somewhat artificial, example of an atomic process, extracted from a larger example that represents a book-buying service. The complete example, known as "CongoBuy," is described in [46]. Listing 2 gives a partial example of an atomic process, with a few details filled in.

```
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),
      preconditions:
              (hasAcctID(SignInInfo, AcctID)
                & credit-card-number(CardUsed, CC-Number)
                & validity(CardUsed, valid))
      outputs: ...
      results: ...);
…}
```

Listing 2. Top-level sketch of an atomic process in the presentation syntax.

Atomic processes are defined entirely by their inputs, outputs, local variables, preconditions, and results. All these fields are optional. Colons 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 by *result* conditions. As explained in Section 3, the distinction between preconditions and result conditions is that preconditions are things that the service consumer must verify before deciding to request the service, whereas result conditions are things that are checked by the

service provider in the course of execution and that impact the outcome produced. As noted earlier, results consist of effect and output specifications. Effects are simply propositions that become true. Values conveyed to output parameters are indicated by the special notation:

output(*output−parameter <= rdf−description*)

Result conditions are specified by the notation p |–> r (read "when p, then r"), which means that if p is true in the situation checked by the service provider 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 service consumer reading this process description and reasoning about it when interacting with the service provider, the only thing that is observable is the output message that is returned by the service provider. Based on the contents of that message, the service consumer can infer, based on the process description, that a particular result condition p is true, and that the listed effects of the process for that result condition should be true.

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

```
define atomic process ExpressCongoBuy
   (...,
        preconditions:
              (hasAcctID(SignInInfo, AcctID)
              & credit_card_number(CardUsed, CC_Number)
              & validity(CardUsed, valid))
        outputs (Status − CongoBuyOutputType),
        result:
              (forall (book − books:Book)
              (hasISBN(book, ISBN) and inStockBook(book)
                  |–> output(Status <= OrderShippedAcknowledgement)
                     & shippedTo(addressOf(AcctID), book))
              &
              (hasISBN(book, ISBN) and ~inStockBook(book)
                  |–> output(Status <= NotifyOutOfStockBook(ISBN)))))
```

Here, two result conditions are described: one where the book is in stock and another 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 is 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=" and 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 service profile and the grounding. It also allows process descriptions to be serialized in the standard RDF/XML notation of OWL.

We have left blanks in the RDF fragments wherever the presentation syntax includes a logical formula. There are many alternative approaches to expressing formulas and inserting them inside RDF, including N3 [9], RuleML [85], SWRL [32], SWRL-FOL [70], and SPARQL [73]. Some of these use the idea of "layering" the notation "on top of" RDF by encoding its syntax in RDF, whereas others embed the formulas as strings or quoted-XML constants.

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 many tasks 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 an example of the classic *planning problem* in Artificial Intelligence [75], often called *AI planning*, which in this context is called *dynamic service composition*. (See Section 7 for a discussion of approaches to composition.) Another

form of inference we expect OWL-S to be useful for is *plan recognition* [75]: given one or more actions by a software 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, or they may be constant values. For steps other than the first, they may also 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 [46].

```
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 that describe 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* [75]; we will come back to that in Section 9.

Another process reasoning problem, related to AI planning, is known as *nondeterminism reduction* [74]. To solve this problem, a reasoner must verify 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. Nondeterminism reduction is often more efficient than AI planning because the search space, i.e. the

number of process choices that the planner must make, is significantly reduced. In other words, 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:

P1 ;? P2 ;? ... ;? Pk

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

P1 ||; P2 ||; ... ||; Pk

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

If P1 then P2 else P3

split-join:

P1 ||> P2 ||> ... ||> Pk

and *split*:

P1 ||< P2 ||< ... ||< Pk

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

5.4 The semantics of OWL-S

As we saw in Section 2.1, OWL-S is an OWL-DL ontology. As OWL-DL has a well-defined semantics, so too does OWL-S, with respect to those things that are expressed in the ontology. However, there is more to be desired.

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 [71]. As a result, many sophisticated definitions and inter-relationships between OWL concepts cannot be described using the OWL language. For example, the OWL-S process model sub-ontology states that Split (or "||<" in the presentation syntax) is a class with certain basic relationships to other classes, such as ControlConstruct, Sequence, Split-Join, etc. However, the formal representation of the execution behavior associated with these terms is missing, because it cannot be adequately expressed in OWL-DL.

We have explored several solutions to this problem. The intended interpretation of the process vocabulary was defined in the Technical Overview of OWL-S [46]. It has also been defined by translation to first-order logic using the situation calculus [74], a language for reasoning about action and change; by using Petri nets [57]; and by using an execution semantics that incorporates subtype polymorphism [3]. More recently, the semantics of OWL-S was described in the Process Specification Language (PSL) [28], a formally axiomatized process ontology. 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.

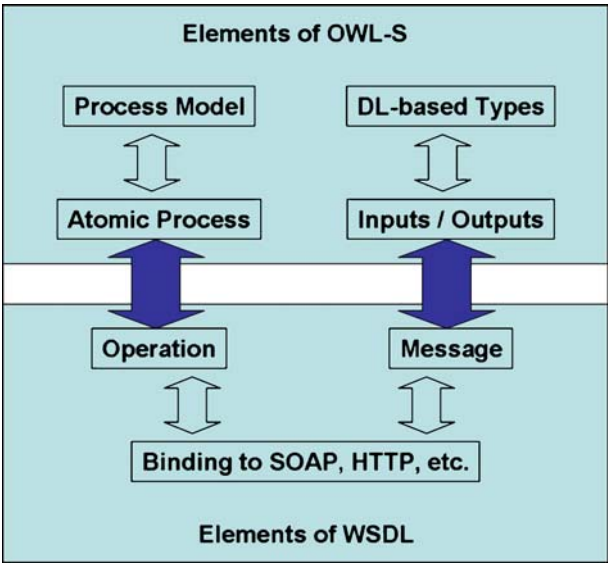

Figure 4 Overview of the relationship between OWL-S and WSDL.

6 The grounding subontology

The grounding subontology 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 consumer requesting a service and the service provider (i.e., the deployed Web Service) that 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 a service consumer into the appropriate WSDL messages for transmission to the service provider, and the translation of service output messages back into appropriate semantic descriptions (i.e., OWL descriptions) for interpretation by the service consumer.[4]

It should be noted that a grounding can be supplied at runtime, so as to allow for dynamic binding to a service provider. This frees the developer (or user) of service consumer code from having to select a specific service provider when that code is written (or deployed).

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 OWL classes based on description logic. 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.

[4] OWL-S allows for groundings to other interoperability frameworks besides WSDL; for example, at least one project has utilized a grounding that is based on UPnP. Here, we discuss only the WSDL grounding as it is included in the online releases and is by far the most widely used.

Figure 4 shows the essence of the relationship between OWL-S and WSDL. In this figure, the dark arrows represent explicit mappings declared in an OWL-S grounding. The clear arrows represent other relationships between elements that exist within OWL-S and WSDL specifications. WSDL *operations* correspond to OWL-S *atomic processes*. The *inputs* and *outputs* of an OWL-S atomic process correspond to *messages* (or parts of messages) in WSDL. Atomic processes and inputs/outputs are given meaning by their relationships with the larger process model of OWL-S and the type system of OWL, respectively. Operations and messages are given utility by their relationships with bindings and other elements of WSDL. By declaring these correspondences, the grounding in effect provides a bridge between the syntax- and protocol-oriented world of WSDL, and the world of OWL that relies on semantic elements defined using description logics.

On the WSDL[5] 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 service consumer sends one message and receives one reply (just as 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.[6] The input and output attributes declare the presence and types of the request to and response from the Web Service.

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>
```

[5] OWL-S release 1.2, and the example that follows, use WSDL 1.1, as do existing OWL-S tools. Due to limitations on development time, and a concern for maintaining compatibility with existing uses of OWL-S, we have not yet updated the grounding for use with WSDL 2.0. We plan to provide such an update in a subsequent release.

[6] Fortunately, WSDL is extensible at various points, so almost all the changes we require are just a matter of defining new namespaces. The few exceptions are described in [45].

For simplicity, we use the type xsd:string here for both inputs to the process; in practice, numeric or structured types can also be used.

These XML fragments are written in WSDL, not RDF, 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. A partial example is given in Listing 3.

```
<WsdlAtomicProcessGrounding rdf:ID="CongoCheckGrounding">
  <owlsProcess rdf:resource="&congo;#congoCheck"/>
  <wsdlOperation>
     <WsdlOperationRef>
        <portType>
           <xsd:uriReference rdf:value ="&congoWsdl;#CongoCheckPortType"/>
        </portType>
        <operation>
           <xsd:uriReference rdf:value="&congoWsdl#CheckBook"/>
        </operation>
     </WsdlOperationRef>
  </wsdlOperation>

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

  <wsdlOutputMessage rdf:resource = "&congoWsdl#CongoCheckOutput"/>
  <wsdlOutput>
     <wsdlOutputMessageMap>
        <owlsParameter rdf:resource = "&congo#Out-Avail"/>
        <wsdlMessagePart>
          …
        </wsdlMessagePart>
     </wsdlOutputMessageMap>
  </wsdlOutput>
</WsdlAtomicProcessGrounding>
```

Listing 3. Partial OWL-S Grounding of an Atomic Process.

This WsdlAtomicProcessGrounding, an excerpt from a hypothetical OWL-S grounding, defines the grounding of a particular atomic process named congoCheck. It expresses related, but complementary, information to the WSDL declarations related to CongoCheckPortType. The wsdlOperation property of a WsdlAtomicProcessGrounding object specifies the {portType, operation} pair from the WSDL specification. The wsdlInputMessage and wsdlOutputMessage properties specify how the atomic process maps into a request/response pattern. The wsdlInput and wsdlOutput properties specify message maps,

i.e. associations between OWL-S parameters and WSDL message parts. This information allows an OWL-S service consumer to find in the WSDL specification the information about the concrete embodiment of the abstract parameters.

Further details about OWL-S groundings may be found in [45] and in the Technical Overview [46].

7 Technology 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

OWL-S can be used in Web Service interactions in which the service consumer formulates a request based on its goals, queries a registry such as UDDI to find a service provider, and finally interacts with the service provider. Typical participants and their roles are shown in Figure 5. The main steps in this process are:

1. *Formulation of a request*: The service consumer needs to meet a goal, and would like to find a Web Service that can satisfy this goal. To find such a Web Service, the consumer expresses its goal as an OWL-S service profile. The translation of this goal into a service 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

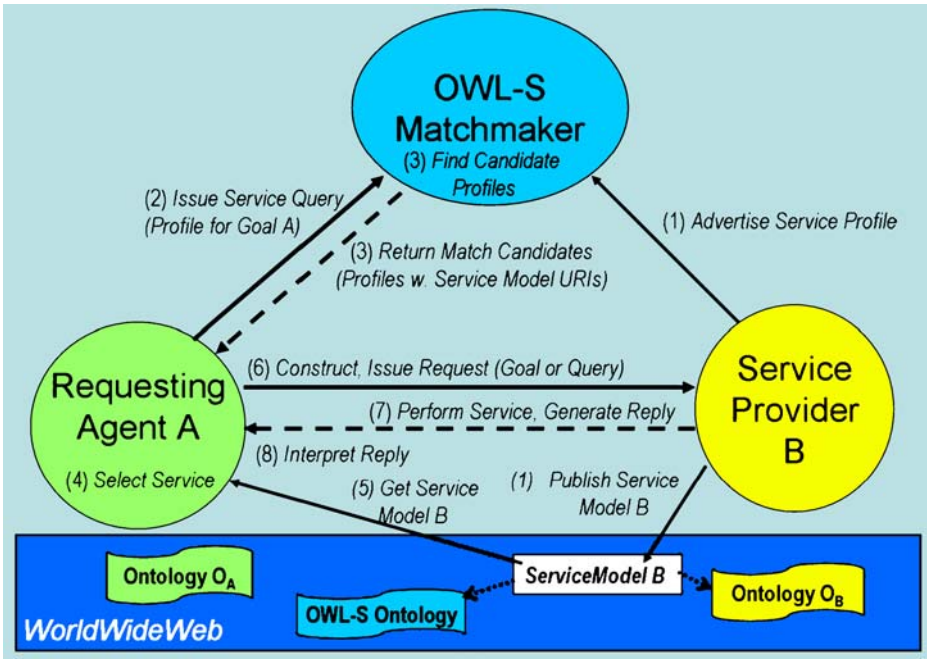

Figure 5 Typical OWL-S Semantic Web Service interaction model.

information produced, as well as the QoS that the consumer expects from a provider. An algorithm for generating these abstractions with OWL-S was described by Paolucci et al. [68].

2. *Service Request*: The service consumer 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 URIs for candidates (i.e., potential service providers) that might achieve that goal. The matching process relies on the relationships between the terms used to specify the advertisements and those in the request, both of which are defined using OWL ontologies [41, 43, 64].
4. *Selection*: The service consumer analyzes the candidates returned by the registry and selects a Web Service to interact with. As it is possible that none of the service providers found will precisely satisfy all the requirements and objectives of the consumer, the selection process may require consideration of a number of trade-offs [6].
5. *Invocation*: The service consumer determines how to make a request to the selected service provider 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 service. 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 provider receives the request and determines whether it can perform it. It may acknowledge the request, send an error, request additional information, or (generally, on completion) send a reply with the service results.
7. *Reply interpretation*: Upon receiving the reply, the service consumer parses it, in accordance with its WSDL specification, and uses the grounding specification to map it into the abstract outputs specified in the process model [62]. If the invocation of the service has been successful, the results of the service (as expressed in OWL-S) are known to hold true. From the service consumer's perspective, this additional knowledge may satisfy preconditions for other services.
8. *Goal satisfaction*: The service consumer uses the 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 [84].

When the process model of a service is a composite process, steps 5, 6 and 7 may be repeated a number of times, for the interaction between the client and the server may require numerous information exchanges. For example, any interaction with a business to consumer (B2C) Web site (e.g., an online bookshop) 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. In this case, the service consumer may need to interact with multiple Web Services concurrently to find out which one will provide the best result, or the best QoS. Detailed information about the semantics and invocation of composite processes can be found in Paolucci et al. [62].

This interaction process is consistent with the 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 [65, 82].

The difference here is that UDDI interactions require human software developers to do the job of interpreting the registry results and creating the calls to the services. Figure 5 shows how the process is implemented using OWL-S for more dynamic Web Service 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 software

agent (service consumer) 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 provider. This execution model has been implemented in the OWL-S virtual machine [62], and a variation on it was described by Sirin et al. [80].

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 subontology may be expected to provide value-added over existing Web Service discovery standards. Capability-based matching enables service consumers to find service providers that aptly match their requirements for particular results, while semantic matching allows flexibility in identifying the service parameters needed (and available to the service consumer) to specify the criteria of the requested service.

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 (technical models in UDDI parlance [17]). 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". As 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 subclass, or *subsumption* [5] relationships.

Broadly speaking, there are two ways of representing 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. An online bookshop, 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 instance, an online bookshop 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 that 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, because 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 covered by the service. On the other hand, enumerating all possible capabilities may be difficult, even in restricted domains. For example, consider the problem of representing translation services from a source language to a target language. 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 to. The 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 the one described by Li and Horrocks [41] assume the availability of ontologies of functionalities to express capabilities. Matching between a request and the available advertisements is reduced to a subsumption test. Different degrees of matching are detected depending on whether the advertisement and the request describe the same capability or whether one subsumes the other.

Other matching algorithms [23, 64, 82] assume that capabilities are described by the state transformation produced by the Web Service. Paolucci et al. [64] describe a matchmaking algorithm that compares the state transformations described in the request to the ones described in the advertisements. More specifically, subsumption relationships between the requested and provided types of inputs and outputs are examined to generate flexible matches, which are beyond the abilities of existing text-based matching engines. This algorithm performs two matches: one comparing outputs and another comparing inputs. If the output required by the service consumer 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 requirements.

Numerous extensions have been proposed to improve the matching algorithms by exploiting more features of subsumption relationships [41]. Instead of looking at only the parameter types, these 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. [8] describe one such ranking method based on the algorithms derived from hypergraph theory. Cardoso et al. [14] 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, however. Several researchers have also investigated how OWL-S can be integrated into existing Web Service discovery architectures. The OWL-S/UDDI matchmaker [82] 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 [65]. 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. For instance, Paolucci et al. [67] presented an approach for using OWL-S based discovery in Gnutella P2P environments. Elenius and Ingmarsson [24] achieved similar functionality in Juxtapose (JXTA) P2P infrastructure. Masuoka et al. [48] presented discovery methods using the UPnP protocol [86] in pervasive environments.

7.3 Composition

The automation of 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 WWW2007 conference trip" or "Buy me an 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 http://www.apple.com will debit your credit card and send you an iPod), as well as information provided by the service provider, 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 kind of information.

One key advantage of automating Web Service composition is that it allows service consumers 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 does not have the product that the service consumer needs, the overall composition does not fail; instead, 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 [55], 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 in Section 5.3). The first such system was based on Golog [55, 56]. This system interleaves the execution of information—providing services with the simulation of world—altering services to generate a sequence of Web Services customized to user's preferences and constraints. In a similar spirit, other researchers [39, 80] have used the paradigm of Hierarchical Task Network (HTN) planning [58] to perform automated Web Service composition. HTN planning has also been used to compose Web Services in the travel domain [66] and in the organization of a Business to Business (B2B) supply chain [80]. Other planning techniques that have been applied to the composition of OWL-S services, range from simple classical STRIPS-style planning [78] to extended estimated-regression planning [49], to Planning as Model Checking (PMC) [81] that deals with non-determinism, partial observability, and complex goals. Some of these systems limit themselves to dealing only with atomic processes, as operator-based planning systems are unable to represent composite processes. McIlraith and Fadel [54] 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. [18] present 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 [81] 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 [47], which helps the average user to find, select, compose and execute local, pervasive and remote Web Services to achieve common tasks. The STEER system combines services that adhere to different ontologies by utilizing semantic mappings and transformation functions that are also published as Web Services.

OWL-S can also be used to augment the standard WS-BPEL execution engine with an ontology to enable run-time discovery and binding of services. This is the case, for instance, in the semantic discovery service (SDS) [43], which performs semantic integration and interoperation in the context of dynamic customization of existing Web Service compositions (WS-BPEL workflows). Customizing constraints specified by the user enables the dynamic binding of user-customized services in the workflow. However, this dynamic binding sometimes requires further semantic integration. SDS uses OWL-S to describe the services and customize user constraints, and to perform Web Service discovery and semantic integration.

The SDS system was more recently extended to add an explanation facility that enables rich explanations of the system success or failure at creating an integrated composition [50]. The explainable system integrates Inference Web [53] 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, thereby providing transparency and audit information. This work provides an example of the use of non-functional parameters in describing Web Services with OWL-S. The information provided by these parameters is used to provide more meaningful explanations.

While all of this work is promising, we are still some distance from the goal of fully automating Web Service composition. With adoption of approaches to Web Service description such as OWL-S and advances in planning-related and program synthesis technologies, we believe that broad-scale automated Web Service composition will be feasible in the not-too-distant future.

8 Tools and extensions

The release of OWL-S has stimulated much work in the Semantic Web Services area. Applications from industry are appearing as the need for extra semantics becomes 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 review a number of applications, tools and extensions that build on or support OWL-S.

The OWL-S Editor [25], a plug-in to the popular ontology development tool Protégé [38], provides a comprehensive set of capabilities for creating and maintaining OWL-S service descriptions. In particular, this editor allows the user to build complex process models using a graphical user interface. Another OWL-S editor [76] developed by University of Malta provides functionality to create, validate and visualize OWL-S services. DINST [37] 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.

The OWL-S development environment (CODE) [83], developed by Carnegie Mellon University, is based on Eclipse [19] and supports developers of Semantic Web Services from the generation of Java code to the compilation of OWL-S descriptions, to the deployment and registration with UDDI. CODE includes modules such as a WSDL to OWL-S translator (WSDL2OWL-S) [63], an Eclipse-based editor for creating and editing OWL-S profiles, process models and groundings, and the OWL-S Virtual Machine [62]. The Mindswap OWL-S API [79] 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 versions of OWL-S.

Certain tools primarily focus on the generation of OWL-S descriptions from WSDL files. Besides the WSDL2OWL-S converter included in CODE, we can also mention Assam (Automated Semantic Service Annotation with Machine learning) [31], 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) [47] is another tool that helps users to generate executable OWL-S descriptions from WSDL files. The emphasis of this tool is on semi-automating the generation of *grounding specifications* by letting the user define mappings between ontologies and XML schema definitions through a graphical user interface. An extensible stylesheet language transformation (XSLT) [16] file is automatically generated based on the mappings that the user defined; the resulting OWL-S grounding is directly executable. OntoLink also facilitates the specification of semantic and syntactic mappings or transformations between concepts defined in different ontologies (in a similar fashion), so that services defined using disjoint ontologies can interoperate. 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 into 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. [22] present security annotations for OWL-S services to enable secure exchanges between service consumers and providers. These annotations are expressed using an ontology that describes credentials, security mechanisms and privacy options. Kagal et al. [35] extend this framework to express such annotations in Rei [36], a policy language that can be used for defining rules and constraints over domain-specific ontologies. The OWL-S Matchmaker system [64] is also extended for policy matching, and the OWL-S Virtual Machine (VM) [62] has been extended to enforce policies and security mechanisms. Similar extensions to OWL-S are proposed by Srinivasan et al. [82] to support policy and contract management using KAoS [87] services and tools. Other extensions to OWL-S have been developed for augmenting the process ontology to model Web Service brokers [68], for describing concrete application servers that facilitate reusing and combining Semantic Web software modules (e.g., ontology stores or reasoners), for describing Web Services in the bioinformatics domain [89], for enriching OWL-S with speech acts to describe agent-based Web Services [27], for modeling processes for learning software to assist and train U.S. Air Force planners [34], etc.

For more details, readers are referred to the OWL-S Web site [46], the Semantic Web Central Repository ([77]), and the public mailing list of the W3C's Semantic Web Services Interest Group (*public-sws-ig*; http://www.w3.org/2002/ws/swsig/).

9 Related work

In this section, we review related work and compare OWL-S to alternate approaches to linking Web Services and the Semantic Web. Because a thorough presentation of related work would require too much space, we limit our discussion to six of the main technologies to date: WS-BPEL, ebXML, CDL, SWSF, WSMO, and WSDL-S.

9.1 Web Services Business Process Execution Language (WS-BPEL)

WS-BPEL [4] enables the specification of executable business processes (including Web Services) and business process protocols in terms of execution logic and 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) describe the messages exchanged by the various Web Services within an interaction.

Business process protocols only consider protocol-relevant data and ignore process-internal data and computations. The effects of such computations on the business protocol are then described using nondeterministic data values. Executable processes, on the other hand, are described using a process description language that deals with both protocol-relevant and process-internal data. WS-BPEL also defines several mechanisms for recovering from faults, such as catching and handling of faults, and compensation handlers for specifying 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. In OWL-S, the process notation includes preconditions and results that enable automated tools to select and compose Web Services. The WS-BPEL notation includes more elaborate 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, i.e. described with sufficient information to enable the *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 [43] 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.

9.2 Electronic Business Using Extensible Markup Language (ebXML)

The ebXML language [40] addresses the broad problem of B2B interactions from a workflow perspective. Business interactions are described through two views: a Business Operational View and a Functional Service View. The Business Operational View deals with the semantics of business data transactions, and includes operational conventions, agreements, and mutual obligations between businesses. The Functional Service View deals with the supporting services: their capabilities, interfaces and protocols. Although ebXML does not restrict the means of B2B interaction to Web Services *per se*, it addresses many of the same issues as Web Services technology.

The ebXML language enables Web Services to describe the business processes they support and the services they offer using collaboration protocol profiles. 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. Collaboration protocol profiles contain industry classification, contact information, supported business processes, interface requirements, etc. They are registered within an ebXML registry [13] (similar to a UDDI registry) in which other Web Services and their business processes can be discovered. However, the format in which the business documents are specified is not in the scope of ebXML. The interacting Web Services have to agree upon this format a priori by creating a collaboration protocol agreement. As OWL-S provides a language for describing 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 [40].

9.3 Web Service Choreography Description Language (CDL)

CDL describes Web Service interactions in terms of externally observable behavior, typically exposed through message exchanges. Each participant in an interaction specifies an interface that describes the temporal ordering and logical dependence of messages it sends and receives. In addition, a global model can be specified to describe 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, the control flow constructs in CDL 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 the execution and generation of executable service compositions (e.g., using AI 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.

9.4 Semantic Web Services Framework (SWSF)

SWSF [7], published online in 2005 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 formal approach to process modeling, the process specification language [29]. SWSF specifies the Semantic Web Services language (SWSL) [7], a Web-oriented language with a logic programming layer and a first-order logic layer. SWSF uses SWSL to define the Semantic Web Services ontology (SWSO) [7], an ontology of service concepts that takes advantage of SWSL's greater expressiveness (relative to OWL) to more completely axiomatize the concepts. As SWSO is completely axiomatized in first-order logic, it avoids the need for using hybrid reasoning (based on both description logics and first-order logic) as in OWL-S.

9.5 Web Services Modeling Ontology (WSMO)

WSMO [88] shares with OWL-S and SWSF the vision that ontologies are essential to support automatic discovery, interoperation, and composition of Web Services. As SWSF, the WSMO effort defines an expressive Web-oriented language, the Web Service modeling language (WSML) [20], which provides a uniform syntax for subdialects ranging from description logics to first-order logic. Like OWL-S, WSMO declares inputs, outputs, preconditions, and results (although using a 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 specifying internal and external choreographies, 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, the Web Service modeling execution environment (WSMX) [30], and on the specification of mediators (i.e., mapping programs that solve interoperability problems between Web Services).

In the WSMO approach, mediators perform tasks such as translating between ontologies, or between the messages that one Web Service produces and the messages that another Web Service expects. WSMO includes a taxonomy of possible mediators that helps to classify the different tasks that mediators are supposed to fulfill. The definition of mediators in WSMO draws attention to some important translation tasks associated with Web Services. Unsurprisingly, these same translation tasks are needed in support of interactions with Web Services described in OWL-S. 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, and as such these mediation services can use the mechanisms provided by OWL-S for discovery, invocation and composition. In addition, mediators can be constructed through OWL-S enabled composition [69].

9.6 WSDL-S

WSDL-S [2] (the final 'S' of the acronym indicates the addition of semantics to WSDL) is a small set of proposed extensions to WSDL, by which semantic annotations may be associated with WSDL elements (e.g., 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, a notation resembling OWL-S declarations could be used to specify 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 specification, 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.

10 Conclusions

In this paper, we have described OWL-S, a representational framework that provides for more complete specifications of the capabilities and behavior of Web Services, so as to support greater automation of service-related activities (notably discovery and composition) than current representational frameworks for Web Services.

OWL-S consists of three parts. The first part is a vocabulary for describing service *profiles*, which are high-level descriptions of Web Services suitable for advertising them and supporting reasoning about their usefulness for achieving a given goal. The second part of OWL-S consists in a subontology and a presentation language for specifying *process models* in detail. The models describe services primarily in terms of inputs, outputs, preconditions, and results. The preconditions and results 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. The third part of OWL-S is a subontology 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. The subontology includes a vocabulary for describing mappings to WSDL ports and messages.

After describing OWL-S, we showed that it has already encountered a large success in the research community. We are encouraged by all the projects, tools, and extensions that are building on it, including efforts carried out by non-members of the OWL-S Coalition.

OWL-S could still be improved in a number of ways. First, there is currently no way of describing desirable security and QoS properties for Web Services; there is a dire need for both in the community, and promising work is underway. Second, the process model lacks exception handling, which would allow for a closer alignment with WSDL faults. Third, we plan to update the grounding subontology so that it works with WSDL 2.0. Fourth, further work is needed on algorithms that effectively support discovery, selection, and composition of services in large-scale, complex settings. Fifth, to encourage the community to develop OWL-S extensions, we are contemplating the prospect of adopting an open-source style of code evolution. Finally, as in other areas of the Semantic Web, we need more domain-level ontologies; there is a need to develop specializations of OWL-S profiles for a variety of domains, and for broad categories of services; in addition, for services that sell goods, ontology modules are needed for specifying cost models, negotiations, contracts, and guarantees.

Acknowledgments We wish to thank all the people who have contributed to OWL-S but did not co-author this paper, especially A. Ankolekar, G. Denker, D. Elenius, J. Giampapa, 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 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 *public-sws-ig* mailing list for discussion of OWL-S issues, and to the many researchers and users who have taken the trouble to share their suggestions and questions on this list. The editor of this special issue, J.P. Martin-Flatin, has been exceptionally helpful and skillful in guiding us to a more polished presentation of this material. We gratefully acknowledge funding from DARPA (DAML program) and thank J. Hendler, M. Burke and M. Greaves (DAML program managers) for their tremendous support of this effort.

References

1. Akkiraju, R., Farrell, J., Miller, J., et al.: 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
2. Akkiraju, R., Goodwin, R., Doshi, P., et al.: A method for semantically enhancing the service discovery capabilities of UDDI. In: Proceedings of the Workshop on Information Integration on the Web (IIWeb) at the 18th International Joint Conference on Artificial Intelligence (IJCAI). Acapulco, Mexico, August 2003
3. Ankolekar, A., Huch, F., Sycara, K.: Concurrent execution semantics of DAML-S with subtypes. In: Proceedings of the 1st International Semantic Web Conference (ISWC). Sardinia, Italy, June 2002
4. Arkin, A., Askary, S., Bloch, B., et al. (eds.): Web Services Business Process Execution Language Version 2.0. Committee Draft, September 2005. Available at: http://www.oasis-open.org/committees/download.php/14616/wsbpel-specification-draft.htm
5. Baader, F., Calvanese, D., McGuinness, D., et al.: The description logic handbook; theory, implementation, and applications. Cambridge University Press (2003)
6. Balke, W.T., Wagner, M.: Towards personalized selection of web services. In: Proceedings of the 12th International World Wide Web Conference (WWW). Budapest, Hungary, May 2003
7. Battle, S., Bernstein, A., et al.: Semantic Web services framework (SWSF) overview. W3C Member Submission, September 2005. Available at http://www.daml.org/services/swsf/1.0/
8. Benatallah, B., Hacid, M., Rey, C., et al.: Request rewriting-based web service discovery. In: Proceedings of the 2nd International Semantic Web Conference (ISWC), pp. 335–350. Sanibel Island, FL, USA, October 2003

9. Berners-Lee, T.: Primer: Getting into RDF and the semantic web using N3. 2000. Available at: http://www.w3.org/2000/10/swap/Primer.html
10. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic Web. Scientific American, May 2001
11. Booth, D., Haas, H., McCabe, F., et al. (eds.): Web services architecture. W3C Working Group Note, 11 February 2004. Available at: http://www.w3.org/TR/ws-arch/
12. Box, D., Ehnebuske, D., Kakivaya, G., et al. (eds.): Simple object access protocol (SOAP) 1.1, W3C Note, 8 May 2000. Available at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
13. Breininger, K., Chiusano, J.M., Damodaran, S., et al. (eds.): Registry information model v2.0. Available at: http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf
14. Cardoso, J., Sheth, A.: Semantic E-workflow composition. J. Intell. Inf. Syst. **21**(3), 191–225 (2003)
15. Christensen, E., Curbera, F., Meredith, G., et al.: Web services description language (WSDL) 1.1, W3C Note, 15 March 2001. Available at: http://www.w3.org/TR/2001/NOTE-wsdl-20010315
16. Clark, J. (ed.): XSL transformations (XSLT), W3C recommendation, 16 November 1999. Available at: http://www.w3.org/TR/xslt
17. Clement, L., Hately, A., von Riegen, C., et al. (eds.): The universal description, discovery and integration (UDDI) Version 3.0.2 Specification, October 2004. Available at: http://www.uddi.org/
18. Constantinescu, I., Faltings, B., Binder, W.: Large scale, type-compatible service composition. In: Proceedings of the 2nd International Conference on Web Services (ICWS). San Diego, CA, USA, July 2004
19. D'Anjou, J., Fairbrother, S., Kehn, D., et al.: The Java developer's guide to eclipse. Addison-Wesley (2005)
20. De Bruijn, J., Lausen, H., Polleres, A., et al.: The Web service modeling language WSML: an overview, DERI technical report 2005-06-16, June 2005. Available at: http://www.wsmo.org/wsml/wsml-resources/deri-tr-2005-06-16.pdf
21. Dean, M., Schreiber, G. (eds.): OWL Web ontology language reference, W3C Recommendation, 10 February 2004. Available at: http://www.w3.org/TR/2004/REC-owl-ref-20040210/
22. Denker, G., Kagal, L., Finin, T., et al.: Security for DAML web services: annotation and matchmaking. In: Proceedings of the 2nd International Semantic Web Conference (ISWC), pp. 335-350. Sanibel Island, FL, USA, October 2003
23. Di Noia, T., Di Sciacio, E., Donini, F.M., et al.: Semantic matchmaking in a P-2-P electronic marketplace. In: Proceedings of the 18th Annual ACM Symposium on Applied Computing (SAC), pp. 582–586. Melbourne, FL, USA, March 2003
24. Elenius, D., Ingmarsson, M.: Ontology-based service discovery in P2P networks. In: Proceedings of the 1st International Workshop on Peer-to-Peer Knowledge Management (P2PKM) at the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous). Boston, MA, USA, August 2004
25. Elenius, D., Denker, G., Martin, D., et al.: The OWL-S Editor—A development tool for semantic web services. In: Proceedings of the 2nd European Semantic Web Conference (ESWC). Heraklion, Greece, May 2005
26. Enderton, H.: A Mathematical introduction to logic, 2nd edn. Academic Press (2000), December
27. Gibbins, N., Harris, S., Shadbolt, N.: Agent-based semantic web services. In: Proceedings of the 12th International World Wide Web Conference (WWW). Budapest, Hungary, May 2003
28. Gruninger, M.: Applications of PSL to semantic web services. In: Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB) at the 29th International Conference on Very Large Data Bases (VLDB). Berlin, Germany, September 2003
29. Gruninger, M., Menzel, C.: Process specification language: Theory and applications. AI Mag. **24**, 63–74 (2003)
30. Haller, A., Cimpian, E., Mocan, A., et al.: WSMX—a semantic service-oriented architecture. In: Proceedings of the International Conference on Web Services (ICWS). Orlando, FL, USA, July 2005
31. Heß, A., Johnston, E., Kushmerick, N.: ASSAM: A tool for semi-automatically annotating semantic web services. In: Proceedings of the 3rd International Semantic Web Conference (ISWC). Hiroshima, Japan, November 2004
32. Horrocks, I., Patel-Schneider, P.F., Boley, H., et al.: SWRL: A semantic web rule language combining oWL and ruleML. W3C Member Submission, 21 May 2004. Available at: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
33. Horstmann, K.: Big Java, 2nd edn. Wiley (2005)
34. Integrated Learning research program. Home page: http://www.darpa.mil/IPTO/Programs/il/index.htm
35. Kagal, L., Paoucci, M., Srinivasan, N., et al.: Authorization and privacy for semantic Web services. IEEE Intell. Syst. **19**(4), 50–56 (2004), July
36. Kagal, L., Finin, T., Joshi, A.: A policy based approach to security on the semantic web. In: Proceedings of the 2nd International Semantic Web Conference (ISWC). Sanibel Island, FL, USA, October 2003
37. Klein, M., König-Ries, B.: A process and a tool for creating service descriptions based on DAML-S. In: Proceedings of the 4th International Workshop on Technologies for E-Services (TES) at the 29th International Conference on Very Large Data Bases (VLDB). Berlin, Germany, September 2003

38. Knublauch, H., Fergerson, R.W., Noy, N.F., et al.: The Protégé OWL plugin: An open development environment for semantic web applications. In: Proceedings of the 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan, November 2004
39. Kuter, U., Sirin, E., Nau, D., et al.: Information gathering during planning for web service composition. In: Proceedings of the 3rd International Semantic Web Conference (ISWC). Hiroshima, Japan, November 2004
40. Levine, P., Clark, J., Casanave, C., et al.: ebXML Business Process Specification Schema. Available at: http://www.ebxml.org/specs/index.htm
41. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proceedings of the 12th International World Wide Web Conference (WWW), pp. 331–339. Budapest, Hungary, May 2003
42. Liberty, J.: Programming C#, 4th edn. O'Reilly (2005)
43. Mandell, D., McIlraith, S.: Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In: Proceedings of the 2nd International Semantic Web Conference (ISWC), pp. 227–241. Sanibel Island, FL, USA, October 2003. Code for the working system is available at: http://projects.semwebcentral.org/
44. Martin, D., Cheyer, A., Moran, D.: The open agent architecture: A framework for building distributed software systems. Appl. Artif. Intell. **13**(1–2), 91–128 (1999)
45. Martin, D., Burstein, M., Lassila, O., et al.: Describing web services using OWL-S and WSDL. in Release 1.2 of OWL-S, available at: http://www.daml.org/services/owl-s/1.2/owl-s-wsdl.html
46. Martin, D., Burstein, M., McDermott, D., et al.: OWL-S 1.2 Release. Available at: http://www.daml.org/services/owl-s/1.2/
47. Masuoka, R., Labrou, Y., Parsia, B., et al.: Ontology-enabled pervasive computing applications. IEEE Intell. Syst. **18**(10), 68–72 (2003)
48. Masuoka, R., Parsia, B., Labrou, Y.: Task computing—the semantic web meets pervasive computing. In: Proceedings of the 2nd International Semantic Web Conference (ISWC). Sanibel Island, FL, USA, October 2003
49. McDermott, D.: Estimated-regression planning for interactions with web services. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS). Toulouse, France, April 2002
50. McGuinness, D.L., Mandell, D., Mcllraith, S., et al.: Explainable semantic discovery services. Stanford Networking Research Center Project Review, Stanford. CA, USA, February 2005
51. McGuinness, D.L.: Ontologies come of age. In: Fensel, D., Hendler, J., Lieberman, H., et al. (eds.) Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press (2003)
52. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation, 10 February 2004. Available at: http://www.w3.org/TR/2004/REC-owl-features-20040210/
53. McGuinness, D.L., da Silva, P.P.: Explaining answers from the semantic web: The inference web approach. Journal of Web Semantics **1**(4), 397–413 (2004), October
54. McIlraith, S., Fadel, R.: Planning with complex actions. In: Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR) at the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 356–364. Toulouse, France, April 2002
55. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 482–493. Toulouse, France, April 2002
56. McIlraith, S., Son, T.C., Zeng, H.: Semantic web services. IEEE Intell. Syst. **16**(2), 46–53 (2001), March–April
57. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: Proceedings of the 11th International World Wide Web Conference (WWW). Honolulu, HI, USA, May 2002
58. Nau, D., Au, T.C., Ilghami, O., et al.: SHOP2: An HTN planning system. J. Artif. Intell. Res. **20**, 379–404 (2003)
59. North American Industry Classification System. Home page: http://www.census.gov/epcd/www/naics.html
60. Nwana, H.: Software agents: An overview. Knowl. Eng. Rev. **11**(2), 205–244 (1995)
61. OWL-S Coalition. Home page: http://www.daml.org/services/owl-s/
62. Paolucci, M., Ankolekar, A., Srinivasan, N., et al.: The DAML-S virtual machine. In: Proceedings of the 2nd International Semantic Web Conference (ISWC), pp. 335–350. Sanibel Island, FL, USA, October 2003
63. Paolucci, M., Srinivasan, N., Sycara, K., et al.: Toward a semantic choreography of web services: From WSDL to DAML-S. In: Proceedings of the 1st International Conference on Web Services (ICWS). Las Vegas, NV, USA, June 2003
64. Paolucci, M., Kawamura, T., Payne, T.R., et al.: Semantic matching of web services capabilities. In Proceedings of the 1st International Semantic Web Conference (ISWC), Sardinia, Italy, June 2002
65. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Importing the semantic web in UDDI. In: Proceedings of the Workshop on Web Services, E-Business and the Semantic Web (WSeBT) at the 14th International Conference on Advanced Information Systems Engineering (CaiSE). Toronto, Ontario, Canada, May 2002

66. Paolucci, M., Sycara, K., Kawamura, T.: Delivering semantic web services. In: Proceedings of the 12th International World Wide Web Conference (WWW), pp. 111–118. Budapest, Hungary, May 2003
67. Paolucci, M., Sycara, K., Nishimura, T., et al.: Using DAML-S for P2P discovery. In: Proceedings of the 1st International Conference on Web Services (ICWS), pp. 203–207. Las Vegas, NV, USA, June 2003
68. Paolucci, M., Soudry, J., Srinivasan, N., et al.: A broker for OWL-S web services. In: Proceedings of the 1st International Semantic Web Services Symposium, 2004 AAAI Spring Symposium Series, Stanford, CA, USA, March 2004. Available at: http://www.daml.ecs.soton.ac.uk/SSS-SWS04/40.pdf
69. Paolucci, M., Srinivasan, N., Sycara, K.: Expressing WSMO Mediators in OWL-S. In: Proceedings of the Semantic Web Services Workshop (SWS) at the 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan, November 2004. CEUR Workshop Proceedings, Vol. 119, paper 10. Available at: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-119/
70. Patel-Schneider, P.F.: A Proposal for a SWRL Extension Towards First-order Logic. W3C Member Submission, 11 April 2005. Available at: http://www.w3.org/Submission/2005/SUBM-SWRL-FOL-20050411/
71. Patel-Schneider, P., Hayes, P., Horrocks, I. (eds.): OWL Web Ontology Language: Semantics and Abstract Syntax. W3C Recommendation, 10 February 2004. Available at: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/
72. Pinheiro da Silva, P., McGuinness, D.L., Fikes, R.: A proof markup language for semantic Web services. Inf. Syst. **31**(4–5), 381–395 (2006), June–July
73. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Working Draft, 4 October 2006. Available at: http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/
74. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
75. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall (2002)
76. Scicluna, J., Abela, C., Montebello, M.: Visual modelling of OWL-S services. In: Proceedings of the IADIS WWW/Internet 2004 Conference (ICWI). Madrid, Spain, October 2004
77. Semantic Web Central Repository. Home page: http://www.semwebcentral.org
78. Sheshagiri, M., desJardins, M., Finin, T.: A planner for composing services described in DAML-S. In: Proceedings of the Workshop on Web Services and Agent-Based Engineering (WSABE) at the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). Melbourne, Australia, July 2003
79. Sirin, E., Parsia, B.: The OWL-S Java API. Poster. In: Proceedings of the 3rd International Semantic Web Conference (ISWC). Hiroshima, Japan, November 2004
80. Sirin, E., Parsia, B., Wu, D., et al.: HTN Planning for Web Service Composition using SHOP2. Journal of Web Semantics **1**(4), 377–396 (2004)
81. Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intell. Syst. **19**(4), 42–49 (2004)
82. Srinivasan, N., Paolucci, M., Sycara, K.: An efficient algorithm for OWL-S based semantic search in UDDI. In: Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC) at the 2nd International Conference on Web Services (ICWS), pp. 96–110. San Diego, CA, USA, July 2004
83. Srinivasan, N., Paolucci, M., Sycara, K.: CODE: A Development Environment for OWL-S Web Services. Technical Report CMU-RI-TR-05-48, Robotics Institute, Carnegie Mellon University, October 2005. Available at: http://www.ri.cmu.edu/pubs/pub_5159.html
84. Sycara, K., Paolucci, M., Ankolekar, A., et al.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics **1**, (1), 27–46 (2003), December
85. The Rule Markup Initiative. Home page: http://www.dfki.uni-kl.de/ruleml/
86. UPnP Forum.: UPnP Device Architecture, Version 1.0. June 2000. Available at: http://www.upnp.org/download/UPnPDA10_20000613.htm
87. Uszok, A., Bradshaw, J.M., Johnson, M., et al.: KAoS policy management for semantic web services. IEEE Intell. Syst. **19**(4), 32–41 (2004), July
88. Web Service Modeling Ontology (WSMO). Home page: http://www.wsmo.org/
89. Wroe, C., Stevens, R., Goble, C., et al.: A suite of DAML+OIL ontologies to describe bioinformatics web services and data. Int. J. Coop. Inf. Syst. **12**(2), 197–224 (2003)

David Martin is a Senior Computer Scientist in the Artificial Intelligence Center of SRI International, where he has done research since 1994. He is involved in ongoing research on agent-based systems, knowledge representation, natural language processing, and intelligent assistants. He is a founding member of the OWL-S Coalition, and served as co-chair of the language subcommittee of The Semantic Web Services Initiative.

Mark Burstein is a Division Scientist and the Director of the Human Centered Systems Group at BBN Technologies. He is co-chair of the Semantic Web Services Initiative's Architecture Committee, and a founding member of the OWL-S coalition. He is author or co-author of over 70 papers on topics including semantic Web services, ontology translation, multi-agent systems, mixed-initiative planning and scheduling, plausible reasoning, knowledge acquisition and machine learning.

Drew McDermott is Professor of Computer Science at Yale University. His research is in planning and knowledge representation, with side excursions into philosophy. He has been one the prime movers of the International Planning Competition, and the language it spawned, the Planning Domain Definition Language (PDDL). He is on the editorial board of *Artificial Intelligence*, and is a Fellow of the American Association for Artificial Intelligence (AAAI).

Sheila McIlraith is an Associate Professor in the Department of Computer Science, University of Toronto. She is a founding member of the OWL-S Coalition. She has done significant research in the area of semantic Web services, including work on OWL-S and on the Semantic Web Services Framework.

Massimo Paolucci is a Senior Researcher at DoCoMo European Laboratories, where he is working on the application of semantic Web services technologies at service provisioning for mobile services. Before joining DoCoMo, he worked at Carnegie Mellon University on service discovery and composition. He has been a member of the OWL-S coalition from its inception, and he is a former member of the UDDI Technical Committee.

Katia Sycara is a Professor in the School of Computer Science at Carnegie Mellon University and holds the Sixth Century Chair in Computing Science at the University of Aberdeen. She is a Fellow of the AAAI and of the IEEE. She has authored or co-authored more than 300 technical papers and given numerous invited talks. She is a founding member both of the OWL-S Coalition and the Semantic Web Services Initiative.

Deborah McGuinness is the acting director and senior research scientist at Knowledge Systems, Artificial Intelligence Laboratory, Stanford University. She is a leading expert in knowledge representation and reasoning languages and systems and has worked in ontology creation and evolution environments for over 20 years. Most recently, Deborah is best known for her leadership role in semantic Web research, and for work on explanation, trust, and applications of semantic Web technology, particularly for scientific applications.

Evren Sirin received his Ph.D. in Computer Science from University of Maryland, College Park in 2006, for which he worked in the areas of Web service composition, Description Logic reasoning and AI planning. He is currently working as a research scientist at the start-up R and D firm Clark and Parsia, LLC which specializes in semantic Web and advanced systems.

Naveen Srinivasan is a consultant at webMethods, a leading business integration software provider. His research interests include the semantic Web, Web services, and tools for multiagent systems. He received his MS in computer science from the University of Maryland, Baltimore County.