

Bringing Semantics to Web Services: The OWL-S Approach

David Martin¹, Massimo Paolucci², Sheila McIlraith³, Mark Burstein,
Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne,
Marta Sabou, Monika Solanki, Naveen Srinivasan, Katia Sycara

¹ Artificial Intelligence Center, SRI International,
Menlo Park, CA, USA
martin@ai.sri.com

² Robotics Institute, Carnegie Mellon University,
Pittsburgh, PA, USA
paolucci+@ri.cmu.edu

³ Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract. Service interface description languages such as WSDL, and related standards, are evolving rapidly to provide a foundation for interoperation between Web services. At the same time, Semantic Web service technologies, such as the Ontology Web Language for Services (OWL-S), are developing the means by which services can be given richer semantic specifications. Richer semantics can enable fuller, more flexible automation of service provision and use, and support the construction of more powerful tools and methodologies. Both sets of technologies can benefit from complementary uses and cross-fertilization of ideas. This paper shows how to use OWL-S in conjunction with Web service standards, and explains and illustrates the value added by the semantics expressed in OWL-S.

1 Introduction

The promise of Web services and the need for widely accepted standards enabling them are by now well recognized, and considerable efforts are underway to define and evolve such standards in the commercial realm. In particular, the Web Services Description Language (WSDL) is already well established as an essential building block in the evolving stack of Web service technologies, and is being developed and standardized in the W3C's Web Services Description Working Group . WSDL, in essence, allows for the specification of the syntax of the input and output messages of a basic service, as well as other details needed for the invocation of the service. WSDL does not, however, support the specification of workflows composed of basic services. In this area, the Business Process Execution Language for Web Services (BPEL4WS) , under development at OASIS, has the most prominent status. The W3C's Web Services Choreography Working Group also has been chartered to explore this technical area. With respect to registering Web services, for purposes of

advertising and discovery, Universal Description, Discovery and Integration (UDDI) has received the most attention to date.

At the same time, recognition is growing of the need for richer semantic specifications of Web services, so as to enable fuller, more flexible automation of service provision and use, support the construction of more powerful tools and methodologies, and promote the use of semantically well-founded reasoning about services. Because a rich representation language permits a more comprehensive specification of so many different aspects of services, they can provide a better foundation for a broad range of activities, across the Web service lifecycle. For example, richer semantics can support greater automation of service selection and invocation, automated translation of message content between heterogeneous interoperating services, automated or semi-automated approaches to service composition, and more comprehensive approaches to service monitoring and recovery from failure. Further down the road, richer semantics can help to provide fuller automation of such activities as verification, simulation, configuration, supply chain management, contracting, and negotiation of services.

To meet this need, researchers have been developing languages, architectures and related approaches; the resulting body of work goes under the heading of *Semantic Web services*. In particular, the authors of this paper, members of the OWL-S Coalition, are developing the Ontology Web Language for Services (OWL-S), which seeks to provide the building blocks for encoding rich semantic service descriptions, in a way that builds naturally upon OWL, the Semantic Web language undergoing standardization at the W3C.

OWL-S (formerly DAML-S) and other related work may be viewed as efforts to lay the foundations for the most effective evolution of Web service-related capabilities that can be supported with current and maturing technologies. But at the same time, our goal is to promote the rapid adoption of semantically expressive technologies that are already well-understood, and there is much that can be done in the near term. Therefore, we have taken pains to construct mechanisms by which OWL-S can be used along with the dominant Web services standards, such as WSDL. The purpose of this paper is to provide an initial roadmap towards deployment of Semantic Web services, using OWL-S in conjunction with WSDL and related standards, and to begin to provide a clear delineation of the potential benefits of richer semantics in specifying Web services.

In this paper, we show how to use OWL-S in conjunction with Web service standards — focusing particularly on its use with WSDL — and explain and illustrate the value added by the semantics expressed in OWL-S. We illustrate these points using a simple running example, which is presented in Section 2. Section 3 explains how OWL-S can be used to describe the example service, and can be *grounded* in the WSDL description. In Sections 4 – 6, we show how the combined OWL-S specifications can be used to support service enactment, service discovery, and service composition, respectively. Sections 7 and 8 present related work and a summary of our approach and its potential importance for the future of Web services.

2 A Motivating Example

Amazon.com provides an openly available Web service which allows client programs to browse Amazon's databases, locate books and other products and put them in a Web 'shopping cart' that can be accessed from the main Amazon Web site using a browser to finalize purchases. Web service client programs, written to avail themselves of the provided WSDL specification of the service, can request a wide range of semi-structured keyword searches on the Amazon Web site data base. Clients can search for books with a given author, products from a particular manufacturer, or DVDs of movies by a given director. Customer reviews to seller profiles are also accessible. (For more information visit <http://www.amazon.com/webservices>.)

Amazon provides a WSDL specification of its Web service describing the operations that can be performed, along with tutorials and code for sample clients. The tutorials and code samples are needed so that *programmers* can properly utilize the WSDL interface. No software system (agent) could read and utilize the WSDL interface without human assistance, because the WSDL specification language provide no means of including representations of the semantics of the defined operations and associated messages elements. For example, all of the inputs and outputs (parts of corresponding WSDL messages) in Amazon's WSDL operations are typed as strings. We take it as a key objective of Semantic Web services and OWL-S to bridge that gap. OWL-S provides a language for specifying the function (preconditions and effects) of an operation and semantic types for each of the inputs and outputs of the service. OWL-S is based on the assumption that the *definitions* of these semantic concepts are available at referenced URIs on the Semantic Web, so that the service and client programs have a means of sharing terms and clients can find the definitions of all referenced concepts, represented in the OWL semantic description language.

The result is that, by taking an OWL-S description of the services together with the WSDL description, a client program can distinguish the operation taking a model number of a camcorder from one requiring a book author's name in what would otherwise look to be similar request operations to search the database. The client can also properly interpret the result of those queries, without programming specific to that interface. By using OWL for semantic typing of the elements of communication, our Amazon client [26] can automatically identify which inputs (elements of its own internal goals) are required for the kind of search desired, *transform* those elements, if necessary, to the appropriate (string) form, and interpret the elements of a returned message.

The WSDL specification of the *outputs* of each call to the service similarly lacks semantic definition. All Amazon's defined search operations return results using the same data structure, named *Details*, regardless of what product information is requested. Product types can be inferred from the data structure by analyzing the elements that are filled in. For example, *Details* contains a field for *Authors* which is used to describe books, and a field *Directors* which is used to describe movies. It is up to the client to recognize that values in these fields indicate whether it is a book or a movie. Even if the type of item specified in a *Details* record were clearly identified

in a Type field by the interface designer, WSDL provides no way uniform way of enabling such interpretations.

WSDL's lack of semantic descriptions of the meaning of inputs and outputs makes it impossible to develop software clients that can, without human assistance, dynamically find and successfully invoke a service. WSDL specifications of services must be interpreted by programmers, who interpret the names of keywords given for message elements using other supporting documentation to integrate *specific* services with their client applications. The objective of Semantic Web services is to support clients that can find and correctly utilize newly discovered services without additional programming. Such clients will, for example, be capable of finding sites selling books or CDs and comparing the prices of particular items from those sites even when those services' WSDL interfaces were not known, in advance, to the developer of that client. Semantic Web service clients will be able to interact with any such service as long as they describe their WSDL operations in terms of compatible, shared, Semantic Web representations for books, CDs, information requests, purchase/sale requests and monetary units. For the same reasons, these Web services can be discovered in a service repository using semantic descriptions characterizing the services provided with no (or minimal) human intervention. Furthermore, both the discovery and use of such services is robust in the face of service design changes over time, because the service protocols would be republished and re-interpreted by the client software at the time of use.

3 Introducing Semantics

OWL-S (formerly DAML-S) is an OWL ontology with three interrelated subontologies, known as the profile, process model, and grounding. In brief, the profile is used to express "what a service does", for purposes of advertising, constructing service requests, and matchmaking; the process model describes "how it works", to enable invocation, enactment, composition, monitoring and recovery; and the grounding maps the constructs of the process model onto detailed specifications of message formats, protocols, and so forth (normally expressed in WSDL). This paper is primarily concerned with some of the fundamental constructs of the process model, and their groundings.

WSDL 1.1 allows for the specification of *operations* as the basic building blocks of Web services. (Although the development of WSDL 2.0 is well underway, it is not yet stable enough at time of writing to allow for OWL-S groundings based on it.) Operations provide the organizational structure around which input/output message syntax and patterns are specified. OWL-S provides an analogous but somewhat more abstract construct known as the *atomic process*, which is characterized primarily in terms of its inputs, outputs, preconditions, and effects (IOPEs).

The inputs and outputs of an atomic process are given types from the (class-hierarchical, description logic-based) typing system of OWL, which allows for the use of concepts defined and shared as part of the Semantic Web. For example, the accompanying code sample gives a simplified OWL-S declaration of an atomic process with its IO specifications. (Due to space constraints, we omit namespace qualifiers in this example). In this case, AtomicProcess, input, output, and

parameterType belong to the OWL-S process model namespace. We assume that Human, BookTitle, and ISBN are classes defined in appropriate domain ontologies having other namespaces.

```
<AtomicProcess ID="AuthorSearch">
  <hasInput>
    <Input ID="Author">
      <parameterType resource="#Human">
    </Input>
  </hasInput>
  <hasInput>
    <Input ID="Title">
      <parameterType resource="#BookTitle">
    </Input>
  </hasInput>
  <hasOutput>
    <Output ID="BookID">
      <parameterType resource="#ISBN">
    </Output>
  </hasOutput>
</AtomicProcess>
```

The grounding for this atomic process would establish its correspondence to a particular WSDL operation, and the correspondence of each IO element to a particular WSDL message part element. Also, if needed, the grounding could specify an XSLT script to transform each OWL-expressed input (an instance of the relevant

class) to the precise syntactic form specified by WSDL, and vice versa for outputs. Additional details and examples of OWL-S groundings may be found in .

An important part of Semantic Web service description is the specification of conditions and constraints, including the preconditions and effects of a process or service. Preconditions are logical formulae that need to be satisfied (ensured to be true) by a service requestor prior to the execution of the service. Effects are logical formulae that state what will be true upon the successful execution of the service. OWL-S Effects are the side-effects of the execution of the service. Many information-providing services have no side effects. Nevertheless, other, often transaction-oriented, services do have side effects in the world, such as debiting the user's credit card, sending goods, etc. Description of these side effects is critical to certain aspects of Web service automation, as we discuss in subsequent sections.

For the specification of a process' preconditions and effects, OWL-S allows for the use of a more expressive language than OWL, such as RuleML , DRS , or the recently proposed OWL Rules Language . For example, one of these languages could be used to express a precondition for a bookselling service, stating that one must have a valid account and a valid credit card in order to make a purchase.

A more complete exposition of OWL-S may be found at , and in the various papers listed there. In the following three sections, we discuss several case studies of OWL-S' contributions in the areas of service enactment, discovery, and composition.

4 Enactment

Enactment is the process by which a client applies a declarative description of a service to request something of the service and interpret the response. Here, the description being interpreted is the OWL-S process model published by the service along with the WSDL specs to which it is grounded. Enactment begins by reasoning backwards from the inputs required by the selected service to find the information

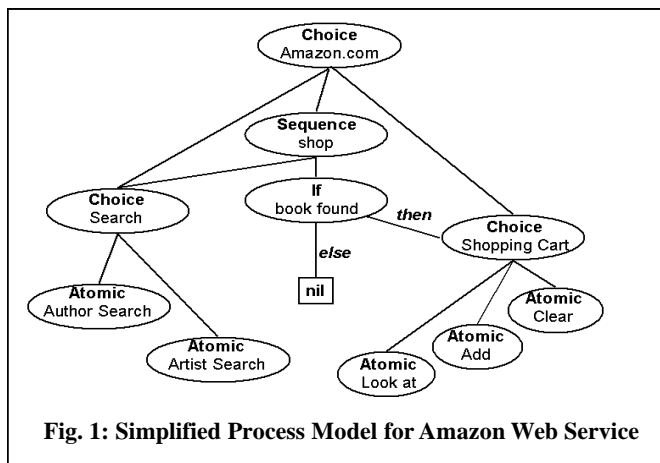
available to the client that is required to successfully invoke the service. These input values are then mapped via the service grounding onto the corresponding elements of a WSDL message pattern, resulting finally in a message being communicated to the service. The output message (if any) is handled by essentially reversing the process. A WSDL output message that is received by the client is transformed (again, via the grounding) into an OWL-S representation of the content of that message which can be interpreted by the client's reasoning engine.

To implement this process with Amazon's Web service, we first require an OWL-S description of that service that more fully represents the inputs and outputs of the service. We can partially automate the creation of this description by generating an initial OWL-S description of Amazon's Web service using tools that transform WSDL into a partial OWL-S specification. Since the WSDL description does not contain sufficient information to form a complete OWL-S process model, we manually supplement the generated description in two steps:

(1) adding semantic descriptions of each input parameter to the generated process model, and supplying any (XSLT-based) data transformations needed to produce the corresponding grounded message parameter strings

(2) constructing a composite process model that links the various operations provided by the Web service into semantically meaningful message patterns (e.g., login before search before add-to-shopping-cart).

The resulting process model is given in Figure 1, which shows the relationships



between the various service operations represented in the resulting OWL-S process model. The client can perform three types of tasks: *search* the Amazon's data bases using author search, artist search or other types of searches; *view or modify the shopping cart* by adding new items, clearing it, or looking at its contents;

or performing the composite *shopping* process that combines the other two by first searching and then adding the product found to the shopping cart. The WSDL description of the Amazon Web service only described the operations corresponding to the leaves of this graph.

Each of the OWL-S process descriptions specifies the semantic types of the data required as input, and returned as output. For example, the input of the author search should be an instance of class *Human* that stands in a particular relationship to the book being sought (*written-by*). The use of OWL classes and properties as constraints on the instances that must be identified for particular input values is critical to the inference process; it allows for the formulation of service requests without requiring

a programmer to write special purpose code specific to each possible type of service request.

If the client has a goal to find the price of a particular book by searching, it can construct the appropriate elements of the search request by identifying the items (such as author) that are relevant, based on their relationship to the information about the book sought. Since the service to be invoked is selected because the right kind of information is described as part of the output of the service, and it describes this information as associated with database elements about books, the client can reason from that output description (ISBN of a book) back to the necessary input elements (author, title of the book whose ISBN is sought). As a consequence the client also knows what data will be returned with no need to guess from the instantiation of the *Details* data structure.

The OWL-S grounding takes care of the mapping from the concepts that describe the inputs and outputs of the processes to the inputs and outputs of the corresponding operations in the Amazon WSDL specification. As a result, while reasoning about the Web service can take advantage of the OWL logics and ontologies, the actual invocation is consistent with Amazon's requirements. Indeed, we are able to interact successfully with the Amazon Web site using the DAML-S Virtual Machine .

5 Discovery

Discovery is the process of finding Web services with a given capability. In general, discovery requires that Web services advertise their capabilities with a registry, and that requesting services query the registry for Web services with particular capabilities. The role of the registry is both to store the advertisements of capabilities, and to perform a match between the request and the advertisements. (Here we assume an infrastructure based on a centralized registry, because this is the type of infrastructure that is emerging for Web services. Nevertheless, our discussion generalizes to other architectures.)

The discovery process requires a language that can be used to encode Web service capabilities for advertisement and for requests. Furthermore, discovery requires a matching process that compares the advertisements with the requests to verify whether they describe matching capabilities.

In this section, we will describe how OWL-S may be used to express and match capabilities. Finally, we will show how OWL-S can be used to add capability matching to UDDI, the de-facto standard discovery registry for Web Services.

Representing capabilities

Capabilities of Web services correspond to the functionalities provided by Web services. Broadly speaking, there are two ways to represent functionalities. The first approach provides an extensive ontology of functions where each class in the ontology corresponds to a class of homogeneous functionalities. A simple example of an ontology which specifies a taxonomy of e-services is shown below. Using such

an ontology, Web services such as Amazon may be defined as instances of classes that represent their capabilities. Amazon, for example, may advertise itself as a Bookselling service.

The second way to represent capabilities is to provide a generic description of function in terms of the state transformation that it produces. The latter is typically used by AI planning languages such as PDDL. For example, Amazon may specify that it provides a service that requests a book title, author, address and a valid credit card number, and produces a state transition where the book is delivered to address,

```
<owl:Class rdf:ID="e_Service">
<owl:Class rdf:ID="Information_Service">
  <rdfs:subClassOf rdf:resource="e_Service"/>
</owl:Class>
<owl:Class rdf:ID="SellingService">
  <rdfs:subClassOf rdf:resource="e_Service"/>
</owl:Class>
<owl:Class rdf:ID="BookSelling">
  <rdfs:subClassOf
    rdf:resource="SellingService"/>
</owl:Class>
<owl:Class rdf:ID="AirlineTicketing">
  <rdfs:subClassOf
    rdf:resource="SellingService"/>
</owl:Class>
```

the credit card will be charged, and the book will change ownership. Despite their differences, both ways to represent capabilities use 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 the two representations of functionalities that help choose the representation by analyzing the task needs. The use of an explicit ontology of capabilities facilitates the discovery process since the matching process is reduced to subsumption between the capabilities in the ontology. On the other hand, enumerating all possible capabilities even in restricted domains for ontology encoding may be difficult. For example, consider the problem of representing translation services from a source language L_S to a target language L_T . Assuming n possible languages, there are n^2 possible types of translation services. A services taxonomy might have different classes of service for each pair of languages that could be translated, or it might just represent translation services as one general category, with explicit properties that allow particular services to describe the languages that they can translate from and translate to. This latter approach is consistent with describing the capability in terms of a state transformation. It distinguishes the translators by describing how they produce different kinds of results.

Note that describing the types of the inputs and outputs of such a service is not sufficient to distinguish capabilities. Consider, for example, a service that takes a geographic region as input and produces the names of different wines as output. This input/output couple can be used by two very different services: one that reports which wines are produced in a region, the other that reports the wines that are sold in a region.

OWL-S supports both views of the capabilities of Web services. The *Service Profile* module of OWL-S provides a high level descriptions of services as a

transformation from one state to another. To this extent, at its core, a Service Profile provides a view of the Web service as a process which requires inputs, and some precondition to be valid, and it results in outputs and some effects to become true. Furthermore, OWL-S provides a schema by which Service Profiles can be subclassed to describe a specific class of capabilities such as translation services, or wine selling services. More precisely, a Service Profile provides two types of information: the first one is a *functional description* of the Web service in terms of the transformation that the Web service produces, the second one is a set of *non-functional properties* that specify constraints on the service provided. The functional description describes both the information transformation which results in the production of outputs from a set of inputs; and the state transformation that results in the generation of the effects starting from a state where the preconditions are satisfied. Non-functional properties specify the quality of service provided by the Web service, or its security requirements, such as the type of encryption and policies that apply.

Since OWL-S synthesizes both an extensional and functional view of Web services, it provides a complete description of the services that it describes. It can take advantage of ontologies of services and products wherever they exist to the extent that they are able to represent the capabilities of a Web service. Furthermore, it can make use of transformation produced by the Web service to provide a finer grain description of the Web service or to be able to describe the effects of using a Web service even when its capability does not correspond to any functional description.

Matching capabilities

Capability matching compares the capabilities provided by any of the advertised services with the capabilities needed by the requester. The goal is to find the advertiser that produces the results required for the requester. In general it is unrealistic to expect that the capability offered will exactly match the query. For example, the requested service may be for stock quote information, and the task of the matching engine is to decide whether it can be accomplished by a service that provides financial news. The matchmaker should determine how likely it is that each capability advertisement indicates that the service will accomplish the particular function specified in the query.

A number of capability matching algorithms have been proposed for OWL-S. They use the service descriptions in the Service Profiles and the ontologies that are available to decide whether there is a match between service requests and the advertisements of the services provided. In general, they exploit one of the two views of the capabilities described above.

Matching algorithms, such as described in [1] and [2], assume the availability of ontologies of functionalities to express capabilities. Matching between the request and the available advertisements is reduced to their subsumption relation. Different degrees of match are detected depending on whether the advertisement and the request describe the same capability or whether one subsumes the other.

Other matching algorithms, such as in [3], [4], and also again [5], assume that capabilities are described by the state transformation produced by the Web service. These

matchmakers compare the state transformation described in each advertisement to the one described in the request. They perform two matches, one comparing outputs and one comparing inputs. If the output required by the requester is of a kind covered (subsumed) by the advertisement, then the inputs are checked. If the inputs specified in the request are subsumed by the input types acceptable to the service, then the service is a candidate to accomplish the requester's requirement.

In reality, there is an asymmetry between the matching of the inputs and the outputs of a Web service. Ultimately, the requester needs a Web service that produces the desired outputs. Once the Web service that provides the desired outputs has been found, the requester can either attempt to satisfy all the inputs, or use its own composition capabilities to find other Web services that can provide the desired inputs.

Relation with UDDI

UDDI (Universal Description Discovery and Integration) is an industrial initiative whose goal is to create an Internet wide network of registries of Web services. UDDI allows businesses to register their presence on the Web by specifying their points of contact both in terms of the ports used by the service to process requests and in terms of the physical contacts with people that can answer questions about the service. In addition, UDDI provides a language to specify an unbounded set of features of services that can help the process of service location and selection as well as service invocation.

UDDI enjoys the support of many prominent software and hardware companies that invested heavily in Web services. Because of this support, UDDI is becoming the de facto standard repository of Web services. Despite its role, UDDI provides a very weak discovery mechanism which does not allow the discovery of any Web service only on the bases of what problems it provides.

The main problem with UDDI is that it does not provide a capability representation language such as the OWL-S Service Profile. As a consequence, UDDI does not provide capability based search. The result is that UDDI supports the location of information about the Web service, once it is known which Web service to use, but it is impossible to locate a Web service only on the basis of what problems it solves.

OWL-S and UDDI complement each other. UDDI provides a World Wide distributed registry that is virtually an industry standard. On the other side, OWL-S provides the information required for capability matching. The OWL-S/UDDI matchmaker integrates OWL-S capability matching in the UDDI registry. This integration is based on the mapping of OWL-S Service Profiles into UDDI Web service representations shown in Fig. 2. The mapping function defines a set of specialized UDDI TModels that store OWL-S information that cannot be represented in the standard UDDI Web Service representation. (TModels are an unbounded set of properties that can be associated with a Web service specification.)

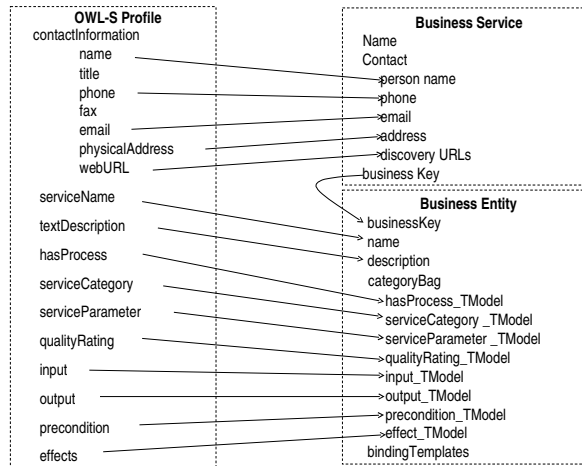


Fig. 2: OWL-S to UDDI mapping

The integrated OWL-S/UDDI provides all the functionalities provided by UDDI using exactly the same API, so that any UDDI can interact with it to retrieve information about available Web services. In addition, OWL-S/UDDI supports capability matching by taking advantage of OWL-S capability representation and the matching process proposed in . The result is a UDDI in which it is possible to search, and find, Web services by their capabilities.

6 Composition

Composition is the process of selecting, combining and executing Web services (WS) to achieve a user's objective. "Make the travel arrangements for my WWW2004 conference trip" or "Buy me an Apple iPod at the best available price" are examples of possible user objectives addressed by composition. Human beings perform manual WS composition by exploiting their cultural knowledge of what a Web service does (e.g. that www.apple.com will debit your credit card and send you an iPod), as well as information provided on the service's Web pages, in order to execute a collection of services to achieve some objective. To automate WS Composition, all this information must be encoded explicitly in an unambiguous computer interpretable form. None of the existing industrial standards for WS description encode this level of detail. Further, the descriptions they provide are not unambiguously computer interpretable and as a consequence not reliably manipulated by an automated reasoning system; hence the need for OWL-S.

Automated WS Composition is akin to both an AI planning problem and a software synthesis problem, and draws heavily on both of these areas of research . In order to perform automated WS composition, a reasoning system must order, combine and execute Web services that collectively achieve the user's objective. This involves resolving constraints between Web service inputs, outputs, preconditions and effects (IOPEs) and (typically) the outputs and effects (OEs) the user desires. For example, if one starts with an agent's goal (some desired outputs and effects), and

matches it to the outputs and effects of a Web service (modeled as a process), the result is an instantiation of the process, plus descriptions of new goals to be satisfied based on the inputs and preconditions of that process. The new goals (inputs and preconditions) then naturally match other processes (outputs and effects), so that composition arises naturally. The constraints between these inputs, outputs, preconditions and effects dictate the composition of Web services. Two types of composition problems can be distinguished: i) those that involve only information-providing services, and ii) those that involve both information-providing and world-altering services. The former requires a rich semantic representation of inputs and outputs (IO). The latter requires a like representation of IOPEs. Recall that the effects (E) are the side effects of the program (e.g., that `www.apple.com` will debit your credit card and send you an iPod). WS preconditions and (conditional) effects are not encoded in any existing industrial standard. They are encoded, in unambiguous computer-interpretable form in OWL-S. Since they supplement the information contained in WSDL, there is no grounding for these features at the WSDL level.

In addition to matching IOPEs, the automated WS Composition problem also can involve selecting from among alternative Web services that match the IOPE constraints of the composition problem. For example, there are many Web services from which a user can buy an iPod. In order to select from among alternative services, a composition engine also requires some form of service selection. This is akin to the discovery problem described in the previous section, and as argued there, requires a representation of the properties, capabilities and functioning of a Web service.

There are several different approaches to WS Composition. All characterize OWL-S processes as actions with inputs, outputs, preconditions and effects, and use planning technology to achieve WS composition. For example, the work of models processes in the same format as a STRIPS operator and plans from a sequence of Web services to achieve the user's goal. In principle the system can string together a series of actions to arrive at a novel plan for dealing with a Web service. However, the system as described is at a very early stage of development, and fails to address such basic problems as how to deal with unpredictable results of actions. also investigates the use of plan synthesis for WS Composition, though their focus is on the specific problem of planning with existing composite Web services and the work reported is preliminary.

In contrast to this approach to WS Composition, several other researchers have taken the approach of using some sort of plan script or task model that describes approximately *how to* achieve some objective. This high-level plan is expanded and refined using automated reasoning machinery. The first such system to be built was the Golog system (e.g.,). It models both world-altering and information-providing services as actions with IOPEs, uses Golog procedures (modeled as OWL-S composite processes) to represent generic procedures of approximately *how-to* perform tasks, and uses interleaving online deductive synthesis and execution to generate a sequence of Web services customized to user's preferences and constraints. Information gathering actions are executed as necessary, while world-altering actions are projected or simulated in order to enable the system to deliberate before committing to the execution of world-altering services.

In a similar spirit, several other researchers (e.g., [10]) have used the paradigm of Hierarchical Task Network (HTN) planning (e.g., [11]) to perform automated WS composition. In this paradigm, a planner is supplied with a library of standard plans, each characterized by what it is supposed to accomplish (that is, effects given preconditions). [10] uses the SHOP2 system (e.g., [12]), which is a state-of-the-art HTN planner. To solve a composition problem, SHOP2 must be given a top-level sketch of the composed plan (encoded in OWL-S as a CompositeProcess). However, many of the steps in the plan are described in a high-level vocabulary (analogous to the OWL-S control constructs) that allows multiple alternative subplans to carry out those steps. The system searches through ways of combining those subplans in order to arrive at an overall plan. Central to the SHOP2 approach to planning with Web Services is the exploitation of the sharp distinction between information-providing and world-altering services in the planning process, given that the information provided by services is often critical to finding a plan. When mapping from a set of OWL-S service descriptions to a SHOP2 domain, information-gathering services are detected and encoded so as to be executed at planning time, rather than at run time (as so-called “book-keeping” operators, or, in current work, as SHOP2 evaluated preconditions). [10], also execute information-gathering services at plan time to reduce the search space for plans and to reduce non-determinism.

HTN planning has also been used in [13] to compose Web services in the travel domain and in the organization of a B2B supply chain. The basic idea explored in this work is that Web services expand their own capabilities through collaboration. Consistently with the work presented above, especially [14] and [15], during the planning process, outputs and preconditions are satisfied either directly using an action that the Web service can perform or by asking other Web services to do something that satisfies that output or precondition. Locating appropriate Web services can be done using the OWL-S/UDDI matchmaker as discussed in Section 5.

There are many systems that deal with the restricted problem of composing services without consideration of preconditions and effects (PEs). Included in these is the work of [16] that augments BPEL4WS, a popular business-process language [17], with a composition module. When the BPEL4WS process requires a certain input, described as an XML data type, their system searches for a WS that can translate from available formats to the desired format. For example, if the process declares a need for a complex type containing a date in US format, and a known service supplies a data type identical except that the date is in UK format, the system searches for a translation service that can perform the desired data transformation. If necessary, it breaks the transformation process into substeps and recursively searches for methods to accomplish the substeps. A similar approach is integrated with an end-user interactive composition system, STEER described in [18]. These approaches represent prototype solutions to an important subtask of service composition, namely, *data-transfer interoperation*. For it to work, it is necessary for process descriptions to include rich, computer-interpretable descriptions of the inputs and outputs of a process — the IO half of IOPEs.

While this early work is promising, we are still some distance from the goal of automated WS composition. We have argued that we need rich, representations of Web services in a language with a well-defined semantics, to enable automated WS composition. Specifically, we require rich, declarative descriptions of Web service IOPEs to determine a composition, and we require rich representations of the

properties, capabilities and functioning of services to enable WS selection during the composition process. We have achieved both these requirements in great measure with OWL-S. In contrast, current industrial standards for WS description only describe WS inputs and outputs and they do so in a language that is not richly expressive and is without a well-defined semantics.

We also require rich declarative representations of composite processes (existing compositions of Web services, such as Amazon's workflow) so that we can exploit them in our WS composition tasks. (Many of the existing WS composition technologies only compose atomic processes.) We have addressed the problem of describing composite processes in OWL-S, but we believe the solution can be improved upon by appealing to a language that is more expressive than OWL, leveraging emerging industrial process modeling standards. To realize the goal of automated WS composition, we also require further advances in automated reasoning/planning technology for WS. A final barrier to the goal of automated WS composition is the need for wide-spread adoption of OWL-S WS descriptions.

Despite the need for further work, the accomplishments of OWL-S and associated composition technologies provide immediate value-added. With existing technology we can perform automated composition of information-gathering services. It has also been demonstrated that we can augment existing industrial WS choreography and orchestration tools with composition technology for data-transfer interoperation and for run-time binding of Web services. These systems enable manual composition of WSs. We can augment this with some semantic integration of the data sources. Finally, as demonstrated, we can currently perform automated WS composition of both information-gathering and atomic world-altering services under controlled conditions. Automated WS Composition is at the heart of seamless interoperation among Web services. With adoption of approaches to WS description such as OWL-S and advances in planning-related technologies, we believe that broad-scale automated WS composition is well within reach.

7 Related Work

Throughout this paper we have identified related work that exploits OWL-S (or DAML-S, the name by which earlier versions of OWL-S were known). Here we briefly note other work on Semantic Web Services that does not use DAML-S or OWL-S to describe Web services.

Most of the work on discovery of Web services using the Semantic Web has been based on OWL-S. Nevertheless, other work on discovery does not assume OWL-S, most notably which bases Web service descriptions on the MIT Process Handbook . In this work, the matching process is based on the workflow description of the process model of a Web service rather than an abstract representation such as the OWL-S Profile. The retrieval mechanism maps the request against all the process models advertised by available services until only the process models that match the request are retrieved.

The matching process allows the requester to ask for Web services that "do X before Y". In other words, the requester can constrain not only the type of process it performs and the results that it achieves, but also the way in which a service is

achieved. Implicitly, it also assumes that the requester and the provider have a shared and intimate knowledge on how processes are performed. In turn, this assumes that the provider and the requester should share ontologies such as the MIT Process Handbook. While OWL-S does not make such strong assumptions on the ontologies needed for discovery, when those assumptions are known to hold, results similar to those obtained in can be obtained by using the matching processes suggested for OWL-S, by first selecting Web services with a given capability and then selecting those services whose process model satisfies the temporal constraint.

In the area of WS Composition, most of the early work has exploited OWL-S. More recently, researchers from the planning community (e.g.,) have begun to examine the WS Composition problem; however, most have not explicitly addressed the problem of how to describe Web services, beyond modeling service IOPEs as actions in first-order logic, propositional logic or PDDL .

8 Summary

Our objective in this paper has been to show how OWL-S can be put to use in the near-term, in the context of emerging Web service standards such as WSDL, UDDI and BPEL. We have explained some of the basics of OWL-S, and the techniques by which it can be used in conjunction with these standards; and we have given an overview of projects that have employed OWL-S in combination with one or more of them.

We have discussed the benefits of the richer service descriptions supported by OWL-S, focusing primarily on the descriptions of inputs, outputs, preconditions, and effects of services. In the area of enactment, OWL-S supports the specification of composite processes, and allows for flexible, robust invocation and interoperation between service clients and providers. In addition, OWL-S grounding mechanisms allow process descriptions and enactment procedures to be used in conjunction with WSDL. In the area of discovery, OWL-S allows service registries and matchmaking algorithms to take advantage of two distinct styles of ontology-based characterization of services, whose use may be integrated with UDDI. In the area of service composition, a variety of approaches exist to reason about OWL-S IOPEs, in support of manual, semi-automated, and, under controlled conditions, automated composition of both information-gathering and world-altering services.

In conclusion, OWL-S can help to enable fuller automation and dynamism in many aspects of Web service provision and use, support the construction of powerful tools and methodologies, and promote the use of semantically well-founded reasoning about services.

References

- [1] J. L. Ambite (Ed.). *Proceedings of the ICAPS2003 Workshop on Planning for Web Services*, 2003.
- [2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Golland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte (Editor), Ivana

- Trickovic, Sanjiva Weerawarana. Business Process Execution Language for Web Services, Version 1.1, 2003. At <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>.
- [3] A. Bernstein and M. Klein. High Precision Service Retrieval. In *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, Sardegnna, 2002.
 - [4] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey and Farouk Toumani. Request Rewriting-Based Web Service Discovery. In *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp 335-350, October 2003.
 - [5] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. At <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
 - [6] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P.F. Patel-Schneider, L. A. Stein. Web Ontology Language (OWL) W3C Reference version 1.0, 18 August 2003. At <http://www.w3.org/TR/2002/WD-owl-ref-20021112>.
 - [7] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan and Katia Sycara. Security For DAML Web Services: Annotation and Matchmaking. In *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp. 335-350, October 2003.
 - [8] Tommaso Di Noia, Eugenio Di Sciacio, Francesco M. Donini and Marina Mongiello. Semantic Matchmaking in a P-2-P Electronic Marketplace. SAC 2003, pp. 582-586, 2003.
 - [9] R. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2, pp. 189-208, 1971.
 - [10] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet. OWL Rules Language, Draft version . Technical report, 29 October 2003
 - [11] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331-339, ACM, 2003.
 - [12] T. W. Malone, K. Crowston, B. P. Jintae Lee, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, 45(3):425--443, March, 1997.
 - [13] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. . In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, pp. 227--241, 2003
 - [14] David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, Sheila McIlraith. Describing Web Services using OWL-S and WSDL. October 2003. At <http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html>
 - [15] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin. Ontology-Enabled Pervasive Computing Applications. In *IEEE Intelligent Systems*, 18(10):68-72, 2003.
 - [16] D. McDermott. Estimated-Regression Planning for Interaction with Web Services. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, pp. 204—211, 2002.
 - [17] D McDermott. The Planning Domain Definition Language Manual. *Yale Computer Science Report 1165* (CVC Report 980003), 1998.
 - [18] D. McDermott and D. Dou . Representing Disjunction and Quantifiers in RDF. *Proceedings of the First International Semantic Web Conference (ISWC2002)*, 2002.
 - [19] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. World Wide Web Consortium (W3C) Candidate Recommendation. August 18, 2003. At <http://www.w3.org/TR/owl-features/>
 - [20] S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pp. 482-493, 2002.

- [21] S. McIlraith., T.C. Son and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46--53, March/April, 2001.
- [22] S. McIlraith and R. Fadel. Planning with Complex Actions. In *Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning (NMR2002)*, pages 356-364, April, 2002.
- [23] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp.968—973, 1999.
- [24] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman 2003 SHOP2: An HTN Planning System. To appear, *Journal Artificial Intelligence Research*.
- [25] OWL-S Coalition. OWL-S 1.0 Release. At <http://www.daml.org/services/owl-s/1.0/>
- [26] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The DAML-S Virtual Machine. In *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pp 335-350, October 2003.
- [27] M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura. Toward a Semantic Choreography of Web services: from WSDL to DAML-S. In *Proceedings of ICWS03*, 2003.
- [28] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference (ISWC2002)*, 2002.
- [29] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the Semantic Web in UDDI. In *Proceedings of E-Services and the Semantic Web (ESSW02)*, 2002.
- [30] Massimo Paolucci, Katia Sycara, and Takahiro Kawamura. Delivering Semantic Web Services. In *Proceedings of the Twelfth World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003, pp 111- 118.
- [31] The Rule Markup Initiative. At <http://www.dfki.uni-kl.de/ruleml/>.
- [32] The Universal Description, Discovery and Integration (UDDI) protocol. Version 3, 2003. At <http://www.uddi.org/>
- [33] Web Services Choreography Working Group. At <http://www.w3.org/2002/ws/chor/>
- [34] Web Services Description Working Group. At <http://www.w3.org/2002/ws/desc/>
- [35] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, 2003.