Chapter 1

OWL-S AND AGENT-BASED SYSTEMS

David Martin, Mark Burstein, Sheila McIlraith, Massimo Paolucci, Katia Sycara
Members of the OWL-S Coalition

Abstract: Over the last decade, research in agent-based systems (ABS) has spawned a multi-faceted field, addressing a broad range of challenges and generating a varied array of technical approaches. Web service technologies, in contrast, have arisen in a more incremental fashion, with more modest aims — although the vision statements associated with Web services sometimes overlap significantly with those of ABS. Work on *Semantic* Web services aims to provide richer specifications of 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. This chapter provides an overview of OWL-S, a Semantic Web services ontology, and discusses its connections with work on agent-based systems. We argue that OWL-S takes some significant, although limited, steps towards a foundation for the deployment of agent technologies on the Web.

1. INTRODUCTION

Over the last decade, research in agent-based systems (ABS) has spawned a multi-faceted field, addressing a broad range of challenges and generating a varied array of technical approaches. ABS research topics may be divided, roughly, into those having to do with individual agents, and those having to do with multiagent systems (MAS). The agenda of ABS has been ambitious and, in many respects, visionary. ABS researchers have sought to endow individual agents with characteristics such as autonomy, proactivity, and cooperativeness. In the area of MAS, the focal areas have included agent communication languages, extended conversations between agents, shared knowledge, shared activities, interoperation frameworks, and various kinds of middle-agents. (Excellent overviews and reference lists for ABS can be

found at `http://agents.umbc.edu/introduction` and `http://www.aaai.org/AITopics/html/agents.html`.)

In pursuing this agenda, most ABS approaches have emphasized various forms of reasoning employing rich representations of knowledge — about agent capabilities, tasks, beliefs, commitments, communications, and so forth. In this sense, ABS have always relied on "semantically rich" representations and techniques.

Web service (WS) technologies, in contrast, have arisen in a more incremental fashion, with more modest aims — although the vision statements associated with Web services sometimes overlap significantly with those of ABS. Web Services Description Language (WSDL) [11], 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) [2] has the most prominent status. With respect to registering Web services for purposes of advertising and discovery, Universal Description, Discovery and Integration (UDDI) [42] 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, it can provide a better foundation for a broad range of activities across the service lifecycle. For example, richer semantics can support greater automation of service selection and invocation (thus reducing the burden on service developers), 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* (SWS) [29]. In particular, the authors of this chapter, members of the OWL-S Coalition, are involved in the development of the Ontology Web Language for Services (OWL-S) [34], which seeks to provide the building blocks for encoding rich semantic service descriptions, in a way that builds naturally upon OWL [27], the

Semantic Web [3] language undergoing standardization at the World Wide Web Consortium (W3C).

We note that several of our references here, and in several other chapters in this book, refer to DAML-S, (DARPA Agent Markup Language for Services) the name by which earlier versions of OWL-S were known. As of version 1.0 (which has been released a short while prior to this writing), the name was changed to OWL-S, so as to reflect the change in the underlying formalism, from DAML+OIL to OWL. (DAML+OIL — DARPA Agent Markup Language + Ontology Inference Language — was the predecessor of OWL.)

1.1 Relating Agents and Services

How can one begin to characterize the relationship between agents and Web services?

The vision of ABS encompasses a broad scope of challenges and approaches to distributed task handling by relatively autonomous components. The approach taken by commercial Web services, in contrast, is necessarily more incremental, as it is tied to near-term products and goals. The ambitions of work on Semantic Web services lies somewhere in between.

ABS technology arose largely independently of the World Wide Web. From a historical perspective, one may view the work on Web services and (to a greater degree) Semantic Web services as efforts to bring aspects of agent-based approaches onto the Web (although the work has generally not been done with this as an explicitly stated goal). Generally speaking, this has been accomplished in relatively modest ways to date. In this chapter we discuss some of the ways in which this has been accomplished.

From an architectural perspective, one can identify three possible views of the relationship between agents and services:

(1) *Agents use services*. In this view, there is no attempt to envision Web services as belonging to the realm of agents. Individual services can remain relatively simple — providers of discrete capabilities accessed via fixed message exchange patterns, exempt from exhibiting proactivity, autonomy, or other more sophisticated attributes of agents. Indeed, in this view, some of the hardest challenges associated with Web services, such as automated general service composition, could ultimately be relegated to the realm of agents, and left out of the scope of Web service standards. In general, this view creates no requirements for service infrastructure to support services that take on the more sophisticated attributes of agenthood.

(2) *Services are agents*, although currently of a limited kind. In this view, which is the most ambitious regarding the future of Web service

technology, individual services will ultimately be free to display the autonomy, proactivity, persistence, etc. that define the notion of software agenthood, and collections of services will interact with the flexible collaboration that defines the essence of MAS. Currently, however, it may be seen that most work on services falls short of this vision. In particular, individual services are for the most part conceived as reactive, short-lived, and intended to engage only two parties in a provider/requester style of use. Although some work (both in WS and in SWS) is aiming to break out of these limitations, it is clear that there's a long ways to go yet.

(3) *Agents are composed of, deployed as, and dynamically extended by services.* This view, which holds that agents are built up from Web services as building blocks [8], can be viewed as a middle ground between (1) and (2). It allows for a notion of service that's more limited than that of (2), but which exists within a more extensive conceptual framework than is required for (1). This perspective draws on work on that combines behavior-based robotics and reactive planning (e.g., Behavior-Oriented Design [7]), and is particularly well-suited to scenarios in which services embody devices that may be viewed as sensors or effectors of the world.

We note that there may well be other views of the agent/service relationship, and it isn't entirely clear which of these three has the greatest applicability; we expect that future developments will make this clear.

In this chapter we have two primary aims: to provide an overview of OWL-S, and to show some of its more significant connections with work on software agents and multiagent systems. Although there is clearly no cut-and-dried characterization of these connections, nevertheless it is useful to trace some of the central similarities, differences, and lines of evolution. We emphasize that this is not meant to be a comprehensive survey: ABS is an enormous field, and we can only hope to touch on a small selection of the relevant work. As a unifying theme, we argue that OWL-S takes some essential, although limited, steps towards a foundation for the deployment of agent technologies on the Web.

In the next four sections, we briefly present OWL-S, beginning with an overview, and turning then to its profile, process, and grounding ontologies. Following that, we discuss its relationship to work on ABS, organized under the topics of *discovery* (including capabilities declarations, advertising, and matchmaking), *agent communication languages* (including conversational protocols); and *service composition*.

2. OVERVIEW OF OWL-S

OWL-S is an OWL ontology that may be used to specify semantically rich characterizations of services on the Web. OWL-S is organized into four parts. The *profile* describes capabilities and discriminating features of Web services for purposes of advertising and matchmaking. The *process model* provides a description of the structure of activities involved in providing the service, from which service requesters can derive information about service invocation and interaction patterns. The *grounding* is a description of how abstract information exchanges described in the process model are mapped onto actual concrete messages that flow between the provider and the requester. Finally, the *service* itself provides a means of bundling together instances of the top-level profile, process, and grounding classes that are meant to be used together. (Because the service concept introduces no new details apart from the bundling of these other elements, we do not give it a separate section in the organization of this chapter.)

OWL-S complements industry efforts such as SOAP, WSDL and BPEL4WS. It builds upon these efforts by adding rich typing and class information that can be used to describe and constrain the range of Web service capabilities more effectively than can be done with XML data types. Further, in the process model, it captures not only the control flow and data flow of Web services, but also their prerequisites and side effects (preconditions and effects) in the world. OWL-S' basis in OWL enables the grouping of like services and like data types into taxonomic hierarchies, together with definitions of the relationships and constraints between classes and their instances. The well-defined semantics enables formal automated manipulation of these structures, with known outcomes, thus providing a foundation for automation of a variety of Web service operations, such as discovery, matchmaking, interoperation, composition, enactment, monitoring, and recovery.

3. DESCRIBING CAPABILITIES WITH OWL-S

The representation of capabilities is supported by the *profile* module of OWL-S, which provides a high-level view emphasizing the functionality of the service; in other words, what the service *does*, rather than *how* it accomplishes its tasks. (The latter information, as discussed in Section 4, is provided by the OWL-S process model). Capabilities descriptions can have widely varying aspects. Capabilities may be described in terms of service categories, such as stock brokering, or functional transformations, such as the transformation from a ticker symbol to the related stock quote.

Furthermore, a particular type of service may have many aspects. For instance, two services may act as stock brokers, but the way they perform their task may be very different in terms of delay on the market, precision of the report, cost and so on. The OWL-S profile provides a language to express the different aspects of service capabilities. Specifically, OWL-S profiles support three kinds of descriptions of capabilities: a *hierarchical classification*, a *functional description*, and a set of *non-functional parameters* that can be used to specify features of the service that are not captured by the other two descriptions. In the rest of this section we discuss these three descriptions in more detail.

The *hierarchical classification* of a capability exploits the fact that any service profile is a concept expressed in OWL. It is therefore possible to construct ontologies of profiles that group together the common features of services performing similar functions. For example, as shown in Figure 1, it is possible to define the class *FeeBasedService* having properties needed to characterize the manner in which payment is made for the service (e.g. property *paymentMethodAccepted*). This class, in turn, could have *PhysicalProductRetail* as one of its subclasses, with properties that are characteristic of online retail sources, such as *product* and *deliveryRegion*. Consistent with the semantics of OWL (and with object-oriented practice in general), each subclass will declare properties that are appropriate to its level of specificity in the class hierarchy. The use of hierarchies of profiles allows service providers to specify precisely and economically what kinds of capabilities they provide, and allows service requesters to use the same framework to precisely specify what kinds of capabilities they seek.

The *functional view* of the capabilities of a service describes the information transformation as well as state transformation performed by that service. At the information level, the service requires a set of inputs and produces a set of outputs; at the state description level, a service requires a set of preconditions to be satisfied and produces a set of effects. The functional description of the profile captures essentially the same information about inputs, outputs, preconditions, and effects (IOPEs), or a subset of the information, as the process model, as discussed in Section 4 below.

The last aspect of capability representations in OWL-S is provided by *non-functional properties* that are used to specify additional information about the Web service, such as security restrictions and quality of service information. The use of non-functional properties essentially recognizes that while two services may provide the same capability, in the sense that they compute the same function, the way they achieve this result may be very different and the resulting services may be qualitatively very different. For

```
<owl:Class rdf:ID="FeeBasedService">
  <rdfs:subClassOf rdf:resource="&profile;#Profile" />
</owl:Class>
<owl:ObjectProperty rdf:ID="paymentMethodAccepted">
  <rdfs:domain rdf:resource="#FeeBasedService"/>
  <rdfs:range rdf:resource="&business;#PaymentMethod"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="PhysicalProductRetail">
  <rdfs:subClassOf rdf:resource="#FeeBasedService"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="product">
  <rdfs:domain rdf:resource="#PhysicalProductRetail"/>
  <rdfs:range rdf:resource="&unspsc;#Product"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="deliveryRegion">
  <rdfs:domain rdf:resource="#PhysicalProductRetail"/>
  <rdfs:range rdf:resource="&geography;#Region"/>
</owl:ObjectProperty>

<PhysicalProductRetail rdf:ID="Books4Sale">
  <paymentMethodAccepted rdf:resource="&business;#Mastercard"/>
  <paymentMethodAccepted rdf:resource="&business;#Visa"/>
  <paymentMethodAccepted rdf:resource="&business;#MoneyOrder"/>
  <product rdf:resource="&unspsc;#Books"/>
  <deliveryRegion rdf:resource="&geography;#NorthAmerica"/>
</PhysicalProductRetail>
```

Figure 1: Simple profile hierarchy and instance representing an online bookseller.

example, two services may provide tax advice, but one may be specialized on the US tax code, while the other may be specialized on the Italian tax code. The two services compute the same function, namely tax advice, but they are not the replacement of one another.

4. DESCRIBING ACTIVITIES WITH OWL-S

The OWL-S *process model* describes *how the service works*. For a one-step (atomic) service, it gives the complete information (about inputs and outputs) that's needed to interact with the service. For a multi-step (composite) service, it describes the control flow and data flow of the program that enacts the service. The process model has been designed for use by service requesters in connection with service selection, invocation, interoperation, composition, and monitoring, and by service tools for service simulation and verification. The OWL-S process model includes a number of elements that are typical of workflow languages, combining a process modeling language with both an AI-inspired action language and a language

```
<process:AtomicProcess rdf:ID="LookupBook">
  <process:hasInput>
    <process:Input rdf:ID="InTitle">
      <process:parameterType rdf:resource="dc:Title"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:UnconditionalOutput
      rdf:ID="OutISBN">
      <process:parameterType rdf:resource="dc:Identifier"/>
    </process:UnconditionalOutput >
  <process:hasOutput>
</process:AtomicProcess>

<process:AtomicProcess rdf:ID="BuyBook">
  <process:hasInput>
    <process:Input rdf:ID="InISBN">
      <process:parameterType rdf:resource="dc:Identifier"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:UnconditionalOutput
      rdf:ID="OutReceipt">
      <process:parameterType rdf:resource="business:E-receipt"/>
    </process:UnconditionalOutput>
  <process:hasOutput>
</process:AtomicProcess>
```

Figure 2: Partial functional description of two (simplified) atomic processes

for describing classes and their inter-relationships. Further, the process model has a well-defined semantics.

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

Inputs, outputs, preconditions, and effects specified in a process model are complete, whereas in a profile they may be partial (that is, a selected subset that is most useful for purposes of advertising and discovery).

OWL-S' process ontology is subdivided into three process types: *atomic*, *simple*, and *composite* processes.

Atomic processes are the units of invocation; that is, an atomic process, somewhat similarly to a programming language procedure, can be called by transmitting an invocation message (which carries its inputs) to the process, and its results returned in a response message. Thus, an atomic process executes in a single, non-interruptible step. The essence of an atomic process

```
<process: CompositeProcess rdf:ID="FindAndBuyBook">
   ...
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:Perform rdf:ID="FindBook1”>
          <process:process rdf:resource=”#LookupBook”>
          <process:hasBinding>
            <process:theParam rdf:resource=”#InTitle”>
            <process:valueForm rdf:parsetype=”Literal”>
              <valueOf>
                <theVar rdf:resource=”#InTitle”>
                <fromProcess rdf:resource=”#FindAndBuyBook”>
              </valueOf>
            </process:valueForm>
          </process:hasBinding>
        </process:Perform>
        <process:Perform rdf:ID="BuyBook1"/>
          <process:hasBinding>
            <process:Binding>
              <process:theParam rdf:resource=”#InISBN”>
              <process:valueForm rdf:parsetype=”#Literal”>
                <valueOf>
                  <theVar rdf:resource=”#OutISBN”>
                  <fromProcess rdf:resource=”#LookupBook”>
                </valueOf>
              </process:valueForm>
            </process:Binding>
          </process:hasBinding>
        </process:Perform>
      </process:components>
    </process:Sequence>
  ...
</process:CompositeProcess>
```

Figure 3: Elements of a composite process.

is in its IOPEs; thus, it provides the same kind of functional description as the service's profile.

Figure 2 shows two simple partial examples of atomic processes, each with a single input and a single output. `LookupBook` takes a book title as input. The type of the input named `InTitle` is `dc:Title,` and the type of the output named `OutISBN` is `dc:Identifier,` both types from the Dublin Core ontology [17]. The returned value is an encoding of the International Standard Book Number (ISBN) for the book with the given title. Similarly, `BuyBook` takes an input of type `dc:Identifier,` and returns an output named `OutReceipt`, representing the purchase of the book.

In these examples, for lack of space, we omit a number of details, such as the payment and shipping information that would be needed to complete such a transaction, handling of unsuccessful outcomes (e.g., title unknown or out-of-stock), and preconditions and effects. A typical precondition for such a service might say, "The buyer must have a current account", and a typical

effect might say, "The buyer's credit card is charged for the price of the book". Outputs and effects may have conditions associated with them. For example, the output(s) and effect(s) for the condition "book-in-stock" could be described distinctly from those for the condition "book-out-of-stock".

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

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

Figure 3 shows a fragment of a composite process, which expresses that the composite consists of sequentially invoking ("performing") the atomic process `LookupBook`, and then invoking the atomic process `BuyBook`. The binding within the second Perform construct indicates that the output from `LookupBook` flows into the input of `BuyBook`. A composite process like this is enacted by the service client, with the execution of `LookupBook` and `BuyBook` handled by the service provider.

5. DECLARING INVOCATION DETAILS WITH OWL-S

The OWL-S *grounding* tells *how the service is used*; that is, it specifies the details of how a computer program or agent can access a service. Typically, a grounding will specify some well known communications protocol, service-specific details such as port numbers used in contacting the service, and an unambiguous means of exchanging data elements of the types required and produced by the service [22].

The default grounding approach provided by OWL-S relies on specification mechanisms already provided by WSDL, while at the same time exploiting the richer descriptions available through the use of the OWL language [10]. In essence, it establishes a correspondence between each atomic process and a WSDL operation, and between each atomic process parameter and a WSDL message part.

In summary, an OWL-S service is described by the three types of information presented above: one or more *profiles* tell what the service does (in support of advertising, discovery, matchmaking, and so forth), a single *process model* tells how the service works (in support of service invocation,

interoperation, composition, and related activities), and one or more *groundings* tell how to access the service (in terms of message formats and other communications details). Whereas the grounding is concerned with the *concrete* details of message syntax and transport, the profiles and process model are more *abstract* specifications that can be used to support classification, planning, and other forms of reasoning about services, thus enabling fuller automation of service-related activities.

6. OWL-S AND DISCOVERY

Autonomous agents and Semantic Web services can be used to realize a distributed computation scheme in which problems are solved through interaction with other agents or Web services. Such a distributed scheme requires agents and Web services to *discover* partners that can contribute to the collaborative effort. The problem of discovering partners is essentially equivalent for agents and Semantic Web services[1]: given a goal, the discovery process should automatically locate those agents[2] that can achieve that goal.

The process of discovery can be roughly divided into three stages: first, agents advertise their capabilities with a registry[3] such as UDDI. Second, an agent in need of a service requests, from the registry, references to providers of the capabilities needed by the agent. Third, the registry reports back to the requester one or more references to providers whose advertisements match the request.

The success of the discovery process hinges on two requirements. First, a language is needed that allows service-providing agents to effectively describe their capabilities for purposes of advertising them to the broader community. This same language should also allow service-seeking agents to formulate requests for the capabilities they need from service-providers. Second, an advertisement/request matching technology is needed that detects when the requested service is equivalent to the advertised service, even if the two descriptions are superficially very different.

As described in Section 3 above, OWL-S directly addresses the first requirement of discovery by providing OWL concepts for the description of

[1] The stress on *Semantic* Web services here is justified by the fact that, in the most conservative view, commercial Web services standards are meant to help programmers, rather than programs, to discover and "glue" multiple agents together.

[2] Since discovery and its requirements are equivalent for Web services and agents, in this section we do not make any distinction between agents and Web services, and we refer to both as agents.

[3] In our discussion we describe the most common discovery mechanism used by Web services. Depending on the overall system architecture other discovery mechanisms are possible.

what agents do and the tasks that they accomplish. In the following section we show how OWL-S can be used to address the second requirement by defining algorithms that exploit the logical relations between the advertisements and the requests, abstracting away from superficial syntactic differences. We will then explore the relation between the view of discovery proposed by OWL-S and the views that have arisen from the fields of agent-based systems and Web services.

6.1 Matching Capabilities

The discovery process performs the matching of requests and advertisements to identify the agents that can perform a desired task. The problem of capability matching is that it is unrealistic to expect that the advertisement and the request describe exactly the same capability. Rather, the challenge of the matching process is to abstract away from the syntactic differences between the advertisements so as to extract the semantic similarities between the advertisement and the request, and verify whether the advertised capability is close enough to the capability requested.

To address this challenge, OWL-S based matching algorithms exploit the semantics of the representation of the advertisements and requests; furthermore, they provide a flexible matching mechanism that recognizes the degree of matching between the advertisement and the request. Exploiting the semantics of descriptions and defining a flexible matching are closely related issues. Indeed by analyzing the semantic relation between the advertisement and the request it is possible to derive a scoring function between the two concepts.

A number of capability matching algorithms have been proposed for OWL-S, which use different types of information provided by the service profile and the available ontologies to match between service requests and advertisements. Some matching algorithms, such as [9] and [18] rely on the subsumption computation provided by OWL inference engines to infer the relation between the advertisement and the request. These matching algorithms rely on the ability to construct taxonomies of service profiles that correspond to the different types of services that are present. Other matching algorithms, such as in [5], [14], and [37], assume that matches are determined exclusively by the relations between inputs and outputs in the advertisement and in the request. Essentially, these matching algorithms 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 then subsumed by the input types acceptable to the service, then the service is a candidate to accomplish the requester's requirement.

Despite the different attempts to provide a matching algorithm for agent capabilities, many challenges are still open. First, the two methods of matching capabilities are not guaranteed to converge to the same set of agents. This is partly because the functional and taxonomical representations convey capability information in very different ways that may not lead to compatible representations for the same capability. The second problem is that no matching has been proposed that extends to preconditions and effects. Finally, there is no predefined set of non-functional parameters and by and large each type of parameter may require a specialized matching process as shown by attempts to match on security requirements [13].

6.2 Relation with Agent technology

The OWL-S service profile is grounded in the research on agent discovery in open multiagent systems, and has been influenced by systems such as LARKS [40], OAA [21] and InfoSleuth [33].

In this areas, a primary contribution of the multiagent literature has been defining the goal of discovery in multiagent systems as the problem of finding the agent(s) that can perform a given task. Ultimately, this problem requires a goal directed search where the agent is located on the bases of the tasks that it performs, or in other words its capabilities, rather than on the bases of incidental properties such as name, port type or keywords attached to the agent. As a consequence, research in multiagent systems provided schemas for representation of capabilities that are reflected in OWL-S. Specifically, LARKS and OAA provided different perspectives on the goal directed view of the capabilities of agents, while InfoSleuth provided a way to represent classifications of agent functionalities.

The second contribution of research in multiagent systems has been to reveal the importance of the semantic matching of capabilities. Essentially, they realized that capability descriptions and capability requests are syntactically going to be very different. Therefore, any attempt to match that does not include an abstraction to semantics is bound to fail. The key insight provided by the multiagent community has been to claim the need for ontologies during the matching process that would support a flexible matching between request and advertisement with a measure of the degree of match.

6.3 Relation with UDDI and WS technology

UDDI is a World Wide initiative, lead by the OASIS Consortium[4], to develop a standard specification for industrial strength 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 who 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 wide support of many prominent software and hardware companies that have invested heavily in Web services. Because of this support, UDDI is becoming the de facto standard repository of Web services.

The main limitation of 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.

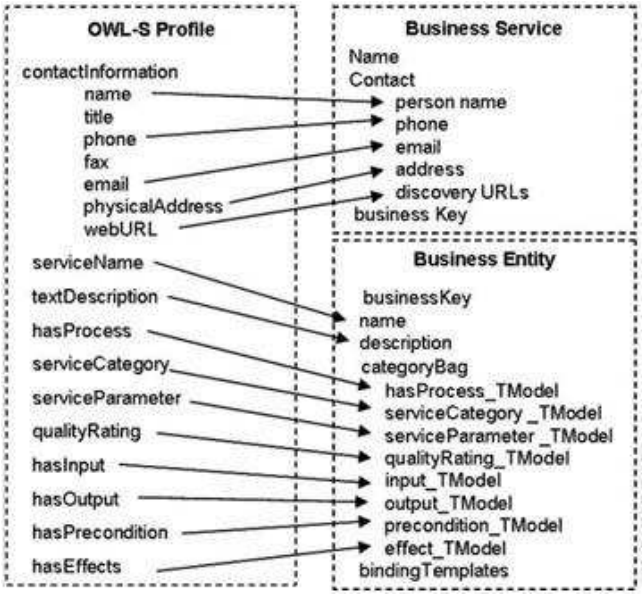

Figure 4. The OWL-S to UDDI mapping

[4] See http://www.oasis-open.org

OWL-S and UDDI complement each other. UDDI provides an Internet-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 [38] integrates OWL-S capability matching into the UDDI registry. This integration is based on the mapping of OWL-S service profiles to UDDI representations shown in Figure 4.

The integrated OWL-S/UDDI registry 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 [36]. The result is a UDDI in which it is possible to search for, and find, Web services by their capabilities.

7. OWL-S AND AGENT COMMUNICATION LANGUAGES

OWL-S explicitly supports the description of services as classes of activities, so that agents can reason about the possible benefit of using them, determine the content of the messages necessary to invoke them, and interpret responses from them. This is substantially different than the rationale behind *agent communication languages* (ACLs), such as the Knowledge Query and Manipulation Language (KQML) or the Foundation for Intelligent Physcal Agents (FIPA) ACL, developed during the 1990's.

ACLs like KQML and FIPA were designed primarily to provide a uniform syntax and semantics for messages with arbitrary content, passed between software agent peers. They divided messages into several layers, and provided a specific syntax and semantics only for the outer layer, which supplied information *about* the message, its handlers (sender and recipient), its context (as it might relate to a prior message or sequence) and the language and ontologies used for the inner *content layer* which could be in any format or language. The semantic content that was standardized in these languages consisted of a set of *performative types*, or message types, representing the conversational attitude of the sender toward the content. The term performative comes from Speech Act theory. ACL performatives include 'request', 'tell', 'ask', 'accept', 'reject', 'deny', etc.

Just as ACLs were explicitly a model for messages, providing a standard meta- model so that they could be arranged in conversations, WSDL is a way of representing messages in the Web services world. WSDL models services as bundles of message patterns, grouped into simple sequence arrangements, such as input-output pairs. In contrast, the OWL-S process

ontology is a framework for describing more abstractly the service activities themselves, and likely sequences of such activities. OWL-S descriptions of the inputs and outputs of individual atomic activities characterize the information conveyed in the underlying WSDL input and output messages. The OWL-S grounding model translates the inputs and outputs of OWL-S service descriptions from OWL into the XML elements of corresponding WSDL messages. But it can just as properly be used to translate that information into a KQML or FIPA message format for communications with agents that provided services using an ACL message transport mechanism. UMBC's implementation of the Trading Agent Game [46] provides an example of this way of using OWL-S.

OWL-S groundings used with KQML or FIPA must relate atomic processes to ACL message patterns with specific performatives and perhaps even specific content forms. The performatives referenced in these messages patterns depend on the type of service provided, and the kind of action triggered by the message. For example, an informational service would have its inputs grounded into an ASK message, while its outputs (the reply) could be an INFORM or TELL (depending on the ACL dialect). A service that resulted in a purchase would have inputs grounded in a REQUEST message pattern, while its outputs might be (conditionally) an ACCEPT or REJECT message pattern. The content pattern of the message would in each case have to contain information about the specific service (such as its type), and serialized descriptions of the input or output parameters of the service. Each reply would also be accompanied by an IN-REPLY-TO property referencing the corresponding input message requesting the service action.

Just as performatives are implicit in the activity model of OWL-S processes, the information about message sequencing that is present in ACLs is represented in OWL-S by inputs and outputs of processes, and activity sequences, the latter modeled using the control constructs for forming OWL-S composite processes. ACLs use the message field IN-REPLY-TO to specify a conversational token that can be used to mark messages as responses to specific other messages, since messages can arrive asynchronously. OWL-S simply describes the semantics of sets of inputs and outputs of a process, which must be translated into messages.

In summary, OWL-S abstracts away the details of the message-level interactions of both ACLs and web service message languages like WSDL. Instead, it focuses on characterizing the content and workflow of interactions with services so that client systems can perform the reasoning necessary to interoperate with them automatically.

8. OWL-S AND TASK COMPOSITION

Web service composition (WSC) is the process of selecting, combining and executing Web services 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 user objectives that might be addressed by such composition. Human beings perform manual WSC 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* WSC, 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.

8.1 The Need for OWL-S

Automated WSC is akin to both an AI planning problem and a software synthesis problem, and draws heavily on both of these areas of research [28]. In order to perform automated WSC, 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). Web service preconditions and (conditional) effects are not encoded in any existing industrial standard. They are encoded, in

unambiguous computer-interpretable form in OWL-S, as described in Section 4. 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 WSC 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 Section 6, and as argued there, requires the OWL-S representation of the properties, capabilities and functioning of a Web service as described in Section 3.

8.2 ABS and Service Composition

ABS technology and research on reasoning about action and change have had a significant influence on the evolution of techniques for WSC. The views of the relationship between agents and services posited in Section 1.1 yield two different characterizations of the WSC task in the context of ABS.

If we view services as agents, then the composition problem can be conceived as a restricted form of a MAS task composition problem or planning problem (e.g., [6]). Each individual service is conceived as an agent with a set of capabilities, and the composition problem requires these services to coordinate to achieve some objective. Unfortunately, current Web services are not capable of the level of coordination required for true MAS task composition or planning. Most Web services are passive and taskable, demonstrating little proactivity. Their capabilities are brittle, limited to execution of a fixed, preconceived workflow, generally with only two-party interaction. As such, realization of WSC by appealing to the view of services as agents generally requires creation of a single agent to coordinate the execution of other Web service agents, which does not exploit the power of MAS planning or task composition. While viewing WSC as MAS planning or task composition provides an ambitious vision for the evolution of empowered next-generation Web services and the role they might play in task composition, the research in MAS can only be exploited in a limited fashion with today's Web services.

The alternative view of agents composed of, deployed as, or dynamically extended by services, yields a much more compelling and fruitful characterization of the composition problem. In such a view, we conceive Web services as rich sensors and end effectors to an agent. Information-gathering Web services provide sensory capabilities to an agent, collecting information about the world for the agent. World-altering Web services act as end effectors, realizing changes in the world on behalf of the agent. In

such a view of the world, the composition task is conceived as an agent-based planning problem. In what follows, we describe how techniques from agent-based planning and reasoning about action and change have been exploited to date for WSC, and the role that OWL-S plays in providing descriptions of Web services that enable realization of this vision.

8.3 Realizing Service Composition

There are several different approaches to WSC. All characterize OWL-S processes as actions with inputs, outputs, preconditions and effects, and use planning technology to achieve WSC. For example, the work of [24] models processes in the same format as a STRIPS operator [15] 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. [30] also investigates the use of plan synthesis for WSC, 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., [28], [29]). 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., [39], [45]) have used the paradigm of Hierarchical Task Network (HTN) planning (e.g., [31]) 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). [45] uses the SHOP2 system (e.g., [31], [32]), 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). [28],[29],[39] 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 [39] 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 [24] and [45], 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. The location of the most appropriate Web service makes an essential use of the OWL-S/UDDI Matchmaker [37],[38] that performs a semantic capability match between a capability description and the service profiles of the available Web services.

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 [20] that augments BPEL4WS, a popular business-process language [2], 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 [23]. 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.

8.4 The Road Ahead

While this early work is promising, we are still some distance from the goal of automated WSC. We have argued that we need rich, representations of Web services in a language with a well-defined semantics, to enable automated WSC. 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 [20] 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 WSC is at the heart of seamless interoperation among Web services. With adoption of approaches to WS description such as OWL-S and advances in agent-based planning-related technologies, we believe that broad-scale automated WSC is well within reach.

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 Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte (Editor), Ivana Trickovic, and 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] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web". *Scientific American*, 284(5):34--43, 2001.

[4] A. Bernstein and M. Klein, "High Precision Service Retrieval". In *Proceedings of the First International Semantic Web Conference* (ISWC 2002), Sardegna, 2002.

[5] 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.

[6] Frances M. T. Brazier, Barbara Dunin-Keplicz, Jan Treur, and Rineke Verbrugge, "Modelling Internal Dynamic Behaviour of BDI Agents". In *Proceedings of the ModelAge Workshop, 1997*, pp. 36-56.

[7] J.J. Bryson and L. A. Stein, "Modularity and Design in Reactive Intelligence". In *Proceedings of the 17th Int'l. Joint Conference on Artificial Intelligence,* pp. 1115-1120. Morgan Kaufmann, San Francisco, 2001.

[8] Joanna J. Bryson, David L. Martin, Sheila A. McIlraith, and Lynn Andrea Stein, "Toward Behavioral Intelligence in the Semantic Web". In *IEEE Computer*, pp. 48-54, November 2002.

[9] Ion Constantinescu and Boy Faltings, "Efficient Matchmaking and Discovery Services". In *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI03).* Halifax, Canada. 2003

[10] DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara), "DAML-S: Web Service Description for the Semantic Web". In *Proceedings of the International Semantic Web Conference (ISWC),* pages 348–363, June 2002.

[11] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1", 2001. At http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, and Said Tabet, "OWL Rules Language, Draft version". Technical report, 29 October 2003

[17] Stefan Kokkelink and Roland Schwänzl, "Expressing Qualified Dublin Core in RDF / XML". At http://dublincore.org/documents/dcq-rdf-xml/index.shtml.

[18] 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.

[19] 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.

[20] 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

[21] David Martin, Adam Cheyer and Douglas Moran, "The Open Agent Architecture: A Framework for Building Distributed Software Systems". In *Applied Artificial Intelligence*, 13(1-2), 1999, pp 92-128.

[22] David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, Sheila McIlraith, "Describing Web Services using OWL-S and WSDL". May 2004. At http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html

[23] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin, "Ontology-Enabled Pervasive Computing Applications". In *IEEE Intelligent Systems*, 18(10):68-72, 2003.

[24] 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.

[25] D McDermott, "The Planning Domain Definition Language Manual". *Yale Computer Science Report 1165* (CVC Report 980003), 1998.

[26] D. McDermott and D. Dou, "Representing Disjunction and Quantifiers in RDF". In *Proceedings of the First International Semantic Web Conference* (ISWC2002), 2002.

[27] 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/ .

[28] 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.

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

[30] 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.

[31] 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.

[32] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN Planning System". To appear, *Journal Artificial Intelligence Research,* 2003.

[33] Marian H. Nodine, Anne H. H. Ngu, Anthony R. Cassandra, and William Bohrer, "Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth™". In *IEEE Transactions on Knowledge and Data Engineering,* 15(5), 2003, pp1082-1098.

[34] OWL-S Coalition, "OWL-S 1.1 Release". At http://www.daml.org/services/owl-s/1.1/.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Katia Sycara, Mattheus Klusch, Seth Widoff and Janguo Lu, "Dynamic Service Matchmaking Among Agents in Open Information Environments". In *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems),* 28(1), 1999, pp 47-53.

[41] The Rule Markup Initiative. At http://www.dfki.uni-kl.de/ruleml/.

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

[43] Web Services Choreography Working Group. At http://www.w3.org/2002/ws/chor/

[44] Web Services Description Working Group. At http://www.w3.org/2002/ws/desc/

[45] 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.

[46] Y. Zou, T. Finin, L. Ding, H. Chen, R. Pan, "Using Semantic Web technology in Multi-Agent Systems: a case study in the TAGA trading agent environment", In *Proceedings of the 5th International Conference on Electronic Commerce*, pp 95-101, 2003.