

Autonomous Semantic Web Services

The DARPA Agent Markup Language for Services (DAML-S) provides a mechanism that begins to bridge the gap between the Web services infrastructure and the Semantic Web.

**Massimo Paolucci
and Katia Sycara**
Carnegie Mellon University

The Web, as we know it, is a collection of human-readable pages that are virtually unintelligible to computer programs. While the Web emerged as a global repository of digitized information, this very information is, by and large, unavailable for automatic computation. Two parallel efforts have emerged in recent years that could overcome this paradox: the Semantic Web¹ is providing tools for explicit markup of Web content, and Web services could create a network in which programs act as independent agents that produce and consume information, enabling automated business transactions.

Ideally, these two efforts should support each other. The Semantic Web will help create a repository of computer-readable data, and Web services will provide the tools for automatically using that data. Somewhat surprisingly, however, there have been few points of contact

between them to date. Semantic Web research focuses mostly on markup languages for annotating Web pages and on the inferential power needed to derive consequences from the annotated pages – essentially transforming the Web into a knowledge base. Web services efforts concentrate on interoperability standards and protocols for performing business-to-business (B2B) transactions.

In our work at Carnegie Mellon, we focus on research that attempts to bridge the gap. We adopt the vision of Web services as autonomous goal-directed agents that select other agents to interact with and that flexibly negotiate their interaction models, acting variously in client-server and peer-to-peer modes. The resulting Web services, which we call *autonomous Semantic Web services*, use ontologies and semantically annotated Web pages to automate the fulfillment of tasks and transactions. In particular, these

services use the Semantic Web to support capability-based discovery and interoperation at runtime.

A first step toward this vision is to develop formal languages and inference mechanisms for representing and reasoning with core Web service concepts. The DARPA Agent Markup Language for Services (DAML-S)² is the first attempt to define such a language. This article describes DAML-S and some example computational models that show how it can be viewed as the first step in bridging the gap between the Semantic Web and proposed industry standards for Web services.

The Semantic Web for Web Services

One objective behind the Semantic Web is to provide languages for expressing the content of Web pages and making that information accessible to agents and computer programs. More precisely, the Semantic Web is based on a set of languages such as the Resource Description Framework (RDF), DAML+OIL, and the more recent Web Ontology Language (OWL), which can be used to annotate Web content. These languages have well-defined semantics and inferential procedures that let agents draw inferences from the languages' statements. Using the semantic markup for the US National Oceanic and Atmospheric Administration's page reporting Pittsburgh's weather conditions, for example, an agent could learn that the current condition is *heavy snow*. The agent might further learn from the Pittsburgh school board site's semantic markup that all schools are closed on days of heavy snow. Combining the two pieces of information, the agent could infer that Pittsburgh schools are closed today.

The Semantic Web's second element is a set of ontologies that provide conceptual models for interpreting the information provided. An ontology of weather might contain concepts such as *temperature*, *snowy*, *cloudy*, and *sunny*, for example, and relationships between the terms.

The Semantic Web vision is about transforming the Web into an Internet-wide knowledge-representation system in which ontologies provide the conceptual framework for interpreting the information provided by Web pages. To produce the types of inferences we've described, the Semantic Web requires computational processes and agents that can interpret semantic content and derive consequences from the information they collect.

The Semantic Web also supports a more distributed computational model in which a requester transacts with multiple Web services, solving prob-

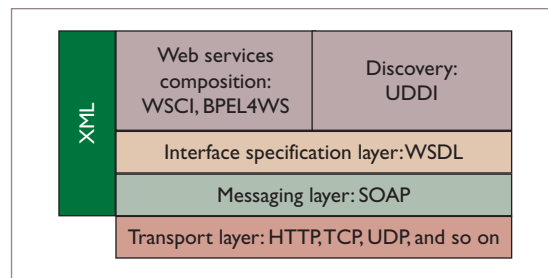


Figure 1. Web services infrastructure. The current infrastructure uses XML to describe multiple layers of abstraction from the transport mechanism. These include message description (SOAP), a mapping from messages to operations performed by the Web service (WSDL), abstract process representation (BPEL4WS and WSCI), and discovery (UDDI).

lems through collaboration and negotiation. Within this scheme, ontologies not only define a shared conceptualization for interpreting semantic markup of Web sites, but also provide a shared vocabulary that lets services across the Web use the same terminology to interpret each other's messages.

Ultimately, the Semantic Web will provide the basic mechanisms for extracting information from Web pages and the basic knowledge that Web services will use in all transactions. In addition to knowledge, however, Web services need an infrastructure that facilitates reliable communication — registries to locate other services, reputation services, guarantees of secure and private transactions, and so on. Such an infrastructure falls outside the current view of the Semantic Web's scope.

Current Web Services Infrastructure

Researchers and developers have already proposed many definitions for Web services. Simply put, they are programs that interoperate on the Web. The success of e-commerce in the late '90s and the recent plethora of interoperability standards proposed for Web-based business transactions have generated significant interest in automating program interactions for B2B e-commerce. These efforts have created a vision of a Web of dynamically interoperating nodes rather than static pages. This is the vision that Web services try to realize.

Basic Standards

The Web services infrastructure will provide the basic standards that let Web services interact. The diagram in Figure 1 shows how some popular proposed standards could fit together. The unifying factor is XML, as shown by the left column,

```

<types> ... </types>

<message name="GetTradePriceInput">
  <part name="request"
    type="xsd:string"/>
</message>
<message name="GetTradePriceOutput">
  <part name="response"
    type="xsd:float"/>
</message>

<portType name="StockQuotePortType">
  <operation name="GetTradePrice">
    <input message="GetTradePriceInput"/>
    <outputmessage="GetTradePriceOutput"/>
  </operation>
</portType>

<binding> ... </binding>

<service> ... </service>

```

Figure 2. Example WSDL description. This service performs the operation `GetTradePrice`, described by the input message `GetTradePriceInput` (string) and the output message `GetTradePriceOutput` (float). Much remains unknown, however, regarding the information to be encoded in the string and how to interpret the output float.

which cuts across all layers. The two most popular proposed standards are SOAP³ and the Web Services Description Language (WSDL).⁴ SOAP defines a format for passing messages between Web services, and WSDL describes each service's interface – how to contact it (through RPC or asynchronous messaging, for example) and how to serialize the information exchanged. SOAP and WSDL describe the atomic components of Web services' interaction. More recent proposed standards, such as the Web Service Choreography Interface (WSCI)⁵ and Business Process Execution Language for Web Services (BPEL4WS),⁶ provide mechanisms for describing how multiple Web Services can be assembled to participate in shared business processes.

To facilitate service discovery, Web services registries are an important part of the Semantic Web infrastructure. The emerging standard for Web services registries – Universal Description, Discovery, and Integration (UDDI) – provides a set of publishing, browsing, and inquiry functionalities for extracting information from a given registry. Using UDDI registries, developers can find Web services that they want to use.

UDDI descriptions include a host of useful information, such as the Web service provider and the binding (including the transport protocol port) that lets a requester invoke the service. In addition, Web service descriptions in UDDI can refer to TModels – unbounded attribute sets that can be associated with Web services – which can represent any type of information about a given service. TModels can specify a Web service WSDL description's location, for example, or a Web service's classification within a taxonomy, such as the United Nations Standard Products and Services Code.

Need for Autonomy

One overarching characteristic of the Web services infrastructure is its lack of semantic information. It relies exclusively on XML for interoperability, but that guarantees only syntactic interoperability. Expressing message content in XML lets Web services parse each other's messages, but it does not facilitate semantic "understanding" of the messages' contents.

Consider the WSDL fragment in Figure 2, for example, which describes a stock-reporting Web service. The service performs only one operation, `GetTradePrice`, which requires one input, `GetTradePriceInput` (of type string), and produces an output `GetTradePriceOutput` (of type float). This description specifies the syntactic type for the data transferred, but not what data it expects. We do not know, for example, whether the Web service expects a company name or a ticker symbol as input; similarly, we do not know whether the output will be the latest quote, the stock's beta value, or the 12-week average.

Industry's current proposals for the Web services infrastructure require programmers to reach explicit agreement on both the way their Web services interact and the format of the messages they exchange. Programmers must hard-code these interactions, as well as the way the services should interpret the messages. Programmers are also responsible for modifying their Web services when something changes in the interaction patterns or when something breaks. Ultimately, the growing Web services infrastructure facilitates the specification of agreements between programmers, but the fact that it does not support automatic Web service reconfiguration creates an infrastructure that is inherently brittle, inflexible, and inevitably expensive to maintain.

To overcome this brittleness, we need to increase Web services' autonomy by letting them reconfig-

ure their interaction patterns. Any increase in autonomy will let Web services react to changes while minimizing programmers' direct intervention.

The lack of explicit semantics prevents Web services from acting autonomously by understanding each other's messages and what tasks each service performs. In addition, current Web service proposals fail to enable semantic representations of business relations, contracts, and business rules in a machine-understandable way. Enriching the Web services infrastructure with semantics will let Web services

- explicitly express and reason about business relations and rules;
- represent and reason about the task a Web service performs (selling books or verifying credit cards, for example), thus enabling automated service discovery based on explicit advertisements and descriptions of service functionality;
- represent and reason about message ordering;
- understand the meaning of exchanged messages;
- represent and reason about preconditions for using services and the effects of invoking them; and
- combine Web services to achieve more complex services.

The Semantic Web has the potential to provide the Web services infrastructure with the information it needs through formal languages and ontologies for reasoning about service descriptions, message content, business rules, and relationships between ontologies.

DAML-S

As both a language and an ontology for describing Web services, DAML-S attempts to close the gap between the Semantic Web and Web services. As an ontology, it uses DAML+OIL-based constructs to define the concept of a Web service; as a language, DAML-S supports the description of specific Web services that users or other services can discover and invoke using standards such as WSDL and SOAP. DAML-S uses semantic annotations and ontologies to relate each Web service's description to a description of its operational domain. For example, a DAML-S description of a stock-reporting service might specify the data it reports, its delay versus the market, and the cost of using the service. The Web service's clients might use DAML+OIL to determine what kind of data the service reports, how to contact it, and so on.

Figure 3 shows DAML-S's structure and how it

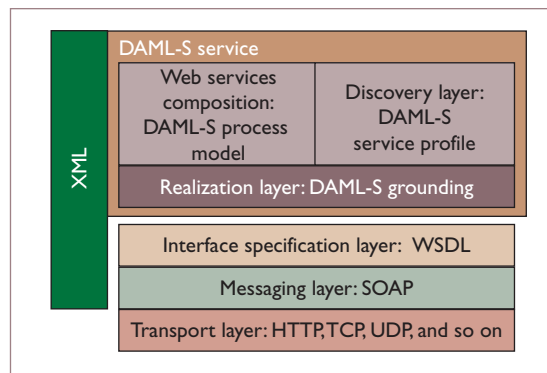


Figure 3. Web services architecture using DAML-S. DAML-S provides semantic composition and discovery layers and a grounding to map Web services descriptions to message-transport specifications.

relates to other components of the Web services infrastructure. A DAML-S Web service is specified by four descriptions:

- the service profile,
- the process model,
- the service grounding, and
- a DAML-S service description that connects the other three.

Furthermore, DAML-S supports the use of WSDL to specify Web service interfaces, SOAP to describe the messaging layer, and some transport protocol to connect two Web services. At the messaging and transport levels, DAML-S is thus consistent with the rest of the proposed Web services standards.

The *service profile* provides a high-level view of a given Web service. It is the DAML-S analog to the Web service representation that UDDI provides in the Web services infrastructure, although the two have some sharp differences as well as similarities. Some information, such as a Web service's provider, is present in both descriptions, but the service profile supports properties such as the representation of capabilities – the tasks the service performs – that UDDI does not support. On the other hand, UDDI describes the ports the Web service uses, whereas DAML-S relegates this information to other modules of the description, such as the grounding (described below).

The *process model* specifies the tasks a Web service performs, the order in which it performs them, and the consequences of each. A client can use the process model to derive the service's choreography, or message-exchange pattern, by figuring out what inputs it expects, when it expects them, what outputs it reports, and when. The process model's role

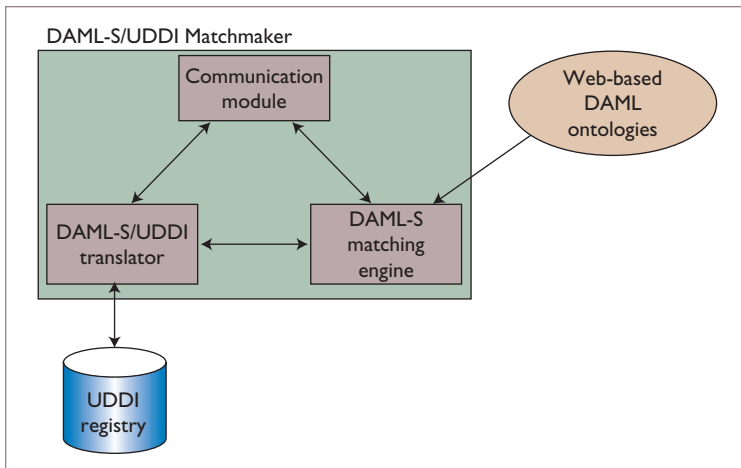


Figure 4. DAML-S/UDDI Matchmaker architecture. Web services advertise or request through the communication module using DAML-S. Advertisements are stored in the UDDI registry, and requests are sent to the DAML-S matching engine.

is similar to emerging standards such as BPEL4WS and WSCI, but focuses more on the effects of executing a service’s different components.

The *service grounding* binds the abstract description of a Web service’s information exchanges – defined in terms of inputs and outputs in the process model – with an explicit WSDL operation, and through WSDL to SOAP messages and transport-layer information.

DAML-S’s reliance on DAML+OIL, as well as WSDL and SOAP, shows how proposed Web services standards can be enriched with semantic information. DAML-S adds formal content representations and reasoning about interactions and capabilities to Web service specifications. Therefore, DAML-S-enabled Web services use UDDI, WSDL, and SOAP to discover other services and interact with them, and they use DAML-S to integrate these interactions in their own problem solving.

Managing Web Services with DAML-S

Now that we have a theoretical framework that relates DAML-S, the Semantic Web, and the Web services infrastructure, we need a computational model that transforms theory into concrete Semantic Web services. We have implemented tools for Semantic Web service discovery and invocation making use of DAML-S and complementing current Web services systems. Here we describe the DAML-S/UDDI Matchmaker and the architecture of a DAML-S-empowered Web service.

DAML-S-Enabled Service Discovery

The DAML-S service profile relies on ontologies

to specify what type of information the Web service reports and what effects its execution produces. At discovery time, a Web service generates a request that contains a profile for the ideal service it wants to interact with. The discovery process selects a Web service provider’s profile that matches the request.

Although DAML-S profiles and UDDI Web-service descriptions contain different information, they share the goal of facilitating Web-service discovery. The combination could thus provide rich representations for Web services.⁷ Using UDDI’s TModels to encode DAML-S capability descriptions, we can reconcile the differences between the two. Once we have the capabilities encoded, we can add a new module to UDDI: the *matching engine* performs inferences based on DAML+OIL logics and effectively adds capability matches to UDDI.⁸ The result is the DAML-S/UDDI Matchmaker for Web services, shown in Figure 4.

The Matchmaker receives Web-service advertisements, information inquiries, and requests for capabilities through the *communication module*, which implements a simple inquiry-and-publish API. The communication module then sends the advertisements and inquiries to UDDI through the *DAML-S/UDDI translator*, which transforms DAML-S encoded advertisements into UDDI format. The communication module directs requests for capabilities to the DAML-S matching engine, which selects those Web services whose advertised capabilities match the request. The matching is complicated by the fact that providers and requesters have different views on Web-service functionality. Thus, the matching engine can’t base the selection on strings or keywords. Rather, it must match semantic descriptions of capabilities to access the deeper meaning of the advertisements and requests.

Consider, for example, a service provider advertising that it sells *pet food*, and a requester looking to buy *dog food*. A UDDI-style registry would be unable to match the request because keyword matching is not powerful enough to identify the relationship between *pet food* and *dog food*. Instead, DAML-S profiles let service providers express concepts that are explicitly related via ontologies. In this case, the provider could specify that *dog* is a type of *pet*, and the DAML-S matching engine could recognize a semantic match between the request and the advertisement.

The DAML-S matching algorithm accommodates the differences between an advertisement and a request by producing flexible matches – recognizing degrees of similarity – on the basis of

available ontologies. Basically, the matching engine attempts to verify whether the requested outputs are a subset of those generated by the advertisement, and whether the advertisement's inputs subsume those of the request. When these conditions are satisfied, the advertised service generates the outputs that the requester expects and the requester can provide all the inputs the Web service expects. The degree of satisfaction between these two rules determines the degree of match between provider and requester.

Figure 5 shows the internal architecture of the DAML-S matching engine, which establishes the degree of match between requests and advertisements. It relies on two data stores: the *advertisement database* stores the Web services' capabilities, and the *ontologies database* stores ontologies that have been downloaded from the Web. The matching engine draws advertisements from the advertisement database and uses a *DAML+OIL reasoner* to verify the relationship between concepts in the ontologies database.

The DAML-S/UDDI Matchmaker's architecture combines the two approaches: it uses UDDI to store advertisements and retrieve information about Web services, and it exploits DAML-S to represent and match capabilities.

Using DAML-S to Manage Interactions

The discovery process produces the Web services that provide a given capability. The next problem is to invoke those Web Services to solve a problem or answer a question. Semantic Web services can use the DAML-S process model and grounding to manage their interactions with other Web services. The diagram in Figure 6 shows a DAML-S-based Web service's architecture. The architecture's core is represented by the three components in the center column:

- the Web service invocation,
- the DAML parser, and
- the DAML-S virtual machine (VM).

The *Web service invocation* module is responsible for contacting other services and receiving messages from them. Such transactions might be based on SOAP messaging, HTTP, or any other mode of communication described by the service provider's WSDL specification.

Upon receiving a message, the Web service invocation module extracts the payload and sends it to the *DAML parser*, which downloads DAML+OIL ontologies and DAML-S descriptions

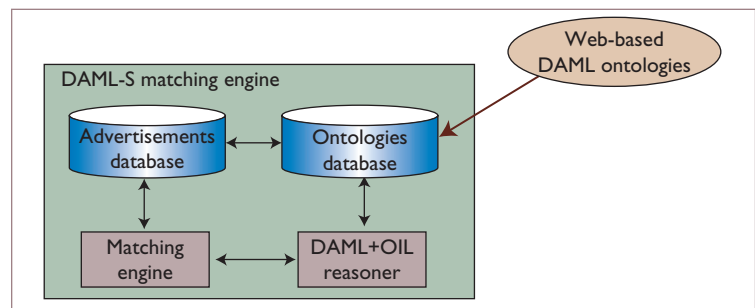


Figure 5. DAML-S matching-engine architecture. The matching process uses advertisements received by the registry and a DAML reasoner to locate which advertisements match the given request.

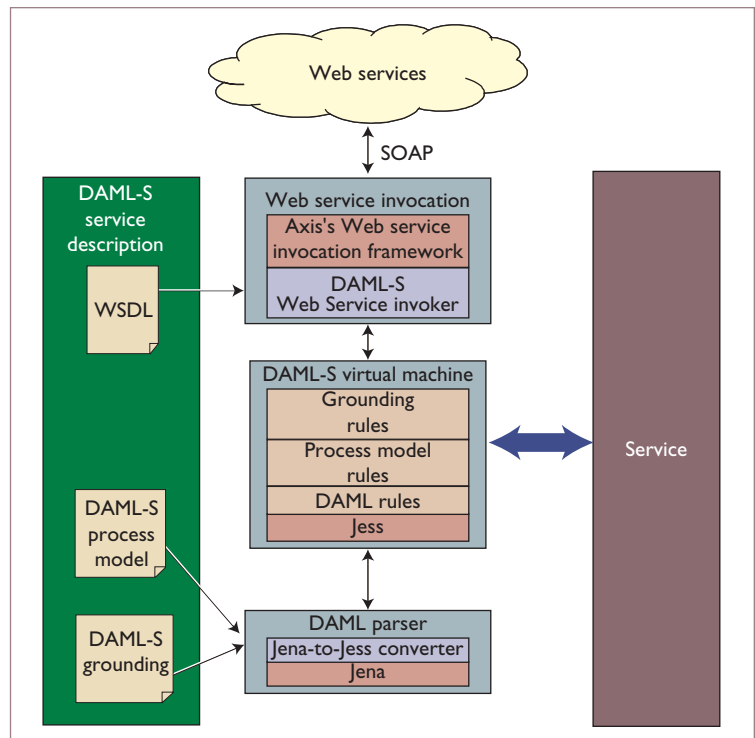


Figure 6. DAML-S-based Web service. The service uses the information on the left to interact with other Web services. The functionalities in the three center modules control the interaction, and the service module on the right is the service's main body.

of other services from the Internet. The parser then transforms fragments of the ontologies into predicates that the DAML inference engine can use.

The *DAML-S VM* is the center of our implementation. It uses the ontologies and DAML-S specifications passed to it by the parser and invocation module to make sense of the messages it receives, and to decide what kind of information to send next. The VM uses a set of rules that implement the semantics of the DAML-S process model and grounding. It uses the grounding to transform the abstract information exchanges described by

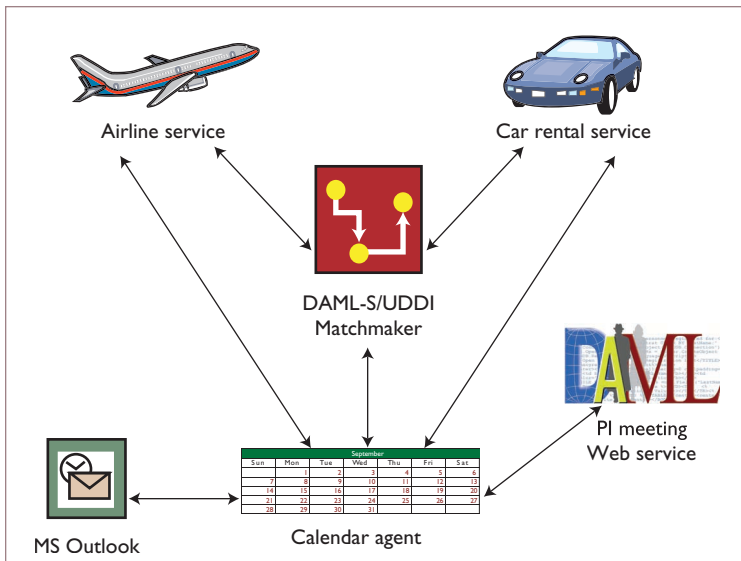


Figure 7. Test application architecture. The calendar agent uses the DAML-PI meeting Web service and the user's MS Outlook calendar to organize a trip.

the process model into concrete message content that it passes to the Web service invocation module, which generates the actual response messages and sends them to other Web services.

The right-hand column in Figure 6 shows the Web service's main body: the service module. In a financial consulting Web service, this module might contain software that performs financial calculations or suggests stocks to buy. In the figure, it is displayed as a "black box" because DAML-S does not make any explicit assumptions about the service. Rather, the goal is to facilitate autonomous interaction between Web services.

The service module is responsible for many of the decisions that the Web service makes while using DAML-S. For example, a financial-consulting Web service might need to interact with a stock-quote service or financial-news services, sending them requests and information. It would then use the information it received to solve some problem. The service also decides what to subcontract to other Web services, and which potential providers' capability descriptions to submit to the DAML-S/UDDI Matchmaker.

Test Application

We tested our architecture by developing a set of Web services that collaborate to organize a trip to a meeting of the DAML Principal Investigators. Figure 7 shows the resulting system's organization.

In our scenario, the user asks the Retsina calendar agent⁹ to make travel arrangements to the meeting. The calendar agent contacts the DAML-

PI meeting Web service to learn the meeting's dates and location. It then uses MS Outlook to verify that there are no schedule conflicts for the user. Finally, the agent books flight, car, and hotel from different Web services using their DAML-S descriptions.

In this application, the calendar agent is the service module, as shown in the generic DAML-S Web service in Figure 6. We expanded the Retsina agent's functionalities by adding a planning component based on HITaP,¹⁰ a hierarchical planner that interleaves planning and execution. During the planning phase, the calendar agent attempts to construct a plan that satisfies the goal of bringing the user to the meeting. When it cannot satisfy one of the preconditions, the calendar agent first queries the DAML-S/UDDI Matchmaker to find a Web service that can fulfill it, and then queries the agent located. The calendar agent then downloads the DAML-S Web service's description and executes the process model using the DAML-S VM. Finally, it extracts the information contained in the messages exchanged with the Web services and constructs a travel schedule, which it saves in the user's Outlook calendar.

Conclusion

DAML-S is not just an academic concept. We can use it to automatically control interactions between Web services, thus leading the way toward autonomous Semantic Web services. The work presented here shows the need for widespread ontologies to provide an inference framework that lets Web services resolve discrepancies and mismatches between the knowledge they're using.

By lowering the boundaries for automatic interoperation, ontology-based languages for describing Web services have enormous potential for business on the Web. Ontology-based languages let Web services adapt to changes in message content or interaction protocol. As our prototype application shows, they also provide the basis for creating on-demand services.

DAML-S provides a first step toward achieving this goal, but a lot of work remains to be done. It is unclear whether DAML-S provides all the information that is needed to represent Web services, for example. More importantly, it is also unclear how to fill the gap between DAML-S and the current Web services infrastructure. Some work in this direction has been done – the DAML-S grounding and the mapping from DAML-S to UDDI, for example – but we do not yet know whether there is a "low hanging fruit" that could provide great and

practical gains for the Web Services infrastructure with minimal use of semantic information. □

Acknowledgments


We thank Anupriya Ankolekar, Takahiro Kawamura, Takuya Nishimura, Terry Payne, and Naveen Srinivasan for useful discussions on different parts of this article. We also thank the members of the DAML-S coalition for the many discussions on DAML-S. DARPA and the US Office of Naval Research supported this work as part of the DAML program under US Air Force Research Laboratory contract F30601-00-2-0592 and the Interoperability program under ONR contract N00014-02-1-0499 to Carnegie Mellon University.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.*, vol. 284, no. 5, 2001, pp. 34-43.
2. DAML-S Coalition, "DAML-S: Web Service Description for the Semantic Web," *Proc. Int'l Semantic Web Conf. (ISWC)*, LNCS 2342, Springer Verlag, 2002, pp. 348-363.
3. M. Gudgin, et al., "SOAP Version 1.2 Part 1: Messaging Framework," W3C recommendation, 24 June 2003.
4. E. Christensen et al., "Web Services Description Language, version 1.1," W3C note, Mar. 2001; www.w3.org/TR/2001/NOTE-wsdl-20010315.
5. A. Arkin et al., "Web Service Choreography Interface (WSCI) 1.0," draft specification, BEA Systems, Intalio, SAP AG, and Sun Microsystems, 2002; <http://www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf>.
6. F. Curbera et al., "Business Process Execution Language for Web Services, Version 1.1," May 2003; <http://www-106.ibm.com/developerworks/library/ws-bpel/>.
7. M. Paolucci et al., "Importing the Semantic Web in UDDI," *Proc. E-Services and the Semantic Web*, LNCS 2512, Springer Verlag, 2002.
8. M. Paolucci et al., "Semantic Matching of Web Services Capabilities," *Proc. Int'l Semantic Web Conf. (ISWC)*, LNCS 2342, Springer Verlag, 2002, pp. 333-347.
9. T.R. Payne, R. Singh, and K. Sycara, "Calendar Agents on the Semantic Web," *IEEE Intelligent Systems*, vol. 17, no. 3, May/June 2002, pp. 84-86.
10. M. Paolucci et al., "A Planning Component for RETSINA Agents," *Intelligent Agents VI*, LNAI 1757, N.R. Jennings and Y. Lesperance, eds., Springer Verlag, 2000.

Massimo Paolucci is a principal research programmer at Carnegie Mellon University. His research interests include agent and multiagent architectures and their relation to the Web and Web services, in particular. He received an MS in computational linguistics from Carnegie Mellon University and an MS in intelligent systems from the University of Pittsburgh. He is a member of both the DAML-S coalition and the Semantic Web Services Initiative (SWSI) architecture committee. Contact him at paolucci@cs.cmu.edu.

Katia Sycara is a research professor in the School of Computer Science at Carnegie Mellon University and director of the Advanced Information Technology Laboratory. Her research interests include autonomous agents; planning; learning and coordination of multiple agents in open; uncertain, and dynamic environments; Web services; the Semantic Web; and case-based reasoning. She received a PhD in computer science from Georgia Institute of Technology. Sycara is a member of the DAML-S coalition and the US chair of the SWSI executive committee. She is a fellow of the AAAI, founding editor in chief of the *Journal of Autonomous Agents and Multi-Agent Systems*, and the 2002 recipient of the ACM Autonomous Agents Research Award. Contact her at katia@cs.cmu.edu.



IEEE
distributed systems
ONLINE
Expert-authored articles and resources

IEEE Distributed Systems Online brings you peer-reviewed features, tutorials, and expert-moderated pages covering a growing spectrum of important topics, including

- Grid Computing
- Mobile and Wireless
- Distributed Agents
- Security
- Middleware
- and more!

dsonline.computer.org

To receive regular updates, email dsonline@computer.org