Editor: James Hendler
University of Maryland
hendler@cs.umd.edu

Bringing Semantics to Web Services

Sheila A. McIlraith, *Stanford University*
David L. Martin, *SRI International*

A revolution is underway in computing, and if you believe pundits such as Vint Cerf, "father of the Internet," it won't be long before your bathroom scale surreptitiously transmits your weight to your doctor, who might command a stop to the rocky road ice cream your fridge automatically orders for you from www.groceries.com.[1] While many of us have heard such amusing tales, ice cream lovers can relax for a little while. Pervasive networked devices and programs that can seamlessly interoperate are still a ways off. Realizing this vision requires a computing infrastructure that supports communication and interoperation between diverse, distributed computer programs and devices. Furthermore, to achieve this seamlessly, those programs and devices must know each others' capabilities and communicate requests and responses unambiguously. Enter Web services and the Semantic Web.

Web services are "self-contained, modular applications that can be described, published, located, and invoked over a network—generally, the World Wide Web."[2] (For further reading on Web services, see this issue's Trends & Controversies on p. 72). Typical examples of Web services include the suite of programs at www.amazon.com that collectively let users buy books, or those programs at www.ual.com that let users determine flight schedules and book flights. Industry leaders such as IBM, Microsoft, Hewlett-Packard, and Sun have been quick to develop distributed-computing infrastructure such as .NET, WebSphere, Web Service Platform, and Java 2 Platform Enterprise Edition. More and more organizations are adopting Web service protocols, such as SOAP (Simple Object Access Protocol), and WSDL (Web Service Definition Language) is slowly becoming the standard for describing communication-level mappings of Web service messages to communication protocols. Likewise, business process modeling languages such as XLANG, WSFL (Web Services Flow Language), and, most recently, BPEL4WS (Business Process Execution Language for Web Services), have been developed to model Web services. All describe Web service content in terms of XML syntax without a well-defined semantics.

Unfortunately, XML alone lacks both a well-defined semantics and sufficient expressive power to realize the vision of diverse Web services having wide-scale interoperability. Truly seamless interoperability between services that have not been predesigned to work together requires programs to describe their own capabilities and understand other services' capabilities. They must communicate the nature of the documents and requests they exchange and of any side effects associated with the requests' satisfaction. Likewise, they must be able to understand these properties in other services. To realize this vision, we must describe Web content, particularly Web service content and capabilities, in a language that goes beyond XML.

The Semantic Web vision of a next-generation Web that computers can unambiguously interpret addresses precisely this problem.[3] A key element to realizing the Semantic Web is developing a suitably rich language for encoding and describing Web content. Such a language must have a well-defined semantics, be sufficiently expressive to describe the complex interrelationships and constraints between Web objects, and be amenable to automated manipulation and reasoning with acceptable limits on time and resource requirements. Several languages build on XML. These include the Resource Description Framework, RDF Schema, DAML+OIL (see www.daml.org/2001/03/daml+oil-index.html),[4] and, most recently, the Web Ontology Language (OWL—see www.w3.org/News/2002#item110). DAML+OIL and OWL are Web ontology languages based on artificial intelligence knowledge representation work in description logics. They provide a natural way to describe class and subclass relationships between Web vocabulary, as well as constraints on the relationships between classes and between class instances.

Web services meet the Semantic Web

The Semantic Web services vision is to describe Web services' capabilities and content in an unambiguous, computer-interpretable language[5–7] and improve the quality and robustness of existing tasks, such as Web service discovery and invocation.[8] Semantic Web services will

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

Semantic Web Services at Your Service

Think of trying to refinance your mortgage on today's Web without the help of a brokering agency. Even assuming that the necessary providers (lender, appraiser, trust company, and so on) all have easy-to-use Web pages, you'll put in a huge effort sifting through search engine results to find them. You'll type in your personal data over and over again at each Web site, go through a painstaking, manual process to collect and organize the information on available options, and return to your computer many times to handle communications with the providers. This will likely take hours to complete, much of them spent in repetitive, mundane activities.

Now imagine doing this using Semantic Web services. By exploiting expressive service descriptions, automated reasoning techniques will support the development of a "personal mortgage assistant" program that helps you locate the various providers, supports you in choosing particular providers, and then, over time, monitors and manages the entire process by which documents are exchanged and approvals given. Semantic Web services' infrastructure will let this program accurately discover, select, and invoke appropriate providers that are available and provide a detailed ranking of them based on your situation and preferences. It will also be able to determine which particular combinations of providers can work together. Once you have approved the choice of lender, appraiser, and so on, the program will

- Compose a workflow, or process model, of steps to complete the mortgage approval process on the basis of the specifications of the individual providers' services
- Invoke constituent Web services at an appropriate time for you
- Monitor the status of each provider's role in the workflow

also enable a broad range of new automation tasks that humans previously performed, including automated composition, interoperation, execution monitoring, and recovery. To support this vision, Semantic Web services will provide more powerful Web service development tools that enable, among other things, automated simulation and verification of Web service properties and consistency-checking and debugging features.[9]

DAML-S

A key component of the Semantic Web services vision is the creation of a language for describing Web services. DAML-S is such a language (see www.daml.org/services).[7,10] It is a DAML+OIL ontology for describing Web services that a coalition of researchers created with support from DARPA.

DAML-S builds on industry efforts such as SOAP, WSDL, WSFL, XLANG, and BPEL4WS by adding rich typing and class information that we can use to describe and constrain the range of Web service capabilities much more effectively than XML data types. Furthermore, it integrates such rich class representations with a process model designed to capture not only the control flow and data flow of Web services but also their real-world side effects (preconditions and effects). Such a language enables the grouping of like services and like data types into taxonomic hierarchies, together with rich definitions of the relationships and constraints between classes and their instances. The well-defined semantics allows automated manipulation of these structures with a known outcome using powerful tools and reasoning techniques. In short, DAML-S makes automated interoperation feasible. (The sidebar "Semantic Web Services at Your Service" illustrates some of the possibilities in one familiar domain.)

The drivers for the design of DAML-S, then, are the envisioned automation tasks, including automated discovery, invocation, interoperation, composition, execution monitoring and recovery, simulation, and verification. To this end, the DAML-S ontology consists of three subontologies: the *service profile*, the *process model*, and the *grounding*.

Describing Web service capabilities

The service profile describes what the service can do, for purposes of advertising, discovery, and matchmaking. It enables creating a richly expressive "yellow pages" of services by taxonomically encoding the kinds of information a service-seeker (whether human or software agent) must have to determine if the service meets its needs. Service profiles can be exposed at a URL for crawlers to find. They can also be published in service registries, such as UDDI (Universal Description, Discovery, and Integration).[8] The formal structure and richness of description provided by DAML+OIL enables powerful forms of querying.

DAML-S supports constructing a subclass hierarchy of the **Profile** class with (potentially multiple) inheritance of properties. Because we describe classes in terms of defining properties, we can characterize a Web service as belonging to multiple classes. For example, we can characterize the **LocateBook** service at www.amazon.com as both a service that determines whether Amazon carries a particular book and a bibliographic reference tool by varying the specifications of its inputs, outputs, preconditions, and effects. Many of a profile's components are domain specific. For example, geographic constraints are relevant to a catering service or an airline but not to a journal archive.

Service profiles will let well-developed existing taxonomies of service categories—such as those in the United Nations Standard Product and Services Classification Code—evolve into more expressive class-hierarchical categorization schemes.

Describing Web service programs

As we mentioned earlier, the process model describes how the service works, including the program's control flow and data flow, which realize the service, and its preconditions and real-world side effects. The process model was designed for service requesters to use in connection with service selection, invocation, interoperation, composition, and monitoring, or for service tools to use for service simulations and verification. Service developers can also use the process model to help populate the service profile. The DAML-S process model is a superset of what typically exists

```
<daml:Class rdf:ID="Process">
 <rdfs:comment>The most general class of processes</rdfs:comment>
 <daml:disjointUnionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#AtomicProcess" />
        <daml:Class rdf:about="#SimpleProcess" />
        <daml:Class rdf:about="#CompositeProcess" />
 </daml:disjointUnionOf>
</daml:Class>
```

Figure 1. DAML-S description of the Process class.

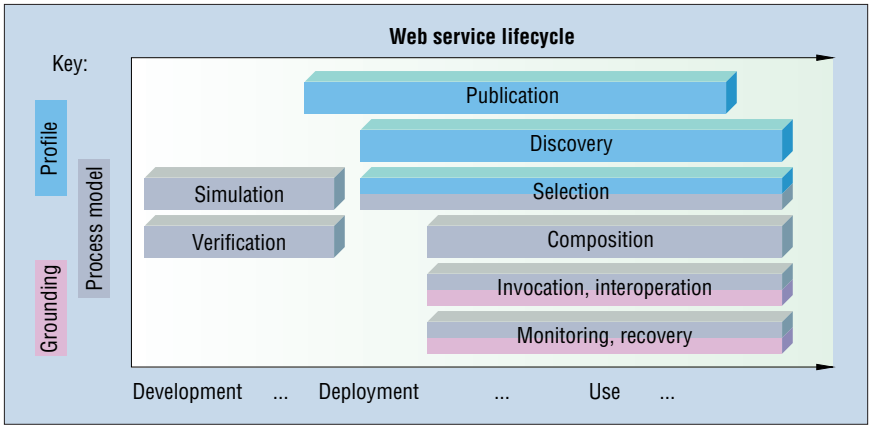


Figure 2. Applicability of the DAML-S profile, process model, and grounding ontologies to the Web service lifecycle.

in process-modeling and workflow languages. It combines a process-modeling language with both an AI-inspired action language and a language for describing classes and their interrelationships. Furthermore, the process model has a well-defined semantics.

Central to a DAML-S process model is the specification of a service's inputs, outputs, preconditions, and effects. Process inputs and outputs are named and typed using either DAML+OIL classes or data types that XML Schema provides. Any number of preconditions might exist, and they must all hold for the process to be invoked. Effects indicate what the service accomplishes or, more generally, real-world changes that the service causes. DAML-S lets you associate conditions with outputs and effects because a service's outputs and effects are often predicated on some observable characteristic of the system. We subdivide DAML-S's process model into three process types. Figure 1 shows how DAML+OIL expresses this subclassing.

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. Its results are returned in a response message.

Simple processes are like atomic processes in that they have 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 consist of subprocesses, which in turn can be either atomic, simple, or composite. Control constructs such as *sequence* and *if-then-else* specify a composite process's structure. Besides describing control flow, this structural specification includes argument binding constructs for indicating data flow.

Describing Web service access

The grounding 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
- An unambiguous means of exchanging data elements of the types the service requires and produces

DAML-S's default grounding approach relies on specification mechanisms that WSDL already provides but in conjunction with semantically richer descriptions available through DAML+OIL.[10]

DAML-S recap

The service profile is the primary construct by which a service is advertised, discovered, and selecte. But in some cases an agent involved in discovery or selection might also find it useful to inspect the service's process model to answer more detailed questions about the service. Having selected a service, an agent uses its process model, in conjunction with its grounding, to construct an appropriate sequence of messages for interacting with the service. As we mentioned earlier, the process model is also important for composing and monitoring processes, as well as simulation and verification. Figure 2 summarizes the relevance of the ontology's three subdivisions to various stages in a Web service's lifecycle.

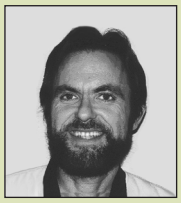
The DAML-S Coalition has recently announced the release of DAML-S version 0.7, which contains DAML+OIL ontology code for all three areas of the ontology, documentation of various kinds, and examples. This material is available at www.daml.org/services. You can also find information about the development of tools and DAML-S applications at this site. Ongoing work—moving toward version 1.0—includes further developments in each of the three areas, as well as development of a process execution subontology. Researchers from the US and the European Union are considering a larger collaborative effort on Semantic Web services that would include architectural issues along with the current focus on language issues. ■

Acknowledgments

DARPA funded this work as part of the DARPA Agent Markup Language (DAML) pro-

gram. The following researchers created DAML-S: Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara (Carnegie Melon University); Mark Burstein (BBN Technologies); Jerry Hobbs (USC Information Sciences Institute); Ora Lassila (Nokia); David Martin (SRI International), Drew McDermott (Yale University); Sheila McIlraith (Stanford University); Srini Narayanan (International Computer Science Institute); and Terry Payne (University of Southampton).

Sheila A. McIlraith is a senior research scientist in Stanford University's Knowledge Systems Laboratory and a project lead on KSL's DAML Web Services project. Her research interests include knowledge representation and reasoning techniques for the Web; for modeling, diagnosing, and controlling dynamical systems; and for model-based programming of devices and agents. She received her PhD in computer science from the University of Toronto. She is a founding member of the DAML Services Coalition. Contact her at sam@ksl.stanford.edu.



David L. Martin is a senior computer scientist at the Artificial Intelligence Center of SRI International and a principal investigator on SRI's DAML project. His research interests include Semantic Web and Web services technologies, agent-based software frameworks, pervasive computing, natural language processing, deductive databases, and software engineering methods. He received his MS in computer science from the University of California, Los Angeles. He is a founding member of the DAML Services Coalition. Contact him at martin@ai.sri.com.

References

1. M.J. Baker, "Vint Cerf: The 'Father of the Internet' Talks About the Future of His Offspring," *Digital Source*, http://www1.worldcom.com/ca/digital_source/articles/vintcerf/?SetLang=en.
2. K. Gottschalk et al., "Web Services Architecture Overview: The Next Stage of Evolution for E-Business," http://www-106.ibm.com/developerworks/library/w-ovr.
3. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, May 2001, pp. 34–43.
4. D. McGuinness et al., "DAML+OIL: An Ontology Language for the Semantic Web," *IEEE Intelligent Systems*, vol. 17, no. 5, Sept./Oct. 2002, pp. 72–80.
5. S. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, Mar./Apr. 2001, pp. 46–53.
6. G. Denker et al., "Querying and Accessing Information on the Semantic Web," *Proc. Semantic Web Workshop*, WWW10 Ltd., Hong Kong, 2001, pp. 67–77.
7. A. Ankolekar et al., "DAML-S: Semantic Markup for Web Services," *Proc. Int'l Semantic Web Working Symp.* (SWWS), SWWS Program Committee, 2001, pp. 411–430; www.semanticweb.org/SWWS/program/index.html.
8. M. Paolucci et al., "Semantic Matching of Web Services Capabilities," *The Semantic Web–ISWC 2002: Proc. 1st Int'l Semantic Web Conf.* (ISWC), Springer-Verlag, Berlin, 2002.
9. S. Narayanan and S. McIlraith, "Simulation, Verification, and Automated Composition of Web Services," *Proc. 11th Int'l World Wide Web Conf.* (WWW-11), 2002; http://www2002.org/CDROM.
10. A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," *The Semantic Web–ISWC 2002: Proc. 1st Int'l Semantic Web Conf.* (ISWC), Springer-Verlag, Berlin, 2002, pp. 348–363.