Toward Behavioral Intelligence in the Semantic Web

The semantic Web should extend the intelligence of agents, not just their knowledge. This agent-oriented design turns Web services into behavioral modules, allowing Web agents to act in the real world.

Joanna J. Bryson
University of Bath

David L. Martin
SRI International

Sheila A. McIlraith
Stanford University

Lynn Andrea Stein
Franklin W. Olin College of Engineering

Realizing the Web's full potential will require more than rapid access to information and services. It will also call for the development and support of intelligent schedulers, planners, and searchers that, with minimal direction, can serve as an omnipresent staff of advisers, secretaries, agents, brokers, and research assistants. These intelligent agents will plan everything from vacations to product development cycles, taking over diverse mundane tasks and substantively assisting with sophisticated projects.

Electronic commerce has brought this vision tantalizingly near. Organizations and individuals have connected an enormous variety of products and services to the Internet, making them accessible to browsers and other programs through simple communication protocols.

How can the AI community build intelligent agents to exploit these services? One approach is to adapt intelligent agents to the Web as it now stands. Agents must somehow understand Web pages in ordinary language, taking into account the host of other cultural protocols that reduce language ambiguities. With the current state of automated Web translation, this approach is difficult: Translate "Put my money in the Bank of Scotland" to Japanese and put it back on AltaVista and you get "Put my money on the Scottish Bank"—not an error a personal agent should make.

A more reliable strategy is to change the Web itself, making it accessible to existing knowledge representation and reasoning techniques. In the semantic Web of the not-too-distant future, service and content providers will mark pages in accordance with standardized conventions designed to reduce ambiguity and make automated reasoning easier.[1,2] Thus, users will be able to access Web resources by content rather than keywords. If you search for a bank, you'll be able to specify whether you want a financial institution, a picturesque riverside, or a toy for a child. What's more, so will your agent.

DEVELOPING A DISTRIBUTED INTELLIGENCE

A significant limitation of both the current Web and the first- generation semantic Web vision is that both are primarily about publishing and retrieving information. Neither concept explicitly recognizes the vast array of tasks that users could accomplish if the semantic Web becomes a distributed intelligence that can actually get things done.

We propose expanding the conceptual framework that underlies the semantic Web to include behavior in an integral way. This expansion will help unlock the Web's full potential by allowing it to function not only as a knower, but also as something that can *act* on that knowledge.

Our agent-oriented approach to building the semantic Web views a system as a collection of agents—human-like actors with beliefs, intentions, and abilities. This approach offers an intuitive way to reason about systems,[3] which makes it easier to build them. Software engineers can reason about large systems using the social skills our species has been honing for thousands of years.[4]

Agent-oriented software engineering emphasizes that the Web is not just about information; it's about services—about getting things done. The Web connects to companies and organizations, peo-

0018-9162/02/$17.00 © 2002 IEEE

ple, devices, and software programs. Services are a form of behavior, and, according to the Turing test, behavior is the fundamental attribute of intelligence. By connecting intelligent agents to these services, we are not just giving agents more information; we are giving them more behavior. We are extending their intelligence so that they can affect the real world.

Of course, agent-oriented software engineering is far from trivial. To achieve this vision for the semantic Web we need two things: a methodology for building agents that can safely extend their behavior and a way to mark up the semantic Web to support this type of extension.

To engineer extendable agents, we propose using an agent-oriented approach to software engineering called Behavior-Oriented Design.[5] BOD supplies both modularity—permitting compartmentalization—and mechanisms for intermodule coordination. The markup language we build on is a refinement of DAML-S (http://www.daml.org/services/).[6] DAML-S markup on services and BOD modules lets a BOD agent make informed decisions about service use. DAML-S also encodes the BOD reactive plans that combine, control, and arbitrate among these modules. The resulting agents are the new semantic Web's active intelligence.

BRINGING AGENTS TO THE WEB

DAML-S stems from the Defense Advanced Research Projects Agency's work to produce the DARPA Agent Markup Language.[7] DAML is itself an extension of the Extensible Markup Language and the Resource Description Framework, which aim to provide better specifications of relationships and ontologies within Web pages. The primary goal is to add unambiguously interpretable semantics to Web pages and thus enable the intelligent use of resources on the Web. DAML-S is DAML devoted to describing Web services. It will let users and software agents automatically discover, invoke, compose, and monitor Web resources that offer services, under specified constraints.

Using a formal language like DAML-S offers enormous benefits. Not only does it enable the precise selection of online services that are appropriate for specific tasks, but it also opens the door to powerful forms of reasoning about those services.[8] Because its designers recognized the problem of combining services, DAML-S already provides most of the infrastructure to support our approach. However, understanding the semantic Web as active agent intelligence does require some refinements to the current DAML-S specification.

Any agent design methodology suitable to exploiting discovered services must both emphasize modularity and have some mechanism for coordinating modules. Using BOD, designers develop modular agents in the behavior-based tradition. To coordinate and arbitrate among these modules, designers build explicit, hand-coded reactive plans. These reactive plans encode the agents' (possibly conflicting) goals and their priorities. For example, an agent may have the goal of buying a plane ticket and of retaining a positive balance in a bank account. Either of these goals may in theory have higher priority; BOD's reactive plans let the agent's designer specify which goal does have priority and in what circumstances. Meanwhile, the designer can encode the nitty-gritty of pursuing the goal inside a software module.

Using DAML-S opens the door to powerful forms of reasoning about Web content.

The BOD agent architecture is one of a number that combine behavior-based AI and reactive planning.[9] BOD's agents differ from other hybrid architectures because BOD gives more power and autonomy to the behavior modules and reduces the need for plans to arbitrate among modules that vie for resources. However, our proposals should apply to a range of agent architectures.

SERVICES, AGENTS, AND BEHAVIOR

A Web service is any Web-accessible program or device, and it may or may not affect the real world.[10] For example, a service that searches the Web for research on Haiti changes only the agent's own knowledge state; a service to buy an airline ticket debits the user's charge account and enters that user on the passenger list. Other examples of Web services are a software company providing a patch to fix a program, the police department sending an officer to check a house in response to an online burglar alarm, or a post office printing an e-mail message and delivering it as surface mail.

A composite service combines individual services in a way that adds value to the user. An example is a travel agency, where a customer specifies the preferred type of trip, and the travel agent selects or assists in selecting specific service providers, such as the airline and hotel. To combine the simpler services of which it is composed, a composite service typically has coordination or arbitration rules that let it prioritize or make tradeoffs, like booking an airline ticket to lock in a good rate before fully investigating hotel options.

An agent is any relatively autonomous actor, typically with

In employing a service, you are also, in a sense, acquiring new behavior.

- *goals*, conditions it works to achieve or maintain;
- *intentions*, goals and subgoals that it is currently pursuing,
- *beliefs*, knowledge about the world, which is necessarily limited and possibly inaccurate; and
- *behaviors*, actions it can take.

Most people view agents as consumers of services. However, in employing a service, you are also, in a sense, acquiring new behavior. For example, if you need to paint your house, you can buy a brush or hire a painter. You can use either to paint your house. Although you might not think of a hired painter as part of yourself, the painter helps you achieve your goal. Further, you must alter your own plans to monitor the painter and to accommodate his actions. The painter might even help you store memories: Years later, when you contact him for the paint brand and color, he can supply it.

So, as Andy Clark noted in *Being There: Putting Brain, Body and World Together Again* (MIT Press, Cambridge, Mass., 1996), things outside your person may become a part of your agency. This is at least as true for a software agent as it is for a human.

MODULAR AGENTS AND BOD

Modularity simplifies software engineering because it lets designers decompose a complex program into relatively simple modules. A programmer or designer can then develop and debug these modules independently. Modularity underlies object-oriented design as well as the behavior-based approach to AI, which has become at least part of many leading agent-architecture paradigms.[9,11]

Modularity generally simplifies design, but it also creates two formidable challenges. One is decomposition: how many modules and what resources in each? The other is coordination. What if independent components attempt contradictory behavior? For example, what if a module dedicated to charity and a module dedicated to housekeeping both decide to make a major expenditure in the same month? The overall system must have some way to arbitrate between these modules and decide which action will execute.

BOD, like object-oriented design, specifies that modules group actions that require shared variable state. Designers initially specify reactive plans as simple sequences of actions that the agent can be expected to take. BOD provides both a six-step guide for initial decomposition and a cyclic development process for actual implementation. The development process features heuristics for reevaluating which parts of the agent's intelligence the designer should represent in separate behaviors and which in the reactive plans. Consequently, the agent's structure stays as simple as possible while its behavioral complexity increases.[5]

Applying BOD to semantic Web development constrains some of this development process. On the Web, services are the behavior modules—black boxes from the perspective of any client agent, either human or artificial. The agent can turn knobs and switches (on a travel service, for example, it can choose a date and destination), but it has no control over how the module actually works. Nevertheless, from the user's perspective, agents are in many respects just like behavior modules. They provide "perceptual" information (tell you the price or availability of a flight), perform actions (sell you a ticket), and maintain state (remember your itinerary).

MODULE COORDINATION

The most popular way to arbitrate behavior-based modular systems is to incorporate hierarchical reactive plans into system execution.[9,11] Reactive planning addresses the problem of action selection by looking up the next action based on the current context, in contrast to deliberate or constructive planning, which involves search and means-to-ends reasoning. Reactive plans are established structures that support the look-up process. Hierarchical reactive plans are simple, robust plans, each element of which may itself be another reactive plan.

BOD uses parallel-rooted, ordered slip-stack hierarchical (POSH) reactive plans.[12] At the root of the plan hierarchy are the agent's top-level goals. For example, suppose Mojda has an agent with two jobs: notify her of meetings and search the Web for articles relevant to her research. The agent's plan hierarchy would have two root branches; the higher priority goal (branch) would be monitoring her schedule.

On every program cycle, a POSH agent's coordination module checks to see which root goal it should attend to (as determined by priority, preconditions, and optional scheduling). It then pursues progress toward that goal, as determined by the state of component behaviors and the last action recorded in that branch of the POSH plan.

A coordination module pursues progress toward a goal in several ways. First, a behavior module/Web service may be making progress independently and in parallel to the coordination module. In this case, the coordination module needs to determine merely whether the behavior module is complete.

Second, achieving the goal may require the agent to perform actions in a set sequence. In this case, the coordination module recalls its current place within the sequence and triggers the next step.

Finally, sometimes a goal requires that the agent perform actions sequentially, but the exact sequencing cannot be determined in advance. In this case, the designer supplies a basic reactive plan—an elaborated sequence that incorporates production-rule-like technology. The "How Basic Reactive Plans Work" sidebar describes this briefly.

How Basic Reactive Plans Work

A basic reactive plan is an elaboration of a simple sequence that allows for a reactive response to dynamic environments. Execution of a BRP can skip or repeat elements of the sequence as necessary. The final (terminating) element has the highest priority. A precondition guards each element, determining whether that element can execute. Each program cycle of the behavior-arbitration module executes the highest priority, currently executable element of the currently attended BRP. A BRP step is a tuple of priority, releaser, and action. Each BRP contains the small set of steps associated with achieving a particular goal condition. The releaser for a step is a conjunction of Boolean perceptual primitives that determines whether the step can execute. Each action can be either another BRP or a more primitive plan element.

The releaser and priority determine the order in which plan steps are expressed. If more than one step is operable, the priority determines which step executes. If no step can fire, the BRP terminates. The top priority step of a BRP is often, though not necessarily, a goal condition. In that case, its releaser recognizes that the BRP has succeeded, and its action terminates the BRP.

AGENT SUPPORT IN DAML-S

The semantic Web is Web intelligence waiting to be discovered and incorporated by intelligent agents. DAML-S has two roles in realizing that vision. First, every service must be described in such a way that an agent can know what it provides and how to interact with it. Second, DAML-S provides support for composite services, combinations of simpler services—or behaviors—and the coordination mechanisms—or reactive plans—used to combine those behaviors.

Composite services

DAML-S composite services create reliable, uniform, higher-level subgoals and specify services that can partially achieve these subgoals.[1] For example, if Mojda decides she needs a vacation, a composite service lets her say, "Buy me a ticket from here to Paris, respecting my usual preferences." instead of "Purchase AcmeAir Flight 309."

You can view a composite service as a conventional program or, more essentially, as another, more powerful service. However, for our purposes, it is more interesting to view a Web service as either an agent in itself or as part of an agent—an extension an agent could incorporate into itself once it finds and chooses to adopt a service.

One way an agent could interact with a composite service is if the composite service itself behaved as an agent. Suppose userAgent is an agent serving a user. userAgent might discover and enlist a number of compositeServiceAgents to provide a particular service. Before making a final purchase, userAgent can expect the compositeServiceAgents to engage in a negotiation among themselves to select the best offer, perhaps with the userAgent serving as an auctioneer.

An even more compelling scenario is if userAgent can absorb the composite service directly as part of userAgent's intelligence. Because modules and services are relatively autonomous in a BOD agent, this requires only that the designer append the composite service's arbitration to userAgent's reactive plan hierarchy. Thus, in Mojda's case, the userAgent might be given a higher-level goal: "Get Mojda somewhere nice as soon as possible without overdrawing her checking account," but be given very little plan or module support for how to succeed at this task. userAgent might then access the Web and find a composite service that can plan trips. It would then absorb the functionality of the composite service plan into its own ontology—its own goal and plan structure.

The advantage of incorporating the composite service as part of userAgent is that it gives userAgent a finer granularity of control. For example, userAgent might discover prices available at multiple sites and hold transactions open in each of them before making a decision about which to terminate and which to accept. A userAgent seeking the cheapest possible vacation might exploit two composite services, "Buy me the cheapest ticket" and "Rent me the cheapest accommodation." The now-augmented userAgent might be able to intervene in the workings of each composite service, altering and pruning each service's search space in light of the information gleaned from the other.

The advantages are even greater if userAgent's designer can encode userAgent in the same formalism as the composite services. In this case, if the userAgent has the capability to test or reason about its own plan structures, it will be able to evaluate composite services in these same terms. Informed and possibly even secure choices among Web service structures would be possible.

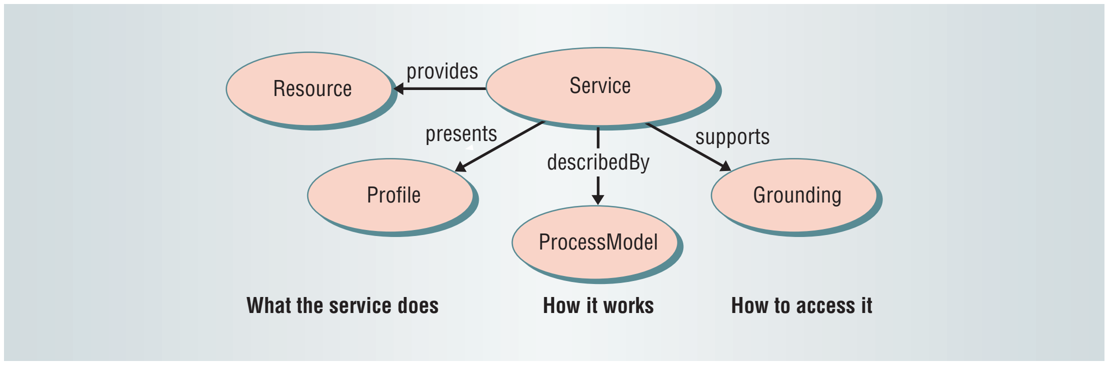

Figure 1. Upper level of the DAML-S service ontology. To automate Web-based service provision and use, the ontology organizes service descriptions into three categories.

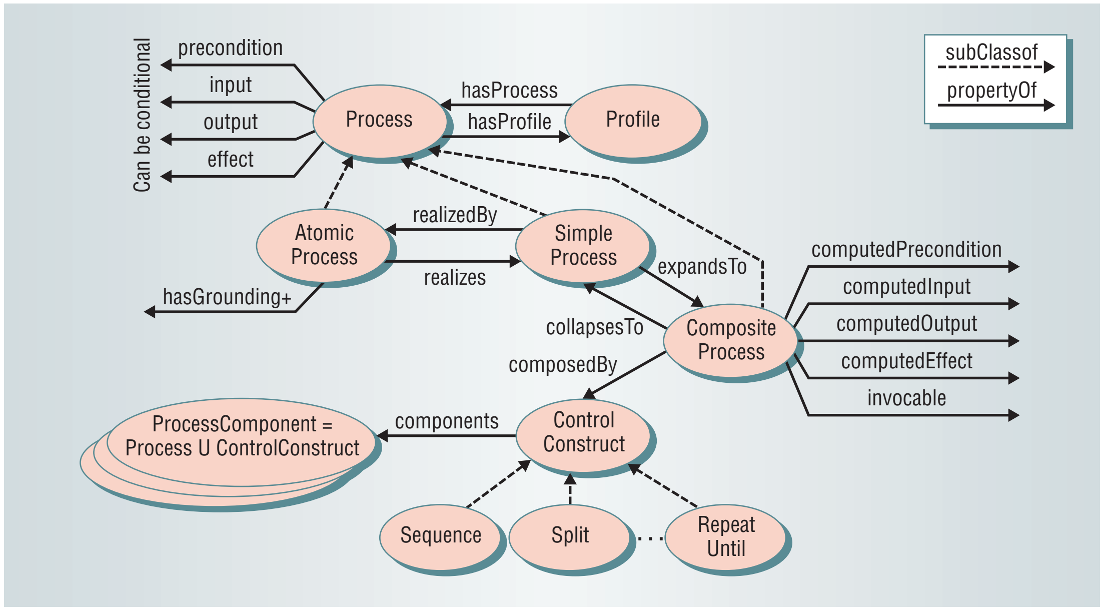

Figure 2. DAML-S process model. The model includes three types of processes: atomic, simple, and composite.

Describing services

As Figure 1 shows, at the highest level, DAML-S aims to provide more opportunity to automate all aspects of Web-based service provision and use. To meet that goal, it organizes a service description into three conceptual areas:[6]

The *profile* describes what the service does. It characterizes the service for advertising, discovery, and matchmaking—the kinds of information service-seeking agents need.

The *process model* tells how the service works, including information about the service's inputs, outputs, preconditions, and effects. It is also important in composing and monitoring processes.

The *grounding* tells how an agent can access a service. Typically, it specifies a communications protocol and provides details such as port numbers used in contacting the service.

Once it selects a service, the agent uses its process model and grounding to build a message sequence for interacting with the service. The profile and process model are abstract specifications, in that they do not commit to any particular message format, protocol, or Internet address. The grounding provides the concrete specification of these details.

Since the process model specifies the service behavior, it is the foundation of intelligent Web services and is of most interest to BOD. As Figure 2 shows, the model includes three types of processes:

- *Atomic* processes are the units of invocation. From the service requester's view, an atomic process executes and returns in a single step.
- *Simple* processes are like atomic processes in that the requester perceives them as having single-step executions. Unlike atomic processes, however, the requester cannot invoke them directly, and they are not associated with a grounding. Simple processes provide abstract views of atomic or composite processes.
- *Composite* processes are constructed from subprocesses, which can be either atomic, simple, or composite. Designers use *control constructs* to specify the structure of a composite process.

Control constructs are compound structures, usually composed of conditions and *process components*, which in turn can be either processes or control constructs. For example, the control construct If-Then-Else contains a condition and two subprocesses, one of which executes when the condition is true and the other when the condition is false.

EXTENDING DAML-S

Viewing the semantic Web as containing intelligence rather than just knowledge provides a new perspective for semantic Web languages like DAML-S.

We propose a number of DAML-S extensions, and we believe that these insights apply to other semantic Web ontologies as well. We also believe that encoding both agents and services in the same way is the easiest and most thorough way to achieve our vision. Consequently, these recommendations are for supporting userAgents as well as conventional Web services.

Data

Data is key to modularity and agent design, yet it is not currently part of the DAML-S ontology. Some data is an integral part of an agent itself such as the agent's current decision history or its progress in a search. Other data is maintained in service modules, such as airline ticket prices or even the userAgent's itinerary. Data recovery characteristics, particularly after failures or crashes, should be indicated using a functional attribute of the DAML-S service profile.

Primitives

Real-time system primitives can behave in two ways. A primitive can compute and return an answer, taking an arbitrarily long time to complete. Or a primitive can trigger a process to run, returning only success or failure in starting the process, leaving the calling program to check whether the process has completed and for any results.

BOD currently uses a hybrid of these approaches. Technically it uses the former (blocked) primitive call, but it expects the module to have an answer ready immediately. Under BOD, behavior modules normally provide an anytime response.[13] Basic services in DAML-S should specify their expected return time and values, possibly guaranteeing time-outs if requested.

Sequences

A sequence's behavior depends on the nature of its primitives. BOD's action selection has two sequence types: a trigger sequence, which expects extremely rapid responses from all its elements and executes within a single planning cycle, and an action pattern, which allows for context-checking and reallocation of control priority between every element. Both sequence types abort if one of their elements returns a failure.

DAML-S includes a sequence subtype, but it does not indicate whether sequences can be interrupted. Also, in DAML-S, sequence elements themselves can be subprocesses (simple or composite), which implies that the sequence type can only be slow, like a BOD action pattern. DAML-S should incorporate atomic sequencing like BOD's trigger sequences and specify conditions and mechanisms for premature termination.

A basic reactive plan provides an organized way to apply a subset of actions until the module achieves a goal.

Basic reactive plans

Often in a dynamic environment, action selection is too nondeterministic for sequences to direct it. Nevertheless, focusing on a particular subset of an action repertoire is a more efficient and effective way to complete tasks. A basic reactive plan provides an organized way to apply a subset of actions until the module achieves a goal. DAML-S does not currently support the expression of a BRP directly, although developers can build BRPs from a repeat-while statement and cascading If-Then-Elses. For clarity, though, DAML-S should support BRPs directly as a composition construct.

Agent-level control

The root of a BOD reactive plan hierarchy is a set of goals that a module checks every time step. A real-time agent requires such a mechanism for monitoring its environment (including itself) to determine whether one of its high-level goals has become urgent and should determine its current intentions. BOD's action selection uses an extended version of the BRP for this purpose.[9,12] DAML-S should also include an agent-level control construct.

The Web is already a storehouse of behavioral intelligence that provides users, at browsers, with knowledge and abilities. While sitting at their desks, users can move people, objects, and information across the planet. The semantic Web will empower users even more. They will be able to exploit intelligence agency, as well as real services, from their desktops. ■

Acknowledgments

We thank the researchers of the Knowledge Systems Laboratory of Stanford University and the DAML group at SRI International, particularly Richard Fikes of KSL and Jerry Hobbs of SRI. Thanks also to Srini Narayanan of ICSI, Mark Burstein of BBN, and Mark Humphrys of Dublin City University.

This article reports effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreements F30602-01-2-

0512, F30602-00-2-0579-P00001, and F30602-00-C-0168. The views and conclusions are the authors' and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, or the US government.

Additional content related to this article is available in "Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web" by J.J. Bryson et al.[3]

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, 2001, pp. 34-43.
2. S.A. McIlraith and T.C. Son, "Adapting Golog for the Composition of Semantic Web Services," *Proc. 8th Int'l Conf. Knowledge Representation and Reasoning*, Morgan Kaufmann, San Francisco, 2002, pp. 482-493.
3. J.J. Bryson et al., "Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web," *Web Intelligence*, N. Zhong, J. Liu, and Y. Yao, eds., Springer-Verlag, Berlin, to be published, 2002.
4. P. Ciancarini and M.J. Wooldridge, eds., *Proc. 1st Int'l Workshop Agent-Oriented Software Eng.*, Lecture Notes in Computer Science 1957 (LNCS 1957), Springer-Verlag, Heidelberg, 2001.
5. J.J. Bryson and L.A. Stein, "Modularity and Design in Reactive Intelligence," *Proc. 17th Int'l Joint Conf. Artificial Intelligence*, Morgan Kaufmann, San Francisco, 2001, pp. 1115-1120.
6. A. Ankolekar et al., "DAML-S: Semantic Markup for Web Services," *Proc. 1st Semantic Web Working Symp.*, I.F. Cruz et al., eds., http://www.semanticweb.org/SWWS/program/full/SWWSProceedings.pdf.
7. J. Hendler and D.L. McGuinness, "DARPA Agent Markup Language," *IEEE Intelligent Systems*, vol. 15, no. 6, 2001, pp. 72-73.
8. G. Denker et al., "Accessing Information and Services on the DAML-Enabled Web," *Proc. 2nd Int'l Workshop Semantic Web*, S. Staab et al., eds., WWW10 Limited, Hong Kong, 2001, pp. 67-78.
9. J.J. Bryson, "Cross-Paradigm Analysis of Autonomous Agent Architecture," *J. Experimental and Theoretical Artificial Intelligence*, vol. 12, no. 2, 2000, pp. 165-190.
10. S.A. McIlraith, T.C. Son, H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001, pp. 46-53.
11. E. Gat, "Three-Layer Architectures," *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, D. Kortenkamp, R.P. Bonasso, and R. Murphy, eds., MIT Press, Cambridge, Mass., 1998, pp. 195-210.
12. J.J. Bryson and L.A. Stein, "Architectures and Idioms: Making Progress in Agent Design," *Proc. 7th Int'l Workshop Agent Theories, Architectures, and Languages,* C. Castelfranchi and Y. Lesperance, eds., Springer-Verlag, Berlin, 2001, pp. 15-30.
13. T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," *Proc. 7th Nat'l Conf. Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, Calif., 1988, pp. 49-54.

Joanna J. Bryson *is a faculty member in the Department of Computer Science at the University of Bath. Her research interests are understanding natural intelligence and software engineering. She received a PhD in computer science from MIT. Contact her at jjb@cs.bath.ac.uk.*

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

Sheila A. McIlraith *is a senior research scientist in Stanford University's Knowledge Systems Laboratory and 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 a PhD in computer science from the University of Toronto and is a founding member of the DAML Services Coalition. Contact her at sam@ksl.stanford.edu.*

Lynn Andrea Stein *is a professor of computer science and engineering, director of the Computers and Cognition Laboratory, and a founding faculty member of the Franklin W. Olin College of Engineering, where she is principal investigator of Olin's DAML project. Her research interests include logic, software agents, cognitive robotics, object-oriented programming, and computer science education. She received a PhD in computer science from Brown University. Contact her at las@olin.edu.*