Semantic Web Services as Behavior-Oriented Agents

Joanna J. Bryson, David Martin, Sheila A. McIlraith and Lynn Andrea Stein

Abstract

Many researchers are working towards the goal of a *semantic* Web — a Web that is unambiguously computer interpretable, and thus very accessable to artificial intelligence. A semantic Web would allow artificial agents to do the work of finding and using services required by humans or organizations. We propose that the semantic Web's contents should be regarded as *intelligence*, not just knowledge. The services encoded on the Web provide behavior, which can be used either as extensions of the user-owned agents attempting to exploit the services, or as independent, collaborative agents that can be 'awakened' to assist the user agents. We draw on our experience in developing intelligent agents to describe how DAML-S, a Web service ontology, can be extended to support this vision.

Introduction: Intelligence and the Semantic Web

The World-Wide Web has revolutionized the communication of information between humans. However, realizing the Web's full potential will require more than rapid access to information and services. It will require support for and the development of intelligent schedulers, planners and searchers that, with minimal direction, can serve as an omnipresent staff of advisers, secretaries, agents, brokers and research assistants. We want agents without issues or personal lives to plan everything from vacations to colloquia, product development cycles to birthday parties. In short, we want competent artificial agents to do the diverse mundane tasks in our lives.

The explosion of electronic commerce has brought this vision tantalizingly near. The community of Web-accessible businesses and organizations, as well as the general public, have done the hard work — they have connected an enormous variety of products and services to the Internet, making them accessible to other computer programs via simple communication protocols. Now the artificial intelligence (AI) community is squarely on the spot. In order for intelligent agents to use the Web as it stands, they must understand ordinary language Web pages and presumably a host of other cultural protocols which reduce the ambiguity rampant in such language.

There is another solution: changing the Web in order to make it accessible to existing AI modeling and reasoning techniques. This new "semantic" version of the Web would consist of pages marked up in accordance with standardized conventions in order to reduce ambiguity and facilitate automated reasoning [McIlraith et al., 2001, Berners-Lee et al., 2001]. This approach has been the focus of a large number of research initiatives, which has lead to the prediction that "Soon it will be possible to access Web resources by content rather than just by keywords" [Ankolekar et al., 2001, p. 411].

We believe the semantic Web should not only be about ways to type, flag and advertise the information on the Web, but about how to make the Web actively usable. We should expect the semantic Web not just to extend the *knowledge* of artificial assistants, but to extend their *intelligence*. This is because a service is not just information, it is *behavior*. And behavior can be viewed as the fundamental attribute of intelligence [Turing, 1950].

This article describes how and why we might achieve this ideal. We will focus on building such a web using DAML-S [Ankolekar et al., 2001], a DAML+OIL ontology designed by the DARPA Agent Markup Language (DAML) Services Coalition, to describe the capabilities of Web services. DAML+OIL [Hendler and McGuinness, 2001, DAML+OIL, 2000] is an AI-inspired ontology language, developed through a joint European Union and North American intitiative. It provides semantics and further expressive power to the Extensible Markup Language (XML) and the Resource Description Framework (RDF).

Definitions: Agents and Services

Before we explore the ways in which Web services can be considered part of an agent's intelligence, we need to define these terms. For the purpose of this article, we will define a *Web service* as a Web-accessible program or device. Such programs and devices may or may not be capable of affecting the real world. Services may either be free or for sale. Examples of services are an airline selling a ticket for a flight, a search engine performing a Web search for a set of keywords, a software company providing a patch to fix a program, a police force sending an officer to check a house, or a post office printing an email and delivering it as surface mail. Any of these services might be solicited via the Web.

Composite services combine predefined constituent services in a way that adds value to the user. An example of this from conventional business is a travel agency, where a customer specifies a sort of trip required and the travel agent selects or assists in selecting specific providers such as an airline or hotel.

We use the term *agent* to describe any (relatively) autonomous actor with sets of:

- *goals*, conditions the agent works to achieve or fulfill,
- *intentions*, goals and subgoals the agent is currently engaged in pursuing,
- *beliefs*, knowledge about the world (which is necessarily limited and possibly inaccurate), and
- *behaviors*, actions the agent is able to take.

Agents are generally perceived as consumers of services. However, the critical insight behind our proposal is that the results of employing a service can also be seen as *actions* that the agent might take to achieve its goals. This sort of reasoning is unusual because people are the archetypal agents, and our sense of identity does not typically extend to include behaviors not performed directly by our organism (though see [Clark, 1996]).

Bringing Services onto the Semantic Web

The semantic Web vision begins with information, and particularly with information discovery [Berners-Lee et al., 2001]. There are inherent limitations of keyword-based search over unstructured material expressed in natural languages. The potential benefits of a Web on which many or most pages present their content, or meta-descriptions of their content, in a structured, semantically grounded language such as DAML+OIL, drawing on shared, publicly accessible ontologies, are enormous. Not only does such a language permit precise handling of information-retrieval searches and more elaborate types of queries over Web content, but it also opens the door to powerful forms of *reasoning* about that content [Denker et al., 2001].

The potential of the semantic Web goes well beyond information discovery and querying. It encompasses the automation of Web-based services as well. Many different kinds of interactions with Web sites can be conceptualized as services. While DAML-S is meant to accommodate this full range of interactions, its primary focus has been on Web sites that go beyond the provision of static information to effect some action or change in the world, such as the sale of a product or the control of a physical device. At the highest level, the goal of DAML-S is to unlock this activity-based potential of the Web, by maximizing opportunities for effective automation of all aspects of Web-based service provision and use.

DAML-S currently organizes a service description into three conceptual areas: the *profile*, the *process model*, and the *grounding* [Ankolekar et al., 2001, DAML-S, 2001]. The DAML-S profile describes *what the service does*. It characterizes the service for the purposes of advertising, discover and matchmaking — it gives the kinds of information service-seeking agents need. The DAML-S process model tells *how the service works*. It includes information about the service's inputs, outputs, preconditions and effects. The DAML-S grounding tells *how the service is used*; that is, it specifies the details of how an agent can access a service[1]. Typically a grounding may specify some well known communications protocol (e.g. RPC, CORBA IDL, Java remote calls, KQML), and service-specific details such as port numbers used in contacting the service.

The service profile is the primary construct by which a service is advertised, discovered, and selected, although in some cases an agent involved in discovery or selection may 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. The process model is equally important for purposes of composing and monitoring processes. The profile and process model are abstract specifications, in the sense that they make no commitments regarding message formats, protocols, and Internet addresses needed to interact with an actual instance of a service. The grounding provides the concrete specification of these details.

Since the process model specifices the behavior of the service (see Box for details) it is the main focus of our vision of Web-based intelligence.

>>>> BOX (DAML-S Processes) About Here <<<<

Developing the Semantic Web with Agent-Oriented Design

One critical problem for the semantic Web is this: who will build it? To be truly a part of the Web, the semantic Web must be open to development by enormous numbers of people with diverse programming skills. One solution is to take an agent-oriented aproach. Agent-oriented software engineering is successful because people (including programmers) are better at reasoning about activities when they represent them in terms of human-like actors ascribed with beliefs, intentions, and abilities [Ciancarini and Wooldridge, 2001]. Nevertheless, building complex agents able to arbitrate between conflicting goals and among multiple, mutually-exclusive means to a single end is still not a trivial task.

In this section we describe two fundamental requirements for building such agents. We describe them in terms of a software design methodology called Behavior-Oriented Design (BOD) [Bryson and Stein, 2001b]. BOD is an instance of one of the currently dominant approaches to agent design: the hybrid between modular, behavior-based systems and reactive planning [Gat, 1998, Bryson, 2000]. BOD agents consist primarily of a number of modules which directly control all of a BOD agent's behavior (action, perception and learning). BOD differs from other hybrid architectures in maximizing the power and autonomy of the behavior modules, and reducing the role of the plans to arbitrating between

[1]This use of the term 'grounding' in this context is somewhat unfortunate. DAML-S grounding has nothing to do with semantic grounding.

modules in the case of conflicts for resources. The primitives of the reactive plans are a method-based interface to the behavior modules.

Let's consider the issues of modularity and modular coordination in more detail.

Modularity

Modularity is a key technique for simplifying software. A complex program is decomposed into a number of relatively simple modules, which can be developed and debugged independently. This strategy underlies object-oriented design (OOD) [Coad et al., 1997]. Modularity also underlies the behavior-based approach to AI (BBAI) (e.g. [Matarić, 1997]), which has become at least part of many dominant agent architecture paradigms [Gat, 1998, Bryson, 2000].

Although modularity generally simplifies design, it also creates two design problems. The first is module decomposition, and the second is coordination between modules. The questions of modular decomposition include:

- how many modules should be used, and
- what resources belong in each?

These questions are exacerbated by the common design issue that the total capacities of the final system cannot generally be fully anticipated, because the real requirements for a system are seldom fully understood before the system is built and used. Fortunately, we can borrow solutions developed in OOD, which has been subject to a great deal of research and experimentation in the last two decades. OOD suggests that a program should be decomposed along the lines of the variable state it will need to maintain. In software engineering, state is the heart of an object, while its methods (procedures) are the actions of the program that incorporates it. These methods either depend on or maintain the state in the object.

One of the chief insights of BOD is that the same reasoning applies to artificial agents. Although the main criteria for judging a behavior module in an intelligent agent is its expressed actions, those actions must be supported by both perception and memory. Perception combines input from outside the agent (sensing) with expectations composed of knowledge and beliefs to determine how and when actions should be expressed. These are wrapped around the heart of the behavior, its variable state. A semantic Web agent might be designed around critical state such as knowledge of private resources (e.g. money, CPU, human clients), information about public resources (e.g. prices elicited from other Web services), and knowledge critical to the current transaction (e.g. the length of time until the current transaction times out, the URL of the current vendor).

Behavior decomposition can be determined by this underlying state and its rate of change. Once a necessary piece of state has been identified (e.g. a bank balance), expressed actions dependent on that state, and sensing actions which maintain the accuracy of that state, can be clustered into a single behavior module.

Coordination

The second problem of distributing intelligence into modules is that these different modules may attempt to execute mutually exclusive actions simultaneously, as in conflict over the use of a limited resource. For example, a distributed artificial travel agent may price a large number of plane tickets at the same time, but only one process should be allowed to actually complete the transaction and purchase the tickets if only one person is going to be doing the flying.

The currently dominant way to arbitrate behavior-based modular systems is to incorporate hierarchical reactive plans into the system execution [Gat, 1998, Bryson, 2000]. 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-ends reasoning. Reactive plans are pre-established structures which support the look-up process. Hierarchical reactive plans are simple, robust plans, each element of which may itself be another reactive plan.

BOD includes a specification for Parallel-rooted, Ordered Slip-stack Hierarchical (POSH) reactive plans [Bryson and Stein, 2001b]. POSH plans include two relatively generic reactive-plan idioms, which are found in a number of architectures [c.f. Bryson and Stein, 2001a]. These are the simple sequence and the basic reactive plan (BRP). (See Box .)

>>>> BOX (Basic Reactive Plans) About Here <<<<

Web Services as Agent Behavior

Is there an advantage to thinking about Web services from the perspective of agent-oriented software engineering? We believe so, for several reasons:

1. Web services are analogous to modular behaviors, thus
2. a great deal of research in coordinating modular intelligence is applicable to developing composite services, and
3. the programming techniques of agent-oriented software engineering could make the development of the semantic Web easier.

Services as Behavior Modules

An important characteristic of a service is that it is a black box as far as any client agent (human or artificial) is concerned. The black box may have knobs and switches (e.g. to choose a date), but the service's underlying decisions and workings cannot be changed by the agent.

From the agent-oriented perspective then, services are essentially behavior modules. They contain encapsulated state, such as information about pricing or definitions. They provide the overall agent with perception primitives, such as 'available?', 'expensive?' or 'transaction successful?', and action primitives such as, 'request phone call', 'purchase', or 'calculate'. And finally, they control the details of how those perceptions and actions are computed.

Composite Services as Module Coordination

If Web services are modular behaviors, then composite services such as those provided for in DAML-S [McIlraith and

Son, 2001] may be seen as a form of coordination. A composite service effectively creates reliable, uniform, higher-level subgoals, together with a specification of services that may (partially) achieve these subgoals, thus again simplifying the reasoning of the core agent. For example, a composite service might allow a user to say “Buy me the cheapest ticket from here to France” instead of “Purchase AcmeAir Flight 309 Date 20th November”.

Program, Agent, or Multi-Agent System?

A composite service might be viewed as a conventional program, or, more essentially, as another, more powerful service. However, we propose that there is an advantage to thinking of a Web service as either an agent *in itself*, or even as a *part of* an agent — an extension that could be added to an existing agent that finds and adopts it.

As an agent, a composite service will work autonomously to complete its goals (in the example above, finding a cheap ticket). As a part of an agent, a composite service might be accessed via the Web by another agent with higher level goals (e.g. “Get me somewhere nice as soon as possible without spending more than I have in my checking account.”) In the former case, an agent serving a user (a userAgent), might discover and enlist a number of compositeServiceAgents to provide a particular service. Before making a final purchase, the userAgent may expect the compositeServiceAgents to engage in a negotiation to select the best offer, perhaps with the userAgent serving as an auctioneer. In the latter case, the userAgent would absorb the functionality of the composite service plan into its own ontology — its own goal and plan structure. This would allow the userAgent to enhance its own abilities while maintaining a fairly strict control over what processes get activated in its name, and what the current priority structure should be.

The advantage of incorporating the composite service as *part* of the userAgent is that it gives the userAgent a finer granularity of control. For example, a 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. Consider the case of a userAgent seeking the cheapest possible vacation. The userAgent might exploit two different composite services, one to “Buy me the cheapest ticket”, and the other to “Rent me the cheapest accommodation”. The now augmented userAgent might be able to intervene in the workings of each composite service, altering and pruning the search space in the light of information gleaned from the other.

There are particularly strong advantages to this model if the userAgent itself can be encoded in the same formalism as the composite services. In this case, and if the userAgent has the capability to test or reason about its own plan structures, then it will be able to evaluate composite services in these same terms. This would allow informed and possibly even secure choices between Web service structures.

>>>> BOX (Implications for DAML-S) About Here <<<<

Summary

This article describes a new vision for the semantic Web — that it should be seen not merely as a repository of *knowledge*, but also of *behavioral intelligence*. We argue this for two reasons:

1. because the semantic Web is based on the concept of *services*, and services *are* intelligent behaviors, and
2. because the semantic Web must be built by programmers, and agent-oriented software engineering is an intuitive way to build behavioral intelligence.

We have given an overview of the current content of DAML-S, a developing means for describing services on the semantic Web. We have also described a standard program decomposition or architecture for software agents, consisting of modular behaviors arbitrated by reactive plans. By combining an agent-based outlook with DAML-S, we have the potential to provide a truly grounded ontology for the semantic Web — one based on action in the real world.

Acknowledgments

The authors would like to thank the researchers of the Knowledge Systems Laboratory of Stanford University and the DAML group at SRI International, particularly Richard Fikes (KSL) and Jerry Hobbs (SRI). Thanks also to Srini Narayanan (SRI), Mark Burstein (BBN) Mark Humphrys (Dublin City University), and especially Terry Payne (CMU Robotics Institute) for allowing us to reproduce his artwork in Figure 1.

This article reports effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreements numbered F30602-01-2-0512, F30602-00-2-0579-P00001 and F30602-00-C-0168. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, Katia Sycara, and Honglei Zeng. DAML-S: Semantic markup for web services. In Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuiness, editors, *The Proceedings of the First Semantic Web Working Symposium (SWWS ’01)*, pages 411–430, Stanford, July 2001. The DAML Services Coalition.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

Joanna J. Bryson. Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190, 2000.

Joanna J. Bryson and Lynn Andrea Stein. Architectures and idioms: Making progress in agent design. In C. Castelfranchi and Y. Lespérance, editors, *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*. Springer, 2001a.

Joanna J. Bryson and Lynn Andrea Stein. Modularity and design in reactive intelligence. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1115–1120, Seattle, August 2001b. Morgan Kaufmann.

Paolo Ciancarini and Michael J. Wooldridge, editors. *First International Workshop on Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

A. Clark. *Being There: Putting Brain, Body and World Together Again*. MIT Press, Cambridge, MA, 1996.

Peter Coad, David North, and Mark Mayfield. *Object Models: Strategies, Patterns and Applications*. Prentice Hall, 2nd edition, 1997.

DAML-S. http://www.daml.org/services/, 2001.

DAML+OIL. http://www.daml.org/language/, 2000.

Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49–54, Saint Paul, Minnesota, USA, August 1988. AAAI Press/MIT Press. ISBN 0-262-51055-3.

Grit Denker, Jerry Hobbs, David Martin, Srini Narayanan, and Richard Waldinger. Accessing information and services on the daml-enabled web. In *Proceedings of the Second International Workshop Semantic Web (SemWeb'2001)*, 2001.

Erann Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. http://www-aig.jpl.nasa.gov/public/home/gat/aero97.html, 1998.

James Hendler and Deborah L. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, 15(6):72–73, 2001.

Maja J. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2/3): 323–336, 1997.

Sheila A. McIlraith and Tran Cao Son. Adapting golog for programming in the semantic web. In *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 195–202, 2001. in press.

Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, October 1950.

BOX: DAML-S Processes

The DAML-S process model is intended to provide a basis for specifying the behavior of a wide range of services, and draws on work in several fields.

As shown in Figure 1, DAML-S includes three types of processes: *atomic*, *simple*, and *composite*:

- *Atomic* processes are the units of invocation. An atomic process executes and returns in a single step, from the perspective of the service requester.
- *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 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 (see Table 1) are used to specify the structure of a composite process.

Control constructs are themselves composite, usually being composed of *condition*s and *process component*s, which in turn can be either processes or control constructs. For instance, 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.

Basic Reactive Plans

The BRP is an elaboration of a simple sequence which allows for reactive response to dynamic environments. This allowance is made by enabling elements of the sequence to be either skipped or repeated as necessary. The elements of the sequence are prioritized, with the ultimate / consummatory element having the highest priority. Each element is also guarded by a precondition that determines whether that element can execute. On each program cycle of the behavior-arbitration module, the highest-priority element of the currently-attended BRP that can execute is executed.

A *BRP step* is a tuple $\langle\pi,\rho,\alpha\rangle$, where π is a priority, ρ is a releaser, and α is an action. A *BRP* is a small set of plan steps $\{\langle\pi_i,\rho_i,\alpha_i\rangle*\}$ associated with achieving a particular goal condition. The releaser ρ_i is a conjunction of boolean perceptual primitives which determine whether the step can execute. Each action α_i may be either another BRP or a more primitive plan element.

The order in which plan steps are expressed is determined by two means: the releaser and the priority. If more than one step is operable, then the priority determines which step's α is executed. If no step can fire, then the BRP terminates. The top priority step of a BRP is often, though not necessarily a goal condition. In that case, its releaser, ρ_1, recognizes that the BRP has succeeded, and its action, α_1 terminates the BRP.

The details of the operation of a BRP are best explained through an example. BRPs have been used to control mobile robots and flight simulators, but for clarity we draw this example from blocks world. Assume that the world consists of stacks of colored blocks, and that an agent wants to hold

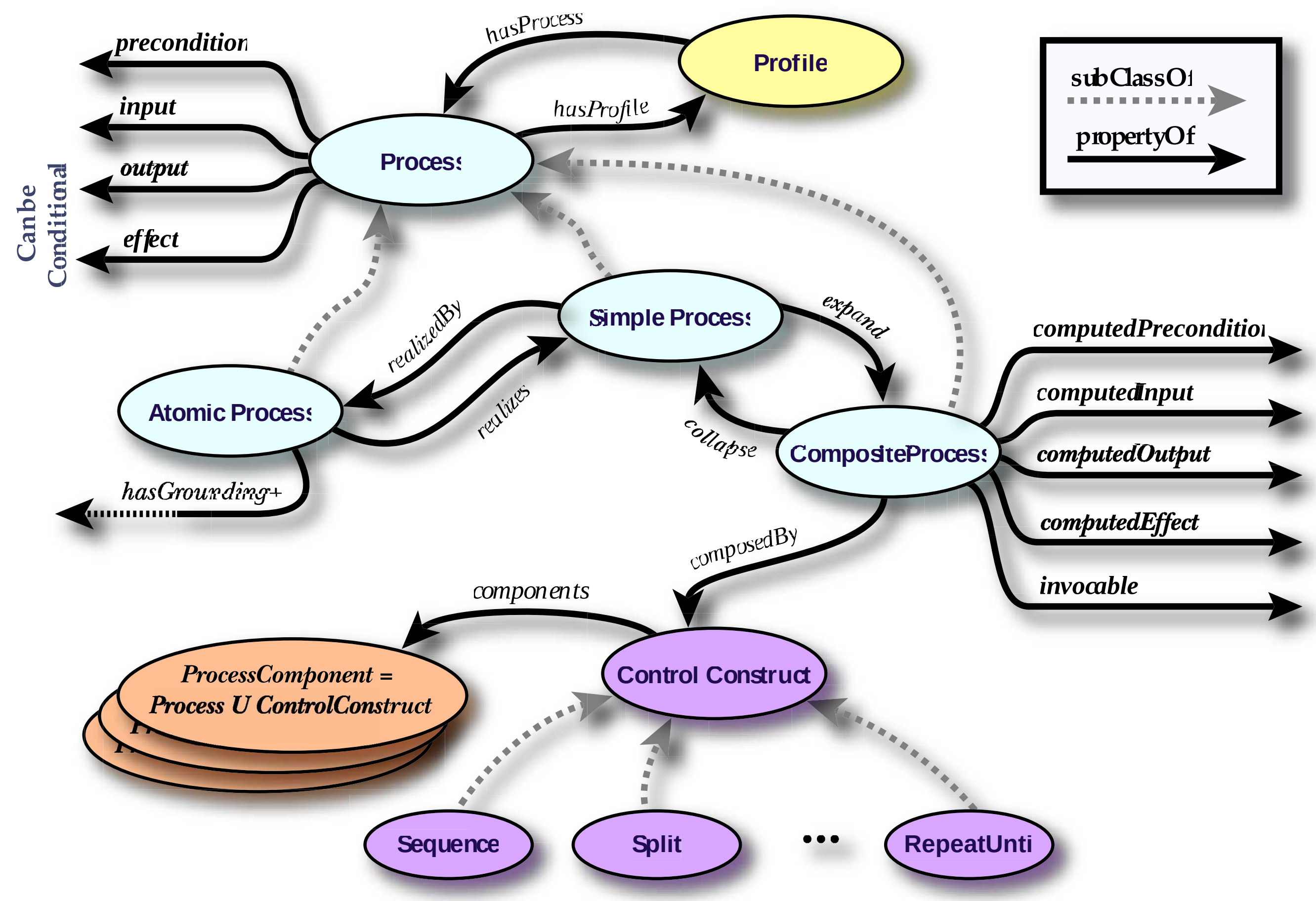

Figure 1: The upper level of the DAML-S process ontology.

a blue block.

$$\uparrow \left\langle \begin{array}{rrl} \underline{\text{Priority}} & \underline{\text{Releaser}} \Rightarrow & \underline{\text{Action}} \\ 4 & \text{(holding) (held 'blue')} \Rightarrow & \textit{goal} \\ 3 & \text{(holding)} \Rightarrow & \text{drop-held, lose-fix} \\ 2 & \text{(fixed-on 'blue')} \Rightarrow & \text{grasp-top-of-stack} \\ 1 & \text{(blue-in-scene)} \Rightarrow & \text{fixate-blue} \end{array} \right\rangle \tag{1}$$

Consider the case where the world consists of a stack with a red block sitting on the blue block. If the agent has not already fixated on the blue block before this plan is activated (and it is not holding anything), then the first operation to be performed would be element **1** because it is the only one whose releaser is satisfied. Once a fixation is established, element **2** will trigger. If the grasp is successful, this will be followed by element **3**, otherwise **2** will be repeated. Assuming that the red block is eventually grasped and discarded, the next successful operation of element **2** will result in the blue block being held, at which point element **4** should recognize that the goal has been achieved, and terminate the plan.

A BRP is robust and opportunistic — it can generate any number of expressed sequential plans. The above paragraph describes the plan 1–2–3–1–2–4. But if the agent is already fixated on blue and fails to grasp the red block successfully on first attempt, the expressed plan would look like 2–1–2–3–1–2–4. If the unsuccessful grasp knocked the red block off the blue, the expressed plan might be 2–1–2–4. If another agent handed our agent a blue block, the expressed plan would just be 4.

Implications for DAML-S

Seeing the semantic Web as containing *intelligence* rather than just *knowledge* provides a new perspective on existing semantic Web protocols like DAML-S. The following are recommendations for DAML-S — similar insights would apply to other semantic Web ontologies.

- *Data:* Data is not a part of the DAML-S specification, but it is key to modularity and agent design. Some data is an integral part of an agent itself and must therefore be stored by it. Examples include the agent's current decision history or data on its progress in a search. State local to the agent should be encapsulated in a module accessible only to the userAgent, but preserving the structure and interfaces of DAML. This would provide for uniform coding. Also, data recovery characteristics, particularly

Construct	Description
Sequence	Execute a list of processes in a sequential order
Concurrent	Execute elements of a bag of processes concurrently
Split	Invoke elements of a bag of processes
Split+Join	Invoke elements of a bag of processes and synchronize
Unordered	Execute all processes in a bag in any order
Choice	Choose between alternatives and execute
If-Then-Else	If specified condition holds, execute "Then", else execute "Else".
Repeat-Until	Iterate execution of a bag of processes until a condition holds.
Repeat-While	Iterate execution of a bag of processes while a condition holds

Table 1: Control constructs in the DAML-S process upper ontology and their intended semantics.

in the case of failures or crashes, should be made one of the functional attributes of the DAML-S service profile.

- *Primitives* Primitives in a real-time system can behave in two ways. A primitive may compute and return an answer, taking an arbitrarily long time to complete. Or a primitive may trigger a process to run, itself returning only success or failure in starting the process. Checking whether the process completes and for any results are separate actions again performed by the calling program. BOD recommends a hybrid between these approaches: it in practice uses the first (blocked) sort, but expects the module to have an 'answer' ready exist. Normally under BOD, the behavior modules are designed to provide (an *anytime* response [Dean and Boddy, 1988]). Basic services in DAML-S should specify their expected return time and values, possibly guaranteeing timeouts if requested.

- *Sequences* The nature of a sequence depends on the nature of its primitives. BOD's action selection has two sorts of sequences: a *trigger sequence*, which expects extremely rapid responses from all 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 sorts of sequences abort if one of their elements returns a failure. The DAML-S process ontology includes a sequence subtype, but it does not currently determine whether sequences can be interrupted. Also, it allows sequence elements to themselves be subprocesses (simple and/or composite), so this indicates the sequence type is only analogous to the latter, slow type of sequence available under BOD. It may be advantageous for DAML-S to incorporate atomic sequencing. It should also specify conditions and mechanisms for premature termination.

- *Basic Reactive Plans* Often in a dynamic environment, action selection is too non-deterministic to be directed by sequences. Nevertheless, focusing on a particular subset of an action repertoire produces more efficient and effective task completion. This is what BRPs are for. DAML-S does not currently support the expression of a BRP directly, though one can be constructed out of a *repeat-while* statement and cascading *if-then-elses*. For clarity though, the BRP should probably be supported directly as a composition construct in the DAML-S process ontology.

- *Agent-Level Control* In order for an agent to be truly reactive, it needs more than a BRP to reorder steps in a small local plan. There must be a mechanism for monitoring the environment (including the agent itself) to determine if one of its permanent goals has become urgent, and should co-opt influence on the agent's current intentions. An example for a semantic-Web-crawling userAgent might include scheduling constraints, such as noting that a recommendation (or even a train ticket) is due in the next few minutes. Another is an event-triggered change, such as a contact from an associate's userAgent with a new set of constraints (e.g. lunch in 5 minutes with the press) or noticing that a previously-established value has become invalid (e.g. a game has been canceled.) BOD's action selection uses an extended version of the BRP for this purpose [Bryson and Stein, 2001b]. To get maximum advantage from of an intelligent Web, the entire agent should be described uniformly. So agent-level control should also be a control construct in DAML-S.