Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web

Joanna J. Bryson[1], David Martin[2], Sheila A. McIlraith[3], and Lynn Andrea Stein[4]

[1] Harvard Primate Cognitive Neuroscience Laboratory
Cambridge, MA 02138
jbryson@wjh.harvard.edu
[2] SRI International
Menlo Park, CA 94025, USA
martin@ai.sri.com
[3] Stanford University Knowledge Systems Lab
Stanford, CA 94305, USA
sam@ksl.stanford.edu
[4] Computers and Cognition Group
Franklin W. Olin College of Engineering
Needham, MA 02492, USA
las@olin.edu

Abstract. Many researchers are working towards the goal of a *semantic Web* — a Web that provides information in a way that is useful to artificial intelligences. A semantic Web would allow artificial agents to do the work of searching for and organizing services required by humans or organizations. DAML-S is a Web service ontology intended to facilitate the semantic Web by describing the properties and capabilities of Web-available services in an unambiguous, computer-interpretable form. In this paper, we propose that an important new perspective on the semantic Web can be obtained by regarding its content as behavioral intelligence. The services encoded in DAML-S can then be viewed as specifications either for extensions of the user 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 agent development to elaborate the specification, particularly of the process ontology of DAML-S, in order to support this vision.

1 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 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 colloquiums, product development cycles to birthday parties. In short, we want competent artificial agents to do the mundane organizational 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 computer programs via simple communication protocols. Unfortunately, this puts our community squarely on the spot. In order for intelligent agents to use the Web as it now 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 some hope that parsing ordinary Web pages might succeed in sufficiently stereotypical transactions: see for example Koller [31] for a discussion of the information beyond natural language available in Web pages.

In this paper, we focus on an alternative approach: changing the Web in order to make it accessible to sophisticated modeling and reasoning techniques developed in AI and related disciplines. 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 [4, 43]. This approach has been the focus of a large number of research initiatives [17], which has lead to the prediction that "Soon it will be possible to access Web resources by content rather than just by keywords" [1, p. 411].

Ankolekar et al. [1] describe a formalism, DAML-S, designed by the DARPA Agent Markup Language (DAML) Services Coalition to facilitate accessing services via the Web. DAML is an extension of the Extensible Markup Language (XML) and the Resource Description Framework (RDF). It is designed to provide for better specifications of relationships and ontologies within Web pages to facilitate their automated parsing and thus the intelligent use of data present on the Web. DAML-S is the part of this language and ontology effort dedicated to supporting Web services [18].

This paper is intended to extend and refine this work. We propose that in designing the semantic Web, we should be thinking not only 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[1].

In this paper, we review the fundamentals of constructing an intelligent agent. We focus on modular, reactive approaches to agent design, because they are conducive to the highly distributed, complex nature of Web intelligence. We then examine how to implement our proposals in DAML-S. Using the DAML-S process ontology formalism means that reasoning and proof-checking can be applied by concerned agents over the outcome of extending their design via Web services [see e.g. 41]. The main contribution of this paper is a set of recommendations for the future design of the DAML-S process ontology, in which we will encode the planning and action selection for our proposed semantic Web intelligence. We begin with some definitions, followed by an overview of current work on DAML-S and agent-based software engineering, in particular Behavior-Oriented Design [11].

[1] That expressed behavior is both the purpose and the criteria for intelligence is a functionalist claim, in keeping with "New AI" [38] and with Turing [58], though not with Newell and Simon [47].

2 Definitions: Agents and Services

Before we discuss whether Web services should be considered a part of an agent's intelligence, we should define what these terms mean. For the purpose of this paper, we will define a *Web service* as a Web-accessible program or device. Such programs and devices may well be capable of affecting the real world, although they needn't do so to be regarded as Web services. 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 is somehow more useful to the ultimate end user than the individual services. The *particular* services to be combined are never specified at the composite level, but rather a characterization of desired outcome is provided to the composite which then selects from a number of vendors [22]. 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. McIlraith and Son [41] propose that for the semantic Web, composite services might be viewed "as customizations of reusable, high-level generic procedures. Our vision is to construct such reusable, high-level generic procedures, and to archive them in sharable generic procedures ontologies." This vision has lead directly to the DAML-S process ontology, described below.

An *agent* we will take to be a (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 to our argument is realizing that the results of employing a service can also be seen as *actions* the agent might take to achieve its goals. This sort of reasoning is unusual because humans are the archetypal agents, and our sense of identity does not typically extend to include behaviors not performed directly by our organism (though see [14].)

3 Bringing Services onto the Semantic Web

The semantic Web vision begins with information, and particularly with information *discovery* [4]. The inherent limitations of keyword-based search, over unstructured material expressed in natural languages, are already well recognized. 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, 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 [21]. The primary goal of the DAML program is to provide the most useful markup language for that purpose [28]; the latest result as of this project to date is DAML+OIL [19], an XML-based language grounded in description logic and having a formal underlying semantics. Additional fundamental goals include the design of tools to facilitate the creation and maintenance of DAML+OIL ontologies and markup, and the development of critical supporting technologies, such as automatic translation between ontologies.

The potential of the semantic Web, however, goes well beyond information discovery and querying. In particular, it encompasses the automation of Web-based *services* as well. As mentioned above, 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 do not merely provide static information but allow one 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.

Work on DAML-S takes on increased importance in light of the recent focus on service provision in the commercial world, as exemplified by the development of Web Services Description Language (WSDL) [12] and Universal Description, Discovery and Integration (UDDI) [59]. The high visibility of this commercial work indicates that within the next few years, large numbers of services may well be accessible not just by human browsers, but also by software agents acting on behalf of individual or organizational users. We note here the complementarity of DAML-S technology with this commercial work on Web services. While commercial efforts are currently focused on standardization of registration and look-up mechanisms, platform-independent interoperability, and interchange of syntactically well-defined document types, DAML-S is concerned with providing greater expressiveness in describing the characteristics of services in a way that can be reasoned about, in support of more fully automated service discovery, selection, invocation, composition, and monitoring.

DAML-S organizes a service description into three conceptual areas: the *process model*, the *profile*, and the *grounding*. After introducing these 3 areas, we give further attention to the process model.

– The DAML-S *profile* describes *what the service does*. It characterizes the service for purposes of advertising, discovery, and matchmaking; that is, it gives the kinds of information needed by a service-seeking agent to determine whether the service meets its needs. Service profiles will normally be organized into ontology-based taxonomies, which provide the first level of discrimination in searching for a desired service. The characteristics of description logic embodied in DAML+OIL lend themselves to this sort of classification scheme. See [26] for a (pre-DAML) matchmaking system based on description logic, and [57] for a system directly based on DAML.
  For all classes of services, profiles can include information about the service's inputs, outputs, preconditions, and effects. This information is derived from the pro-

cess model, as described below. In addition, various classes of services can have a wide variety of class-specific attributes specified for them, which provide additional information and requirements. For example, a book selling service might indicate specific categories of books in which it specializes, and a catering service might indicate the kinds of cuisine available and the types of events for which it caters. Other more general qualifiers may also be included, such as professional credentials belonging to the service provider, and quality guarantees, expected response time and geographic constraints (in appropriate domains).

- The DAML-S *process model* tells *how the service works*. It includes information about the service's inputs (with some indication of whether they are required or optional), outputs (possibly with specification of the conditions under which various outputs will occur), preconditions (circumstances that must hold before the service can be used) and effects (what is accomplished by the service, or more generally, changes brought about by the service). For a complex service (one composed of several steps over time), the process model shows how it breaks down into simpler component processes, and the flow of control between them. This description may be used by a service-seeking agent in (at least) four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate its activities with those of the service provider; and (4) to monitor the execution of the service.
- The DAML-S *grounding* tells *how the service is used*; that is, it specifies the details of how an agent can access a service[2]. Typically a grounding may specify some well known communications protocol (e.g., RPC, HTTP-FORM, CORBA IDL, SOAP, Java remote calls, KQML), and service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the ServiceModel, an unambiguous way of exchanging data elements of that type with the service (that is, marshaling / serialization techniques employed).

In summary, the service *profile* is the primary construct by which a service is advertised, discovered, and selected – but 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.

[2] This use of the term 'grounding', particularly within the context of a *semantic* Web is somewhat unfortunate. DAML-S grounding has little to do with *semantic grounding* [27, 35, 54].

3.1 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. This includes work in AI on standardization of planning languages [25], work in programming languages and distributed systems [44, 45], emerging standards in process modeling and workflow technology such as the NIST's Process Specification Language (PSL) [52] and the Workflow Management Coalition effort (http://www.aiim.org/wfmc), work on modeling verb semantics and event structure [46], previous work on action-inspired Web Service markup [42], work in AI on modeling complex actions [34], and work in agent communication languages [24, 39] and Multi-Agent infrastructure[55].

As mentioned above, inputs, outputs, preconditions, and effects are central to the characterization of processes. DAML-S process inputs are named and typed using either DAML+OIL classes or primitive data types provided by XML Schema [60]. Inputs are specified as DAML+OIL properties with appropriate range restrictions. In addition to specifying what information is required by the process, the inputs also (implicitly) specify the contents of the invocation message(s) for the process. Outputs, similarly named and typed, indicate the information that the process provides when its execution completes. Preconditions, of which there may be any number, must all hold in order for the process to be invoked. Similarly, the process can have any number of effects, which are guaranteed to hold true at its completion. Conditions can be associated with outputs and effects. DAML+OIL subclassing can be used to establish a hierarchy of process classes for increasingly specific domains, adding more specific inputs, outputs, preconditions, and effects as appropriate.

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

- *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. (The precise requirements for the message format are given by the atomic process' grounding.) 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 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. Such a view of an atomic process typically indicates a specialized way of using the atomic process. When abstracting from a composite process, a simple process typically provides a simplified representation, for use by planning and reasoning engines that don't need the full details of decomposition. In terms of DAML+OIL properties, a simple process said to be *realizedBy* an atomic process; or to *expand* to a composite process.
- *Composite* processes are constructed from subprocesses, which in turn can be either atomic, simple, or composite. Control constructs such as *Sequence* and *If-Then-Else* are used to specify the structure of a composite process. In addition to describing control flow, this structural specification also shows which of the various inputs of

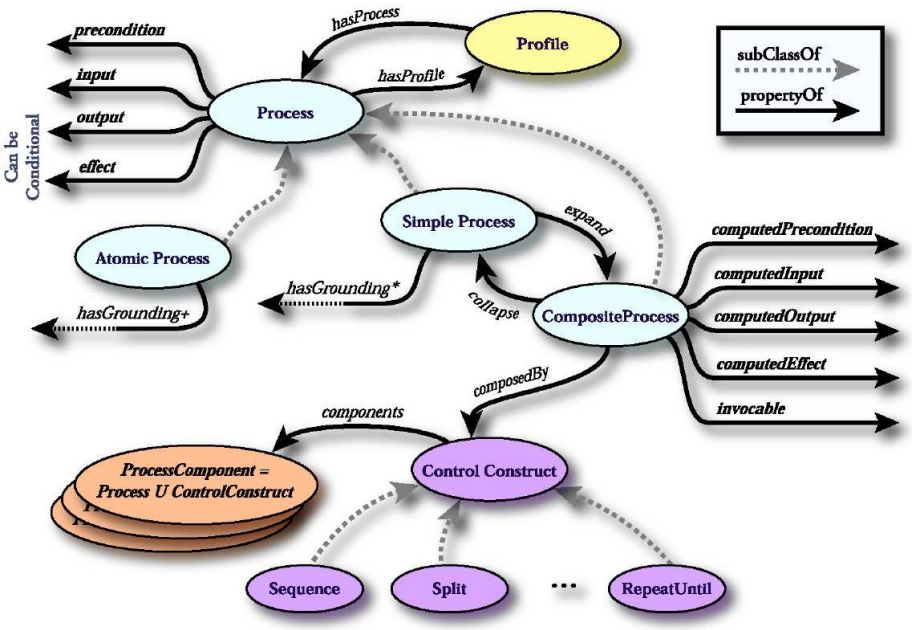

Fig. 1. The upper level of the DAML-S process ontology.

the composite process are accepted by which of its subprocesses (and similarly for outputs).

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 — that is, is related to a condition by a particular property — and two subprocesses, one of which executes when the condition is true and the other when the condition is false. The table in Figure 2 outlines the set of control constructs available in the process upper ontology, and their intended semantics.

It is often useful to view a process at different levels of granularity. A“black box” view allows one to deal with the process as primitive and undecomposable, whereas a “glass box” view allows one to inspect and reason about its control structure and internal data flow. In DAML-S, a simple process can be used to present a black-box view of a composite process, which in turn represents the corresponding glass-box view. This relationship between the simple and composite views of a process can be captured in DAML-S using the inverse *expand* and *collapse* properties.

4 Semantic Web Development and Software Agent Architecture

One critical problem for the semantic Web approach to automating the Web is this: who will build it? To be truly a part of the Web, the semantic Web must be open to

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

Fig. 2. Control constructs in the DAML-S process upper ontology and their intended semantics.

development by an enormous number of people with truly disparate amounts of skill and talent as programmers. This is true even if we assume that not all Web pages will be on the semantic as well as the ordinary Web, but only those representing fairly-well-established companies and organizations.

Part of the success of the agent-oriented approach to software engineering has been the fact that people (including programmers) seem better at reasoning about activities when they represent them in terms of humanoid actors ascribed with beliefs, intentions, and abilities [13]. However, building complex agents able to arbitrate between conflicting goals and between multiple, mutually-exclusive means to a single end is still not a trivial task.

In this section we describe the requirements for building such agents. The concepts introduced here have been described elsewhere more fully as a software design methodology for constructing complex agents called Behavior-Oriented Design (BOD) [8, 11]. BOD is an instance of one of the currently dominant approaches to agent design: the hybrid between modular, behavior-based systems and reactive planning [7, 29, 32]. 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 modules in the case of conflicts for resources. The primitives of the reactive plans are a method-based interface to the behavior modules.

We will now consider the issues of modularity and action selection through behavior arbitration in more detail. In Section 5 we will relate this model of agent intelligence to Web services. In Section 6 we will use consider the ramifications for DAML-S of the requirements of this agent-oriented outlook on Web services.

4.1 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 the current dominant software engineering paradigm, object-oriented design (OOD) [15, 49]. Moving more specifically to

the problem of engineering intelligence, modularity also underlies the behavior-based approach to AI (BBAI) [2, 6, 40], which has become at least part of many dominant agent architecture paradigms [see for reviews 7, 29].

Although modularity in some sense simplifies design, it also creates at least two design problems. The first, module decomposition, will be addressed here, the second, coordination between modules, is addressed in 4.2.

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 [5]. 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. State is the heart of an object, while its methods 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. Things a semantic Web agent might store in memory include knowledge of private resources (e.g. money, CPU, human clients), 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.

4.2 Action Selection

As mentioned in the previous section, the main advantage of modular intelligence is the relative simplicity of the units, which in turn facilitates the engineering of the overall system. The drawback of distributing intelligence is that the different modules may attempt to execute mutually exclusive actions — in particular, they may conflict over the use of a fixed 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.

Early modular AI systems resisted the use of centralized arbitration because this solution was seen as homuncular — as recreating the problem that modularity was intended to address [36]. Instead, behaving modules were expected to recognize their own

context for expression [6, 53]. However, this approach leads to combinatorial problems in a complex agent where multiple behaviors could in theory fire in a similar contexts, and thus each behavior might need to model others' behaviors as well as their own [50]. Further, analysis has shown that arbitration between modules is not as complicated a problem as the monolithic direction of intelligence by a non-modular approach. This is because the behavior modules themselves provide information on their context and applicability, making arbitration between them a relatively simple computation [c.f. 37, 51].

The currently dominant way to arbitrate behavior-based, modular systems is to incorporate hierarchical reactive plans into the system execution [7, 29, 32]. 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 step of which may itself be another reactive plan.

BOD includes a specification for Parallel-rooted, Ordered Slip-stack Hierarchical (POSH) reactive plans. The details of POSH action selection are available elsewhere [8, 11]. In this paper, we focus not on the specifics of BOD's action selection, but on two relatively generic reactive-plan idioms, which are found in a number of architectures [c.f. 10]. These are the simple sequence and the basic reactive plan (BRP).

The BRP is an elaboration of the 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 the 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 more complete description of the BRP is available in Appendix A, below.

5 Web Services as Agent Behavior

Is there really 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 could therefore be applicable to developing composite services, and
3. the programming techniques of agent-oriented software engineering could be generally useful for the development of the semantic Web.

In this section, we discuss each of these points in more detail.

5.1 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) that uses it is concerned. The black box may have knobs and

switches (e.g. to choose a date, location or title), but the service's underlying decisions and workings cannot be changed by the agent.

If we switch to the agent-oriented perspective introduced in Section 2, then what this means is that 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. As with all modular systems, the more powerful these primitives are, the more simple the agent's action-selection can be [37]. But at the same time, there may be costs associated with giving up fine-grain control of action or timing [56].

5.2 Composite Services as Action Selection

If Web services are modular behaviors, then composite services such as are described by McIlraith and Son [41] and provided for by DAML-S may be seen as a form of behavior arbitration, which is in turn a form of action selection. The purpose of a composite service is effectively to create reliable, uniform, higher-level subgoals, thus again simplifying the reasoning of the core agent as it arbitrates between its overall goals. 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".

5.3 Program, Agent, or MAS?

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 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, either between each other or with the userAgent, perhaps serving as an auctioneer. In the latter case, the userAgent might 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's action selection 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 on their discovery. This would facilitate relatively informed and possibly even secure choices between Web service structures, although of course the security would be limited by the behavior of the encapsulated Web services that serve as the ultimate behavior primitives.

6 Implications for DAML-S

In the previous sections, we have discussed motivations for thinking of the contents of the semantic Web as either agents which can be negotiated with, or preferably as agent components which can be incorporated into agents representing individual human users (userAgents). We also briefly described key elements of agent-oriented software engineering, which might make populating the semantic Web more intuitive for programmers.

In this section, we examine the DAML-S process ontology, which, among other things, is designed to support composite services. We discuss its ability to support the design elements key to agent organization.

6.1 Data

Although data is not a part of the DAML-S specification, it is a key issue to modularity and agent design. As should be apparent from both our previous discussion of modular decomposition, and the example of integrating two composite services, one for travel tickets and one for accommodation, some data is an integral part of the 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. In the case where the semantic Web is viewed as a populated MAS, then each agent must maintain its own data about its current transaction state. This is true also of the single agent employing Web services — some data can (and even must) be stored within the underlying Web services. The agent must store more locally information representing not only its decisions, but other information such as characteristics or requirements provided by or about the underlying user. These might include their preferred airlines, their personal schedule, or their security clearance.

One suggestion is that state local to the agent should be encapsulated in a Web-service-like module accessible only to the userAgent, but preserving the structure and interfaces of DAML. This would provide for uniform coding. Thus our first recommendation with respect to data is not so much for a change in DAML-S, but for a strategy of building DAML-S agent behaviors when they are *not* actually Web services.

Another consideration with respect to data is the responsibility for data retention, particularly in the case of failures or crashes. It may be in the interest of the userAgent to

maintain persistent storage and records of information transmitted even if negotiations were not fully committed, in order to save time when the Web-based agent or intelligence becomes available again. Since data belongs to the agent, we do not consider this an important consideration for composite services, unless they are operating as independent agents (the MAS option described above.) As for the Web services themselves, presumably the transaction standards for such situations have already been established for Web-based services. We only recommend that these be made one of the functional attributes of the DAML-S service profile.

6.2 Primitives

Primitives are the lowest-grained individual actions that are performed by the Web service — they are the interface to the black box. This includes sense-acts such as reading a value. We have already discussed primitives in terms of their relationship to Web services. In this section we will describe them as members of the DAML-S process ontology, that is, their behavior in composite services / action-selection plans.

There are fundamentally two ways primitives in a real-time system can behave. The first is that a primitive may compute and return an answer, and take an arbitrarily long time to complete. The second is that a primitive may trigger a process to run, itself returning only success or failure in starting the process. Checking whether the process completes, and, if it is performing a computation, what its result is, are separate actions again performed by the calling program.

BOD actually recommends a hybrid between these approaches: it in practice uses the first (blocked) sort of primitive, but it expects the "answer" to already exist. Normally under BOD, the behavior modules to be designed to provide *anytime* responses [20] to the invocation of a primitive. This is critical in order for a BOD agent to be reactive: it can only switch attention to another information source (such as a successful search return or a user interrupt) if its control is not pending on a function call.

Consequently, we recommend that primitives / basic services in DAML-S specify their expected return time and values, possibly guaranteeing timeouts if requested. Again, this approach is not mandatory: the DAML-S process ontology allows for parallel operation, but it is necessary for any temporal reasoning or guarantees by the overall agent. This is again a recommendation for the Service Profile, that for each possible communication to the service it should specify whether the call will be blocked or unblocked, and whether they have fixed, variable or no timeout.

6.3 Sequences

The sequence is the most fundamental composite element. Some agent architectures try to do away with it in favor of chains of production rules, but these are not equivalent[3] [10, 30]. A sequence provides an extra piece of control state. It specifies a problem space — a particular subset of possible actions or productions — which simplifies the triggering information each element needs. It also allows sequential firing to happen

[3] One of the best established and well researched production-based architectures, Soar, had to be supplemented with sequences when it was applied to real-time systems [33].

independently of perceiving the results of an action, which may be important if the sequencing needs to be faster or more reliable than perception can be.

The nature of a sequence depends on the nature of its primitives. BOD's POSH action selection actually has two sorts of sequences: a trigger sequence, which expects extremely rapid responses from all elements and executes entirely within a single 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. This will usually be the result of a sensory predicate check, but actions may fail sufficiently radically to abort a sequence as well.

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 also atomic sequencing. Further, it should specify conditions and mechanisms for premature termination, possibly including a global (to the sequence) timeout factor.

6.4 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. We have elsewhere identified an idiom, the basic reactive plan (BRP), which handles this sort of situation [10]. It is sufficiently similar to a sequence that it can also be easily designed by conventional programmers. The BRP is applied iteratively. Its elements are prioritized with the highest priority assigned to the steps closest to achieving the goal of the plan. The elements are also guarded by preconditions. On each iteration, the highest priority step that can be executed, is. See BOX A for a more thorough explanation including an example.

In POSH action selection, an element of a BRP may itself be a BRP, thus leading to hierarchical control. This is critical in order to maintain simplicity and clarity. We have found that the optimal number of steps in a BRP tends to be no more than seven. POSH action selection does not maintain a stack in a BRP hierarchy, thus allowing for cycles — that is, an element can be its own descendant. This is explained briefly in Section 6.5, and in more detail by [11].

DAML-S does not currently support the expression of a BRP directly, though one can be constructed out of a while statement and cascading if-thens. However, the idiom is sufficiently powerful and provides sufficient clarity that we feel it would be wise to support it directly as a construct in the DAML-S process ontology.

6.5 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. It must also be able to respond to events which indicate that a complete change in plan may be necessary. 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.)

POSH action section uses an extended version of the BRP for this purpose. On every program cycle for the action-selection module, it first checks the agent's global priorities, then attends to the BRP or sequence currently being pursued by the highest-priority goal. This high-level, extended BRP is called a *drive collection*, and serves as a root of the action-selection hierarchy. Each of its elements keeps track of two things:

1. its root of the action-selection hierarchy, and
2. the composite element / intention it is currently pursuing.

If a composite element (a BRP or a sequence) selects a plan which is itself a composite, that becomes the new intention of the drive collection goal element. On the other hand, if an intention terminates, then the drive collection returns attention to the root of its plan hierarchy. This process, referred to as a *slip stack*, keeps the agent reactive by revisiting its decision history periodically, as well as eliminating the problem of a stack slowing action selection.

As an example, suppose that a userAgent has two goals: reminding the user of meetings, and making arrangements for dinner. Assume the meeting priority is higher, but often its action simply consists of checking whether the time to the next meeting is less than five minutes. Dinner arrangements might be more complex, because they involve determining appropriate companions, locations, food and expense. Suppose the userAgent has gotten sufficiently far in tonight's dinner arrangements that it is currently trying to book a reservation at a particular restaurant, but it is now five minutes before a meeting and the agent must now divert all its resources (or at least its sound card) to ensuring the user is aware of it. When this task is over, the second goal will remember where in its plan hierarchy it was (phone the restaurant.) On the other hand, if the plan has now been invalidated (the restaurant is fully booked), the goal can return to the top of the plan hierarchy, revisit and recheck its decisions, and ultimately select a new plan.

If a compound Web service is to be viewed as an extension of a userAgent, then specifying this highest level of the agent in DAML-S is not strictly necessary. Nevertheless, it might be useful for reasoning about the entire agents plan framework given that we expect the plans to include extensions in DAML-S from composite services found on the semantic Web. If, on the other hand, a new complete agent is to be spawned to represent the composite service, and the composite service is entirely specified in DAML-S, some mechanism must exist in DAML-S for specifying the agent's core, and this one may be desirable. The drive-collection is specifically designed to be continuous with the BRP and therefore also easy to learn to design by ordinary programmers. From the DAML-S implementation side, its similarity to the BRP also makes it a relatively simple addition if the BRP is already present in the ontology. It should however be noted that, unlike the BRP, the drive collection is not currently a widely accepted mechanism for encoding the goals and intentions of an agent.

7 Summary

In this paper, we have argued that the semantic Web 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 (the goal of which is behavioral intelligence) is an intuitive way to build software services.

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. Although we have emphasized a single approach to developing such agents, Behavior-Oriented Design, we have made an effort to motivate all our recommendations in terms of generally- or at least widely-accepted practice.

We have then turned our knowledge of agent-oriented software engineering into a set of specific recommendations about the nature of standardized ontologies for the Web. Perhaps the most critical recommendation is the highest-level one — that DAML-S should be able to support the description of an entire software agent, so that agent intelligence can be continuous with the semantic Web and accessible to the same forms of AI reasoning. We have focused our recommendations on DAML-S, although they could be adapted to other standards if necessary or desirable. 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 for their assistance on this document. Thanks in particular to Richard Fikes (KSL) and Jerry Hobbs (SRI). Thanks also to Srini Narayanan (SRI), Mark Burstein (of BBN) and Mark Humphrys (of Dublin City University). And a very special thanks to Terry Payne of the CMU Robotics Institute for generously allowing us to reproduce his artwork in Figure 1.

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 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

[1] 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.

[2] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[3] Scott Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, December 1996. Department of Computer Science.

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

[5] Frederick P. Brooks, Jr. *The Mythical Man-month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Reading, MA, 20th anniversary edition edition, 1995.

[6] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47: 139–159, 1991.

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

[8] Joanna J. Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, MIT, Department of EECS, Cambridge, MA, June 2001.

[9] Joanna J. Bryson and Brendan McGonigle. Agent architecture as object oriented design. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, pages 15–30. Springer-Verlag, 1998.

[10] 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, 2001.

[11] 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 2001. Morgan Kaufmann.

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

[13] 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.

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

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

[16] Luis Correia and A. Steiger-Garção. A useful autonomous vehicle with a hierarchical behavior control. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life (Third European Conference on Artificial Life)*, pages 625–639, Berlin, 1995. Springer.

[17] Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuiness, editors. *The Proceedings of the First Semantic Web Working Symposium (SWWS '01)*, Stanford, July 2001.

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

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

[20] 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.

[21] Grit Denker, Jerry Hobbs, David Martin, Srini Narayanan, and Richard Waldinger. Accessing information and services on the daml-enabled web. In *Proc. Second Int'l Workshop Semantic Web (SemWeb'2001)*, 2001.

[22] Marlon Dumas, Justin O'Sullivan, Mitra Heravizadeh, David Edmond, and Arthur ter Hofstede. Towards a semantic framework for service description. In *In Proceedings of the IFIP Conference on Database Semantics*, Hong Kong, April 2001. Kluwer Academic Publishers.

[23] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[24] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

[25] Malik Ghallab et. al. Pddl-the planning domain definition language v. 2. Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[26] Yolanda Gil and Surya Ramachandran. Phosphorus: A task-based agent matchmaker. In *Proc. Agents'01*, May 2001.

[27] Steve Harnad. Categorial perception: A critical overview. In Steve Harnad, editor, *Categorial perception: The groundwork of perception*. Cambridge University Press, 1987.

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

[29] Henry Hexmoor, Ian Horswill, and David Kortenkamp. Special issue: Software architectures for hardware agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2/3), 1997.

[30] George Houghton and Tom Hartley. Parallel models of serial behavior: Lashley revisited. *PSYCHE*, 2(25), February 1995.

[31] Daphne Koller. Representation, reasoning, learning. IJCAI Computers and Thought Award talk, August 2001.

[32] David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press, Cambridge, MA, 1998.

[33] John E. Laird and Paul S. Rosenbloom. The evolution of the Soar cognitive architecture. In D.M. Steier and T.M. Mitchell, editors, *Mind Matters*. Erlbaum, 1996.

[34] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A Logic programming language for dynamic domains. *Journal of Logic Programming*, 31 (1-3):59–84, April-June 1997.

[35] Karl F. MacDorman. Grounding symbols through sensorimotor integration. *Journal of the Robotics Society of Japan*, 17(1), 1999.

[36] Pattie Maes. A bottom-up mechanism for behavior selection in an artificial creature. In Jean-Arcady Meyer and Stuart Wilson, editors, *From Animals to Animats*, pages 478–485, Cambridge, MA, 1991. MIT Press.

[37] Chris Malcolm and Tim Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly system. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 123–144. MIT Press, Cambridge, MA, 1990.

[38] Chris Malcolm, Tim Smithers, and John Hallam. An emerging paradigm in robot architecture. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, volume 2, pages 545–564, Amsterdam, 1989. Elsevier.

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

[40] 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.

[41] Sheila 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.

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

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

[44] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[45] Robin Milner. *Communicating with Mobile Agents: The π-Calculus*. Cambridge University Press, Cambridge, 1999.

[46] Srini Narayanan. Reasoning about actions in narrative understanding. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI'1999)*, pages 350–357. Morgan Kaufman Press, San Francisco, 1999.

[47] Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. In John Haugeland, editor, *Mind Design*, chapter 1, pages 35–66. MIT Press, Cambridge, MA, 1981.

[48] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[49] David Lorge Parnas, Paul C. Clements, and David M. Weiss. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, SE-11(3): 259–266, March 1985.

[50] Tony J. Prescott, Kevin Gurney, F. Montes Gonzalez, and Peter Redgrave. The evolution of action selection. In David McFarland and O. Holland, editors, *Towards the Whole Iguana*. MIT Press, Cambridge, MA, to appear.
[51] Peter Redgrave, Tony J. Prescott, and Kevin Gurney. The basal ganglia: a vertebrate solution to the selection problem? *Neuroscience*, 89:1009–1023, 1999.
[52] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, and J. Lee. The Process Specification Language (PSL): Overview and version 1.0 specification. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD., 2000.
[53] Luc Steels. A case study in the behavior-oriented design of autonomous agents. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From Animals to Animats (SAB94)*, pages 445–452, Cambridge, MA, 1994. MIT Press. ISBN 0-262-53122-4.
[54] Luc Steels and Paul Vogt. Grounding adaptive language games in robotic agents. In C. Husbands and I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, London, 1997. MIT Press.
[55] Katia Sycara and Mattheus Klusch. Brokering and matchmaking for coordination of agent societies: A survey. In A et al Omicini, editor, *Coordination of Internet Agents*. Springer, 2001.
[56] Kristinn R. Thórisson. A mind model for multimodal communicative creatures & humanoids. *International Journal of Applied Artificial Intelligence*, 13(4/5): 519–538, 1999.
[57] David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A semantic web approach to service description for matchmaking services. In *Proc. International Semantic Web Working Symposium (SWWS)*, 2001.
[58] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, October 1950.
[59] UDDI. The UDDI Technical White Paper. http://www.uddi.org/, 2000.
[60] XML. XML Web Site. http://www.w3.org/XML/Schema, 2001.

A Basic Reactive Plans

This section describing the BRP is based on the description in [11]. We first developed this idiom independently, and called it a *competence*. However, it occurs in a number of other architectures [e.g. 16, 23, 48] and is so characteristic of reactive planning, that we refer to the generic idiom as a *Basic Reactive Plan* or BRP.

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 (typically 3–7) 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 priority π_i is drawn from a total order. Each action α_i may be either another BRP or a sequence as described above.

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 more than one step is released with the same priority, then

the winner is determined arbitrarily. Normally the releasers ρ_i on steps with the same priority are mutually exclusive. 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 such complex systems as mobile robots and flight simulators [3, 9, 16]. However, 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 a blue block.

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

In this case priority is strictly ordered and represented by position, with the highest priority step at the top. We refer to steps by priority. We now describe the operation of this plan.

In 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. If, as a part of some previous plan, the agent has already fixated on blue, **1** would be skipped because the higher priority step **2** has its releaser 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.

This single reactive plan can generate a large number of expressed sequential plans. In the initial context of a red block stacked on a blue block, we might expect the plan 1–2–3–1–2–4 to execute. 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. This BRP is identically robust and opportunistic to changes caused by another agent.

If an action fails repeatedly, then the above construction might lead to an indefinite behavior cycle. This can be prevented through several means. Our competences allow a retry limit to be set at the step level. Thus, in POSH a *competence step* is really a quadruple $\langle \pi, \rho, \alpha, \eta \rangle$, where η is an optional maximum number of retries. Other systems often have generic rules which are applied in the absence of progress or change.

The most significant feature of a BRP is that it is relatively easy to engineer. To build a BRP, the developer imagines a worst-case scenario for solving a particular goal, ignoring any redundant steps. The priorities for each step are then set in the inverse order that the steps might have to be executed. Next, preconditions are set, starting from the

highest priority step, to determine whether it can fire. For each step, the preconditions are simplified by the assurance that the agent is already in the context of the current BRP, and that no higher priority step can fire. For example, step **3** does not need the precondition (not (block blue)), and no step needs to say “If trying to find a blue block and nothing more important has happened then...”