FLOWS: A First-Order Ontology for Semantic Web Services [*]

Michael Grüninger[†] Richard Hull[‡] Sheila McIlraith[§]

June 20, 2008

We argue that an unambiguously, computer-interpretable description of the process model of a Web service, and a client's constraints on that process model, are critical to automating a diversity of tasks, including Web service discovery, invocation, composition, monitoring, verification and simulation. That the process model descriptions be unambiguously computer-interpretable is key to our argument, and is a shortcoming of a number of existing process modeling frameworks that have been proposed to describe aspects of Web services.

We commence this position paper with a brief overview of some existing process modeling frameworks used within the Web service community, providing analysis of some of their key merits and shortcomings. Next we present a small number of use cases that motivate and illustrate the need for computer-interpretable process models for the automation of Web service discovery and composition. With our use cases in hand, we propose a set of desiderata for a Web service description language that enables an unambiguous description of Web service process models. Our working hypothesis is that the process model be described as an ontology of first-order logic. To this end, we present the Process Specification Language (PSL) [**?**, **?**], a first-order logic ontology for modeling processes, as a straw proposal for a foundation from which a Web service process modeling framework can be developed.

The opinions expressed in this position paper reflect the position of the Semantic Web Services Language Committee (SWSL), a subcommittee of the joint North American-EU Semantic Web Services Intitiative (SWSI). They are included in *FLOWS*, the SWSL Committee's First-order Logic Ontology for Web Services [**?**].

1 Current State of Process Modeling for Web Services

A Web service *process model* describes the program that implements a Web service. Over the past 4 years, industry has proposed a number of process modeling languages for describing the process models of Web services. Examples include Microsoft's XLANG, a Web service process modeling language based on pi-calculus; IBM's WSFL based on Petri Nets; BPEL4WS, a Microsoft, IBM, BEA, SAP and Siebel effort, which merges XLANG and WSFL; HP's Web Service Conversation Language (WSCL); BEA, Intalio, SAP and Sun's Web Service Choreography Interface (WSCI); BPML, backed by the Business Process management initiative; the XML Process Description Langauge (XPDL) backed by the Workflow Management Coalition; the Business Process Specification Schema (BPSS) of ebXML; and now under development, a W3C Choreography effort, which draws on pi-calculus. While not exhaustive, this list identifies some of the major efforts in Web

In evaluating and comparing existing Web service process modeling efforts, a key observation is that these efforts were designed to address a diversity of process management tasks. Some like BPEL4WS were designed to address Web service orchestration issues and to standardize workflow and execution with the objective of increasing transaction reliability and synchronization. Others have focused on issues of Web service choreography, requiring defining the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal[1]. As a consequence of the diversity of uses for which these languaes have been designed,

[*]This position paper resubmits many of the points raised in a previous position paper by the authors [**?**].

[†]National Institute of Standards and Technology, Gaithersburg, MD, USA

[‡]Lucent Technologies, Murray Hill, NJ, USA

[§](Contact Author) University of Toronto, Toronto, ON, Canada, Email: sheila@cs.toronto.edu, Phone: 416-946-8484.

[1]http://www.w3.org/TR/ws-glossary/

comparing them based on concept coverage is important, but not necessarily pertinent, as many of these languages could be extended to incorporate further concept descriptions.

It is our view that the most important shortcoming of these languages, and the one that is least easily addressed, is their lack of well-defined semantics. Many of these languages have their origins in existing process specification languages, process algebras, and research in formal methods. Nevertheless, none has a well-defined semantics, except for XLANG and WSFL, which have effectively been abandonned in favour of BPEL4WS. Without a well-defined semantics, the process model cannot be manipulated, queried and interpretted reliably by a computer program. Adoption of a process model without a well-defined semantics severely restricts its practical use.

We take the point of view espoused by Semantic Web Services [**?**], that Web service descriptions should be unambiguously computer-interpretable, and that their concept coverage should be adequate to enable automation of Web service discovery, invocation, composition, monitoring, verification and simulation. We argue that this vision necessitates unambiguously computer-interpretable process models, which in turn necessitates having a well-defined semantics for these process models. We further argue that it requires certain content coverage, not addressed by current industry Web service process description languages.

In 2001, a coalition of semantic Web researchers, under the auspices of the DARPA DAML program, undertook to develop an ontology for Web services, using the Semantic Web ontology language DAML+OIL. This has culminated in the creation of OWL-S (formerly DAML-S) [**?**] a Web service ontology developed in OWL (the successor of DAML+OIL) [**?**]. OWL is an artificial intelligence description logic-based language for describing content on the Web. Most importantly OWL, and thus OWL-S, has a well-defined semantics, which contrasts it with other efforts. Unfortunately, OWL has not proven sufficiently expressive to characterize Web service process models. While OWL-S does indeed have a description of the process model of a Web service, OWL is not sufficiently expressive to denote all and only the *intended interpretations* of that process model. As such, like other process modeling languages, the OWL process model must be human interpretted to resolve ambiguities, or translated to another, richer language, in which this new model can be unambiguously interpretted by a program. Indeed there have been four efforts towards defining the intended interpretation of the OWL-S (or DAML-S) process model: a Petri Net-based operational semantics [**?**], an interleaving function-based operational semantics based on subtype polymorphism [**?**], a semantics via translation to the first-order language of the situation calculus [**?**], and most recently a semantics provided by translation to PSL, the National Institute of Standards' Process Specification Language.

OWL-S has many strong features. In particular, the concept coverage of OWL-S provides a firm foundation for our process modeling efforts. Further, OWL's expressiveness limitations, which OWL-S inherits, exist to address the important trade-off between expressiveness and decidability and tractability, and so, while limiting in this case, are certainly easily defensible.

Thus, we take OWL-S as our conceptual starting point and adopt much of its conceptual model which we do not describe here for lack of space. Nevertheless, we differ importantly in our choice of language. Rather than adopting OWL, we propose a first-order logic language for Web services descriptions. Before describing this language in further detail, we present a small set of motivating use cases.

2 Motivating Use Cases

In this section we present a small number of use cases that motivate and illustrate the need for computer-interpretable Web service process models in the context of Web service discovery and composition[2]. While many of the use cases are presented as client-side requests, these requests serve as constraints on existing process models and must be answered by manipulating server-side descriptions of the process models of Web services. As such, the challenge put forth in these use cases is to develop descriptions of the process models of relevant services, client-side requests, and (ideally) automated reasoning machinery to process the request.

Use Cases for Automating Web Service Discovery

1. Find me an airline service that enables me to reserve a flight before providing a credit card number.

[2]We chose these two tasks for diversity. Use cases requiring process modeling can likewise be developed for the other Semantic Web Service tasks.

2. Find me a book-buying service that returns a list of available second-hand copies of my requested book, if the book is out of print.

3. Find me a travel service that books hotels, flights and trains and that coordinates the timing. (I.e., that uses the output of one selection as input to the search for another.)

4. Find me a florist that enables me to pay with PayPal.

5. Find me an online financial advice service that queries www.morningstar.com prior to making mutual fund recommendations.

Use Cases for Automating Web Service Composition

1. Given a workshop registration service, a flight booking service, a car rental service, a taxi reservation service, and a hotel service, Ima's home address, and a Ima's online schedule, please book Ima Cheapskate's trip to the W3C Workshop on Web Services.

2. Modify problem 1, to add the constraints that Ima wants to travel on October 11, from Montreal, return on October 13 or 14, is unwilling to take an overnight flight, prefers to stay at the Sheraton in Palo Alto, but will stay at any hotel within 8 miles of the workshop and would like to rent a convertible car, if no rain is forecast. She does not want to register for the workshop until after her travel plans are made and does not want to rent a car, preferring a taxi, if her flight arrives in California between 3pm and 6pm.

3 Desiderata Derived from Use Case Analysis

Analysis of these and other use cases confirm the need for a computer-interpretable process model, and prompt a list of representational desiderata. We simply list these desiderata as follows: model-theoretic semantics, atomic and composite processes represented as first-class objects in the language, taxonomic representation, leverages OWL-S, embraces and integrates with exisitng and emerging industry Web service standards, provides for explicit representation of messages and dataflow, captures activities, process preconditions and program side effects, captures process execution history.

To address these use cases, we have developed a First-order Logic Ontology for Web Services (FLOWS).

4 FLOWS – A First Order Logic Ontology for Web Services

4.1 The Case for First-Order Logic

OWL is too weak to completely axiomatize the intended semantics of OWL-S, and consequently, any implementations must resort to extralogical mechanisms if they are to conform to the OWL-S semantics. FLOWS (First Order Logic Ontology for Web Services) provides a first-order axiomatization of the intended semantics of OWL-S, and implementations of FLOWS will be able to use the axioms directly.

First-order logic provides a well-understood model-theoretic semantics. Its rich expressive power (e.g., variables, quantifiers, terms, etc.) overcomes the expressiveness issues that have haunted OWL-S.

First-order logic enables characterization of reasoning tasks for semantic web services in terms of classical notions of deduction and consistency. For example, web service discovery can be characterized as deductive queries, and web service composition, reachability, and liveness as satisfiability. This enables exploitation of off-the-shelf systems such as existing FOL reasoning engines and DB query engines, thereby facilitating implementation and improving our understanding of the reasoning tasks.

First-order logic has been criticized because it is semi-decidable (as opposed to OWL-DL, which is decidable). However, the motivating scenarios for semantic web services show that in general we will need to solve intractable reasoning problems. Intractable reasoning problems are inherently intractable – using a different language does not make them tractable. The restriction to a language that is tractable simply means that there will exist reasoning problems that cannot be specified in the language.

Furthermore, many intractable tasks often prove easily solved in practice. One powerful strategy is to explicitly axiomatize the extensions of a first-order theory that have attractive computational properties. The idea is to focus on the complexity of a particular first-order theory, and to introduce domain assumptions that can be used to guarantee that a particular reasoning problem using the theory is tractable.

4.2 FLOWS Overview

The goal of FLOWS is to enable reasoning about the semantics underlying Web (and other eletronic) services, and how they interact with each other and with the "real world". FLOWS does not strive for a complete representation of web services, but rather for an abstract model that is faithful to the semantic aspects of service behavior. Following the lead of the situation calculii, and in particular the situation calculus semantics [**?**] of OWL-S, the changing portions of the real world are modeled abstractly using the notion of fluents. These are first-order logic predicates and terms that can change value over time. The FLOWS model provides infrastructure for representing messages between services; the focus here is on the semantic content of a message, rather than, for example, the specifics of how that content is packaged into an XML-based message payload. FLOWS also provides constructs for modeling the internal processing of Web services.

FLOWS is intended to enable reasoning about essential aspects of Web service behavior, for a variety of different purposes and contexts. Some targeted purposes are to support (a) descriptions of Web services that enable automated discovery, composition, and verification, and (b) creation of declarative descriptions of a Web service, that can be mapped (either automatically or through a systematic process) to executable specifications. A variety of contexts can be supported, including: (i) modelling a service as essentially a black box, where only the messaging is observable; (ii) modelling the internal atomic processes that a service performs, along with the impact these processes have on the real world (e.g., inventories, financial accounts, commitments); (iii) modelling many properties of a service, including all message APIs, all or some of the internal processing, and some or all of the internal process and data flows. Of course, the usability of a Web service description may depend on how much or how little information is included.

It is important to note that the specification of the FLOWS ontology in first-order logic does not presuppose that the automated reasoning tasks described above will be realized using a first-order logic theorem prover. We can certainly use FOL to specify solutions to these tasks, using notions of entailment and satisfiability. Nevertheless, while some tasks may naturally be realized through theorem proving, it has been our experience that most AI automated reasoning tasks are addressed by special-purpose reasoners, rather than by general-purpose reasoners such as theorem provers.

A key premise of FLOWS is that an appropriate foundation for formally describing Web services can be built as a family of PSL extensions. PSL – the Process Specification Language – is a formally axiomatized ontology [**?**] that has been standardized as ISO 18629. PSL was originally developed to enable sharing of descriptions of manufacturing processes. FLOWS refines aspects of PSL with Web service-specific concepts and extensions.

A complete description of our FLOWS ontology can be found at [**?**].

5 Closing Remarks

In closing, we re-articulate our position that an unambiguous computer-interpretable process model of Web service is essential to automation of many Web service tasks. Our position is embodied in FLOWS, developed first-order ontology for Web services.