A Short Overview of FLOWS: A First-Order Logic Ontology for Web Services*

Michael Grüninger
University of Toronto
Toronto, ON, Canada

Richard Hull
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA

Sheila A. McIlraith
University of Toronto
Toronto, ON, Canada

Abstract

FLOWS is a first-order logic ontology for Web services and a W3C Submission. In this article, we describe some of the motivation behind the development of FLOWS, together with its key features.

1 Introduction

The *First-order Logic Ontology for Web Services* (FLOWS) [9], also known as the Semantic Web Services Ontology (SWSO), was initially developed during 2002 to 2004 by a team of academic and industrial researchers, as part of the larger Semantic Web Services Framework (SWSF) effort, which culminated in a W3C Member Submission [10] in 2005. FLOWS was created on the premise that an unambiguous, computer-interpretable description of the process model of Web services and how they are composed, and client constraints on the services to be provided, are critical to automating a diversity of tasks, including Web service discovery, invocation, composition, monitoring, verification and simulation. To this end, FLOWS is based on first-order logic, and provides a rigorous axiomatization that captures the salient process-level semantics of Web services.

FLOWS is an extension of the Process Specification Language (PSL) [4], a first-order logic ontology for modeling processes that was originally developed for manufacturing processes, and realized as an ISO standard in 2004. FLOWS enables partial and/or complete specifications of the properties of Web services, including pre- and post-conditions, internal structures, composition patterns, messaging behaviors, and impact on the external world, all in the context of a rigorously axiomatized first-order logic framework. This article provides a very brief overview of FLOWS, including comparisons with related work, key principles underlying the creation of FLOWS, and illustrations of key components of the framework.

Background. A Web service *process model* describes the program that implements a Web service, which might itself be formed as a composition of other Web services. Over the years, a number of languages have been proposed for describing the process models of Web services. Some of the most important 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

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported in part by NSF grants IIS-0415195 and CNS-0613998, and by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Research and Innovation.

Interface (WSCI); BPML, backed by the Business Process management initiative; the XML Process Description Language (XPDL) backed by the Workflow Management Coalition; the Business Process Specification Schema (BPSS) of ebXML; and the W3C Choreography effort, WS-CDL, which draws on pi-calculus. The most popular language for Web Service orchestration in use today is WS-BPEL 2.0 (or BPEL for short), which offers significant enhancements over its predecessor, BPEL4WS.

In evaluating and comparing these efforts, a key observation is that they were designed to address a diversity of targeted process management tasks. Some, like BPEL, were designed to address Web service orchestration issues and to standardize workflow and execution with the objective of increasing transaction reliability and synchronization. Others, like WS-CDL, have focused on issues of Web service choreography, which involves message exchange to coordinate the activities of independent agents. As a consequence of the diversity of uses for which these languages have been designed, 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. For example, several attempts have been made to formalize predecessors to WS-BPEL 2.0 using Petri nets, process algebras, and abstract state machines. While the WS-BPEL 2.0 specification is more precise with respect to the semantics, it is still informally defined [6].

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 culminated in the creation of OWL-S (formerly DAML-S) [2] a Web service ontology developed in OWL (the successor of DAML+OIL) [5], and a W3C Member Submission in 2004. OWL is a family of knowledge representation languages for authoring ontologies and is endorsed by the World Wide Web Consortium (W3C). The semantics of most of the OWL languages, specifically OWL DL and OWL Lite, is based on artificial intelligence description logic, a subset of first-order logic. OWL Full (soon to be replaced by OWL 1.1) is based on a novel semantics that provides compatibility with RDF Schema. Most importantly OWL, and thus OWL-S, has a well-defined (formal) semantics, in constrast to efforts listed above.

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 [7], an interleaving function-based operational semantics based on subtype polymorphism [1], a semantics via translation to the first-order language of the situation calculus [7], and a semantics provided by translation to PSL [3].

OWL-S has many strong features. In particular, the concept coverage of OWL-S provides a firm foundation for process modeling efforts, including the ability to create both partial and complete specifications of relevant aspects of a group of Web services. Further, OWL's expressiveness limitations, which OWL-S inherits, exist to address the important trade-off between expressiveness on the one hand and decidability and tractability on the other, and are thus easily defensible in this context. Nevertheless, it was experience with OWL-S that, in part, motivated the development of FLOWS.

The Web Service Modeling Ontology (WSMO) [11] also provides an expressive, layered ontology for web services and their compositions, using a semantics based on a combination of description logic and horn logic.

Principles underlying FLOWS. FLOWS was developed based on the following three principles.

Provide a fully expressive language and framework. The goal of FLOWS is to enable reasoning about the semantics underlying Web 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. In that context, FLOWS enables a variety of reasoning tasks, by supporting descriptions of Web services that enable automated discovery, composition, and verification. This also includes the creation of declarative descriptions of a Web service, that can be mapped (automatically or through a systematic, partially manual process) to executable specifications. In particular, then, unlike the industrial process modeling languages listed above, FLOWS is intended to support, within one language and underlying framework, reasoning about Web services from a very broad range of perspectives.

FLOWS is a modular and extensible ontology. It is thus possible to provide alternative extensions to represent different approaches to message handling, choreography, and orchestration. As such, FLOWS can serve as an interlingua ontology that can facilitate interoperability of Web services that use different ontologies.

Use first-order logic as the basis. First-order logic enables the characterization of reasoning tasks for semantic Web services in terms of classical notions of deduction and consistency. FLOWS can be used to specify tasks in support of Web service discovery and composition; checking service properties such as reachability, liveness, and compliance with behavioral patterns and constraints; and querying about a wide range of semantic and temporal properties of services. This enables exploitation of off-the-shelf systems such as existing FOL reasoning engines and database query engines, thereby facilitating implementation and improving our understanding of the reasoning tasks. At the same time, the use of first-order logic does not preclude the use of alternative reasoning methods on selected subsets of FLOWS.

First-order logic has been criticized because it is semi-decidable (as opposed to OWL DL, which is decidable). Nevertheless, 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.

Capture full semantics using an extensible family of axioms. Although other approaches to semantic Web services specify concepts contained in FLOWS, they do not provide a rigorous and complete axiomatization to support automated reasoning about the concepts. Incomplete axiomatizations require the use of additional extralogical mechanisms rather than reasoning from the axioms alone. Since automated Web services can share axioms, but not the specification of special-purpose axioms, incomplete axiomatizations restrict the reusability and sharability of an ontology.

In FLOWS, the process model for Web services and their compositions is formally specified using PSL This provides predicates and axioms that enable representation of, and reasoning about, core process modeling concepts, including fluents (that is, first-order predicates representing some portion of the "real world" that can change over time), activities (such as Web services), activity-occurrences (such as individual executions of Web services), and the values of fluents before and after activity-occurrences. The PSL standard is comprised of a layered collection of families of axioms that can be used to reason about a broad class of processes. As discussed in more detail below, FLOWS provides additional families of axioms, layered on top of a subset of the PSL axiom families, to enable representation of, and reasoning about, Web services and their compositons.

2 Key aspects of FLOWS

After providing some motivating use cases, this section briefly highlights some key aspects of the FLOWS language and framework, and illustrates how it can support the use cases.

Motivating use cases. The space of use cases that motivates and illustrates the need for computer-interpretable Web service process models is vast, ranging from discovery and composition to analysis, monitoring and error-recovery, and including activities such as queries about individual services and over families of services, contract enforcement, service histories, and provenance. We illustrate now with just a few examples, centered around services that focus on selling and shipping books.

1. *Inquiries about one service:* Does the Acme bookseller service always return a list of available second-hand copies of my requested book, if the book is out of print? Under what conditions does the Acme bookseller service permit me to pay for books using Paypal?

2. *Discovering services:* Find all bookselling services that will, at least in some cases, return a list of available second-hand copies of my requested book if that book is out of print. Or, find all bookselling services that will always return a list of available second-hand copies of my requested book, if it is out of print.

3. *Discovering composite services:* Find all book seller-shipper partnerships that are able to split shipments (e.g., as a consequence of delayed availability of some books) without an additional charge.

4. *Requesting (composite) services:* Create a service that can book a flight to Toronto, find and reserve a hotel room there for next Monday to Wednesday, identify the best way to get from the airport to the hotel, and ship a guide-book about Toronto to me at that hotel in time for my arrival. Furthermore, the hotel must be within 15 minutes travel (by foot and/or public transportation) of the Computer Science department.

5. *Responding to exceptions:* If the preceding scenario is underway and the flight into Toronto is delayed, then dynamically provide new recommendations on the best way to get from airport to hotel.

Ontology. As noted above, in the FLOWS ontology Web services (both atomic and composite) are represented as PSL *activities*, and Web service executions are represented as PSL *activity-occurrences*. Predicates that change in the real world due to activity-occurrences are modeled using *fluents*. An activity-occurrence is a limited, temporally extended piece of the world, with a clear temporal start-point and end-point.

As in PSL, the set of possible executions of a composite Web service is modeled essentially as a tree, whose nodes correspond to individual activity-occurrences, i.e., service executions, and where the children of one activity-occurrence correspond to the set of all possible activity-occurrences that could immediately follow it. For a fluent such as *book_available*(t, w) for title t and warehouse w, and service-occurrence o, the value of the fluent immediately before o occurs is given by the predicate *pre*(*book_available*(t, w),o), and the value of the fluent just after o occurs is given by the predicate *holds*(*book_available*(t, w),o). It is from these basic constructs that the full family of possible executions of a composite Web service can be represented and reasoned about. These constructs can also be used to specify pre-conditions and effects of services.

FLOWS also provides constructs for modeling the internal processing of composite Web services, including sequences, nondeterminism (i.e. alternative activities), iterated activities, conditional activities, and concurrency. As with Golog [8], these constructs are formally modeled as constraints, which enables both partial and complete specifications of processing characteristics. Activities can be decomposed into primitive activities or composed into more complex activities, and different classes of activities are defined with respect to ordering and temporal constraints on the subactivities. In this way, FLOWS supports reasoning with both complete and incomplete process specifications. In addition, FLOWS refines aspects of PSL with Web service-specific concepts and extensions, such as providing the infrastructure for representing messages between services.

Axiomatization. The FLOWS axiomatization is layered on top of the axiomatization of PSL Outer Core. The axioms provide a rigorous and complete specification of the FLOWS constructs, including such concepts as service, service-occurrence, messages and channels, control constructs for service composition, various kinds of constraints, and exceptions.

Enabling the use cases through queries and reasoning. The approach to support reasoning tasks with the FLOWS ontology and axioms is now illustrated with the use cases. The focus here is to illustrate the expressive power of FLOWS. It is clear that many of the problems that can be specified in FLOWS have high complexity or are undecidable; as noted above it is possible to create restricted versions of these reasoning tasks in order to obtain decidability and lower complexity.

Inquiries about an individual service, such as Acme book seller, can be achieved by reasoning over the tree of possible executions of the service. Note that both universal and existential quantification will be called for. For service discovery, the properties characterizing the desired services can be specified using formulas over the FLOWS ontology. This essentially reduces discovery to querying a database of service specifications. Even the composition of services is achieved through the specification of a formula with one free variable which describes the desired properties of the composition; each solution for this variable will be a (composite) service that provides a composition with the desired capabilities. Finally, FLOWS is able to represent the state of the world when a composite service has partially completed its execution. As such, it enables exception handling.

3 Closing Remarks

In this article we described some of the motivation behind the development of FLOWS, together with key aspects of the FLOWS ontology. In doing so, we argued that existing languages for modeling Web services were either lacking in expressivity, or did not have a well-defined semantics. As such, their ability to model and enable automated reasoning about Web services was limited. In contrast, FLOWS' use of first-order logic provides sufficient expressivity, a well-defined semantics, and a diversity of automated reasoning tools. FLOWS presents a natural evolution in the modeling of semantic Web services, reflecting a trend in semantic Web techonologies towards the use of more expressive ontology languages. Recent extensions to OWL, both realized and proposed, bear witness to this trend. Readers interested in further details on FLOWS are encouraged to consult the FLOWS (a.k.a. SWSO) specification at [9], which includes the full ontology and selected use cases.

References

[1] A. Ankolekar, F. Huch, and Katia Sycara. Concurrent execution semantics for DAML-S with subtypes. In *Proceedings First International Semantic Web Conference (ISWC2002)*, 2002.

[2] DAML-S Coalition: A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. DAML-S: Semantic markup for Web services. In *Proc. International Semantic Web Working Symposium (SWWS)*, pages 411–430, 2001. `http://www.daml.org/services/`.

[3] M. Grüninger. Applications of PSL to semantic web services. In *Proc. of Workshop on Semantic Web and Databases, in association with the Very Large Databases Conference (VLDB)*, 2003.

[4] M. Grüninger and C. Menzel. Process specification language: Theory and applications. *AI Magazine*, 24:63–74, 2003.

[5] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[6] N. Lohmann, H. M. W. Verbeek, C. Ouyang, C. Stahl, and W. M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Tech. Univ. Eindhoven, The Netherlands, August 2007.

[7] S. Narayanan and S. McIlraith. Analysis and simulation of web services. *Computer Networks*, 42:675 – 693, 2003.

[8] R. Reiter. *KNOWLEDGE IN ACTION: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[9] SWSF Committee. Semantic Web Service Ontology (SWSO): First-order Logic Ontology for Web Services (FLOWS), September 9, 2005. `http://www.w3.org/Submission/SWSF-SWSO/`.

[10] SWSF Committee. Semantic Web Services Framework (SWSF) overview, September 9, 2005. `http://www.w3.org/Submission/SWSF/`.

[11] WSMO Committee The Web Service Modeling Ontology (WSMO) Submission, June, 2005. `http://www.w3.org/Submission/2005/06/`.