A Critical Evaluation of Web Service Modeling Ontology and Web Service Modeling Language

Omid Sharifi[1] and Zeki Bayram[2(✉)]

[1] Computer and Software Engineering Department, Toros University, Mersin, Turkey
omid.sharifi@toros.edu.tr
[2] Computer Engineering Department, Eastern Mediterranean University, Famagusta, Cyprus
zeki.bayram@emu.edu.tr

Abstract. Web Service Modeling Language (WSML), based on the Web Service Modeling Ontology (WSMO), is a large and highly complex language designed for the specification of semantic web services. It has different variants based on logical formalisms, such as Description Logics, First-Order Logic and Logic Programming. We perform an in-depth study of both WSMO and WSML, critically evaluating them by identifying their strong points and areas in which improvement would be beneficial. Our studies show that in spite of all the features WSMO and WSML support, their sheer size and complexity are major weaknesses, and there are other areas in which important deficiencies exist as well. We point out those discovered deficiencies, and propose remedies for them, laying the foundation for a more tractable and useful formalism for specifying semantic web services.

Keywords: Semantic web services · WSMO · WSML · Evaluation

1 Introduction

The goal of web services is to allow normally incompatible applications to interoperate over the Web regardless of language, platform, or operating system [10]. Web services are much like remote procedure calls, but they are invoked using Internet and WWW standards and protocols such as Simple Object Access Protocol (SOAP) [2] and Hypertext Transfer Protocol (HTTP) [1].

Web Services Modeling Ontology (WSMO) [3] is a comprehensive framework for describing web services, goals (high-level queries for finding web services), mediators (mappings for resolving heterogeneities) and ontologies. Web Services Modeling Language (WSML) [5] is a *family* of concrete languages based on F-logic [11] that implement the WSMO framework. The variants of WSML are WSML-core, WSML-flight, WSML-rule, WSML-DL, and WSML-full. WSML is large, relatively complex, and somewhat confusing, with different variants being based on different formalisms. The complexity and confusion arise mainly from the many variants of the language, and the rules used to define the variants.

T. Czachórski et al. (Eds.): ISCIS 2016, CCIS 659, pp. 97–105, 2016.
DOI: 10.1007/978-3-319-47217-1_11

The variants of WSML form a hierarchy, with WSML-full being on top (the most powerful) and WSML-core being at the bottom (weakest).

Our literature search has failed to reveal any significant industrial real-life application that uses WSML. We believe this is due to the inherent complexity of the language, the "less-than-complete" state of WSML (e.g. the syntax of WSML-DL does not conform to the usual description logic syntax, choreography specification using abstract state machines (ASM) [8] seems unfit for the job due to the execution semantics of ASMs, goals, choreographies and web services are not integrated in the same logical framework etc.), as well as the lack of proper development tools and execution environments. So WSML looks like it is still in a "work-in-progress" state, rather than a finished product.

In this work, we critically evaluate the strengths and weaknesses of WSMO and WSML, and determine the areas of improvement that will result in a usable semantic web service specification language. This is the main contribution of this work, which will be input to the next phase of our research, the actual design and implementation of such a language.

The remainder of the paper is organized as follows. Section 2 contains a critical evaluation of WSMO and WSML, including their strengths, weaknesses and deficiencies, discovered both through our detailed study of the documentation provided for WSMO and WSML, as well as experimentation with the paradigm in several use-cases. In Sect. 3 we have a brief discussion of related work, and finally Sect. 4 is the conclusion and future research directions.

2 Evaluation of WSMO and WSML

In this section we discuss the strong and weak points of WSMO and WSML as discovered through our studies of their specification and the practical experience gained through experimentation. We also suggest possible improvements wherever possible.

2.1 General Observations

WSMO boasts a comprehensive approach that tries to leave no aspect of semantic web services out. These include ontologies, goals, web services and mediators. In the same spirit of thoroughness, designers of WSML have adopted the paradigm of trying to provide everything everybody could ever want and let each potential user chose the "most suitable" variant of the language for the job at hand. This approach has resulted in a complex syntax, as well as a complex set of rules that differentiate one version of the language form another.

2.2 Deficiencies in Syntax

WSML-DL and WSML-full have no explicit syntax for the description logic component [5], relying on a first-order encoding of description logic statements. Without proper syntax, it is not possible to use them in the specification of semantic web services in a convenient way.

2.3 Logical Basis of WSMO

The ontology component of WSMO is based on F-logic, which gives this component a solid theoretical foundation. However, its precise relationship to F-logic has not been given formally, and what features of F-logic have been left out are not specified explicitly.

2.4 Lack of a Semantics Specification for Web Service Methods/Operations

In spite of all the effort at comprehensiveness, there are significant omissions in WSMO, such as specification of the semantics of actual methods (operations) that the web service provides, which makes it impossible to *prove* that after a "match" occurs between a goal and a web service, the post-condition of the goal will indeed be satisfied. Even worse, once matching succeeds and the web service is called according to the specified choreography, the *actual results* of the invocation may not satisfy the post-condition of the goal. Below, we explain why.

In WSMO, matching between a goal and web service occurs by considering the pre-post conditions of the goal and web service, and this is fine. The problem occurs because of the *lack* of a semantic specification (for example, in the form of pre-post conditions) for web service *methods/operations*, and how these methods are actually called through the execution of the choreography engine. Method calls are generated according to availability of "data" in the form of instances, and the mapping of instances to parameters of methods. There is no consideration of logical conditions which must be true before the method is called, and no guarantee of the state of the system after the method is called, since these are not specified for the web methods. Instances of a concept can be parameters to more than one web method. Assuming two methods A and B have the same signature, it may be the case that an unintended method call can be made to B, when in fact the call should have been made to A, which results in wrong computation. Consequently, not only is it impossible to *prove* that after a "match" occurs between a goal and a web service, the post-condition of the goal will be satisfied, but also once the web service execution is initiated, the computation itself can produce wrong results, invalidating the logical specification of the web service.

Unfortunately, the interplay between choreography, grounding and logical specification of what the web service does (including the lack of the specification of semantics for web service methods) has been overlooked in WSMO. All these components need development and integration in order to make them part of a *coherent* whole.

2.5 Implementation and Tool Support

Some developmental tools, such as the "Web Services Modeling Toolkit" [4] exist which make writing WSML specifications relatively easy. However, these

tools depend on external reasoner support, rather than having intrinsic reasoning capabilities. As such, development and testing of semantic web service specifications cannot be made in a reliable manner. For example, no explanations are given when discovery fails for a given goal.

2.6 Choreography in WSMO

We have already talked about how the interplay of choreography and grounding can result in incorrect execution, invalidating the logical specification of a web service. In this section, we delve more deeply into the problems of WSMO choreography.

- WSMO choreography is purportedly based on the formalism of abstract state machines [8], but in fact it is only a crude approximation. Very significantly, evolving algebras are magically replaced with the state of the ontologies as defined by instances of relations and concepts. This transformation seems to have no logical basis, so the applicability of any theory developed for abstract state machines to WSMO choreography specifications is questionable. The choreography attempt of WSMO looks more like a forward-chained expert system shell, where the role of the "working memory" is played by the current set of instances in the ontologies. It probably would be more reasonable to consider WSMO choreography in this way, rather than being based on abstract state machines.
- The fact that in an abstract state machine rules are fired in parallel does not match well with the real life situation that method calls implied by the firing of rules have to be executed sequentially.
- Both goals and web services have choreography specifications, but there is no notion of how the choreographies of goals and web services are supposed to match during the discovery phase. It is also not clear how the two are supposed to interact during the execution phase. Although restrictions on who can modify the state of the ontology and in what way can be specified in the form of modes of concepts, this is relatively complex, and far from practical. In the documentation of WSMO, only the choreography of the service is made use of.
- Choreography grounding in WSMO tries to map instances to method parameters of the web service methods by relating concepts to the methods directly. Methods are then called when their parameters are available in the current working memory. The firing of the rules are intermixed with the invocation of methods (with appropriate lowering/lifting of parameters), and changes to working memory by actions on the right hand side are forbidden (presuming that any changes will be made by the actual method call). This is a strange state of affairs, since the client may itself need to add something to the working memory, and there is no provision for this.
- The choreography rule language allows nested rules. Although this nesting permits very expressive rules to be written, using the "if", "forall" and "choose" constructs in any combination in a nested manner, the resulting rules are prohibitively complex, both to understand, and to execute.

- As mentioned before, in the grounding process, only the availability of instances that can be passed as parameters to methods, and the predetermined mapping between concepts and parameters, are considered, with no pre-conditions for method calls. This is a major flaw, since it may be that two methods have exactly the same parameter set, but they perform very different functions, and the wrong one gets called.
- The choreography specification is disparate from the capability specification (pre-conditions, post-conditions), whereas they are in fact intimately related and intertwined. The actions specified in the choreography should actually take the initial state of the ontologies to their final state, through the interaction of the requester and web service. This fact is completely overlooked in WSMO choreography.
- Choreography engine execution stops in WSMO when no more rules apply. A natural time for it to stop would be when the conditions specified in the goal are satisfied by the current state of the ontology stores. Again this is a design flaw, which is due to the fact that the intimate relationship between the capability specification and choreography has been overlooked.

2.7 Orchestration in WSMO

The orchestration component of WSMO is yet to be defined. The creators of WSMO say it will be similar to choreography, and be part of the interface specification of a web service. At a conceptual level, however, we find the specification of orchestration for a web service somewhat unnecessary. Why would a requester care about *how* a service provider provides its service? Composition of web services to achieve a goal *would* be much more meaningful, however. So the idea of placing orchestration within a web service specification seems misguided. Its proper place would be inside the specification of a *complex* goal, which would help and guide the service discovery component to not only find a service that meets the requirements of the goal, but also mix-and-match and compose different web services to achieve the requirements of the goal.

2.8 Goal Specification

The goal specification includes the components "assumptions," "pre-conditions," "post-conditions" and "effects," just like the web service specification. The logical correspondence between the "pre-conditions," "assumptions," "post-conditions" and "effects," of goals and web services is not specified at all. The usage of the same terminology for both goals and web services is also misleading. In reality, the web service *requires* that its pre-conditions and assumptions hold before it can be called, and *guarantees* that if it is called, the post-conditions and effects will be true. On the other hand, the goal declares that it *guarantees* a certain state, perhaps by adding instances to the instance store, of the world before it makes a request to a web service, and *requires* certain conditions to be true as a result of the execution of the web service. The syntax of the goals should be consistent with this state of affairs.

2.9 Reusing Goals Through Specialization

Being able to reuse an existing goal after specializing it in some way would be very beneficial. The template mechanism of programming languages, or "prepared queries with parameters" in the world of databases are concepts which can be adapted to goals in WSMO to achieve the required specialization. Such functionality is currently missing in WSML.

2.10 Specialization Mechanism for Web Service Specifications

Developing a web service specification from scratch is a very formidable task. Just like in the case of specializing goals, a mechanism for taking a "generic" web service specification in a domain, and specializing it to describe a specific web service functionality would be a very useful proposition. To take this idea even further, a hierarchy of web service specifications can be published in a central repository, and actual web services can just declare that they implement a pre-published specification in the hierarchy. Or, they can grow the hierarchy by specializing an existing specification, and "plugging" their specification into the existing hierarchy. Such an approach will help in service discovery as well. A specialization mechanism for web services does not exist in WSMO, and would be a welcome addition to it.

2.11 Missing Aggregate Function Capability

The logic used in WSML (even in WSML full) does not permit aggregate functions in the sense of database query languages (sum, average etc.). Such an addition however would require moving away from first order logic into higher order logic, with corresponding loss of computational tractability. Still, it may be worthwhile to investigate restricted classes of aggregate functionality which lend themselves to practical implementation. For example, a built-in *setof* predicate could be used to implement aggregate functions.

2.12 Extra-Logical Predicates

The ability to check whether a logic variable is bound to an object, or whether it is in an unbound state (the *var* predicate of Prolog [16]) is missing. The availability of this feature is of practical importance, since for example a web service pre-condition may be a disjunction, and depending on the input provided by the goal, some variables in the disjunction may remain unbound after a successful match.

2.13 Multiple Functionality in a Web Service

A WSML goal or web service may only have one capability [9]. This is a severe restriction, since a web service can possibly provide different results, depending on the provided input. Ideally, each web service specification should be able to have a *set* of capabilities. This is not currently available in WSMO or WSML.

2.14 Automatic Mapping Between Attributes and Relations

Although one can define a binary relation for each attribute using an axiom, relating objects and their attribute values, this is cumbersome when done manually. Having it done automatically would be nice, a feature currently not available in WSML.

2.15 Error Processing

There is currently no mechanism specifying how to handle errors when they arise. For example, what should be done when a constraint is violated in some ontology? There should be a way of communicating error conditions to the requester when they arise. This could be the counterpart of the exception mechanism in programming languages.

2.16 No Agreed-Upon Semantics for WSML-Full

WSML-full, which is a combination of WSML-DL and WSML-rule, has no agreed-upon semantics yet [9] yet. With no formal semantics available, it is hard to imagine how WSML-full specifications could be processed at all.

3 Related Work

The authors have benefited from practical experience gained through semantic web service specification use cases reported in [6,13,14] in determining weak points of WSMO and WSML, in addition to unreported extensive experimentation. Although some of the drawbacks of WSML reported here have been pointed out in the master thesis by Cobanoglu [7] as well, our coverage of the choreography issue is unique in its depth and scope. We also offer solutions wherever possible to improve WSMO and WSML.

Our literature search failed to reveal any additional comprehensive study on the weaknesses of WSMO and WSML. However, we should also mention WSMO-lite [12,15], a relatively recent bottom-up semantic web service specification framework inspired by WSMO, that recognizes and provides solutions for the problems of specifying pre and post conditions for web service operations, as well as dealing with error conditions.

4 Conclusion and Future Work

We investigated the WSMO semantic web service framework, and the WSML language through an in-depth study of both, as well as extensive practical experimentation. Our investigation has revealed several deficiencies and flaws with WSMO and WSML, which we presented in this paper. We also provided suggestions for improvement where possible.

In future work, we are planning to develop a logic based semantic web service framework that builds on the strengths of WSMO, but at the same time remedies the weaknesses identified in this paper. Our proposal will aim to be coherent, where all the components are in harmony with each other, manageable, not unnecessarily complex, and practical enough to be used in real life.

References

1. HTTP - hypertext transfer protocol. http://www.w3.org/Protocols/. Accessed 19 Apr 2016
2. SOAP version 1.2 part 1: Messaging framework (2nd edn.). https://www.w3.org/TR/soap12/. Accessed 19 Apr 2016
3. Web Service Modeling Ontology. http://www.wsmo.org/. Accessed 30 Mar 2016
4. Web Services Modelling Toolkit. https://sourceforge.net/projects/wsmt/. Accessed 18 Apr 2016
5. WSML - Web Service Modeling Language. http://www.wsmo.org/wsml. Accessed 30 Mar 2016
6. Çobanoğlu, Ş., Bayram, Z.: Semantic web services for university course registration. In: Kim, W., Ding, Y., Kim, H.-G. (eds.) JIST 2013. LNCS, vol. 8388, pp. 3–16. Springer, Heidelberg (2014)
7. Cobanoglu, S.: A critical evaluation of web service modeling language. Master Thesis, Eastern Mediterranean University, February 2013
8. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Heidelberg (1984)
9. Group, W.W., et al.: D16.1v1.0 WSML language reference final draft 2008–08-08 (2008). http://www.wsmo.org/TR/d16/d16.1/v1.0/. Accessed 20 Apr 2016
10. McGovern, J., Tyagi, S., Stevens, M., Mathew, S.: Java Web Services Architecture. Morgan Kaufmann, San Francisco (2003)
11. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**(4), 741–843 (1995). doi:10.1145/210332.210335
12. Roman, D., Kopeck, J., Vitvar, T., Domingue, J., Fensel, D.: WSMO-Lite and hRESTS: lightweight semantic annotations for web services and RESTful APIs. Web Semant. Sci., Serv. Agents WWW **31**, 39–58 (2015)
13. Sharifi, O., Bayram, Z.: Database modelling using WSML in the specification of a banking application. In: Proceedings WASET 2013, pp. 263–267 (2013)

14. Sharifi, O., Bayram, Z.: Specifying banking transactions using web services modeling language (WSML). In: Proceedings of the Fourth International Conference on Information and Communication Systems (ICICS 2013), pp. 138–143 (2013)
15. Vitvar, T., Kopecký, J., Viskova, J., Fensel, D.: WSMO-lite annotations for web services. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 674–689. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68234-9_49
16. Clocksin, W.F., Mellish, C.S.: Programming in Prolog. Springer, Verlag (1984)