A Novel Concise Specification and Efficient F-Logic Based Matching of Semantic Web Services in Flora-2

Shahin Mehdipour Ataee and Zeki Bayram

Abstract We propose a novel concise specification of semantic web services conforming to the WSMO standard using the Flora-2 language, as well as a precise logical definition of what it means for a goal to match a web service. Our innovative usage of Flora-2 allows very short but expressive descriptions of both goals and web service capabilities, which are then used by a matching engine to discover which web services can satisfy a given goal. The matching engine, using the meta-level F-logic inferencing capabilities of the underlying Flora-2 reasoner, is very efficient and has a very concise definition itself.

1 Introduction

In service-oriented architecture (SOA), web services are defined, registered, invoked, and interconnected via some pre-agreed specifications [1]. Web service discovery is the process of finding one or more appropriate web service(s) among a possibly large pool of diverse web services. The non-intelligent way is to manually refer to web service repositories and use traditional text retrieval techniques to find some candidates for the specified application. On the other hand it can be done (semi) automatically by applying certain AI techniques such as logical inference. The latter approach requires the availability of rich semantic description of web service capabilities, user requirements and other related aspects of web services (such as non-functional properties), commonly in the form of logical statements in some appropriate form of logic, and in this case discovery amounts to proving certain logical inferences.

The Web Service Modeling Ontology (WSMO) [2] is a meta-ontology for describing relevant aspects of semantic web services that facilitates the logical description of web services, user requests in the form of goals, common vocabulary in the form of ontologies, and bridging the heterogeneities among web services and goals through mediation. Several languages of varying expressive power, such as WSML-Rule,

S.M. Ataee (✉) · Z. Bayram
Eastern Mediterranean University, TRNC via Mersin 10, Famagusta, Turkey
e-mail: shahin.mpa@gmail.com

O.H. Abdelrahman et al. (eds.), *Information Sciences and Systems 2015*,
Lecture Notes in Electrical Engineering 363,
DOI 10.1007/978-3-319-22635-4_17

WSML-Flight, and WSML-Full [3] have been proposed to specify web services according to WSMO, all based on Frame-logic (F-logic) [4].

Flora-2 by Michael Kifer et al. [5] is a powerful language for knowledge representation and reasoning. It is based on F-logic, HiLog [6], and Transactional logic [7]. Frames are fundamental structures in Flora-2. They provide a means for defining classes and objects logically. In the case of objects, a frame can be shown generally in form of `objectId[attribute`$_1$`-> value`$_1$`, attribute`$_2$`-> value`$_2$`, ..., attribute`$_n$`-> value`$_n$`]`.

In this work, we use Flora-2 as a specification language for semantic description of web service components according to WSMO and implement a matching engine based on inference in F-logic in order to discover web services that can satisfy user requests specified in the form of goals. Our semantic specification is very concise since it makes use of the underlying Flora-2 syntax to the highest degree possible. The implementation of the matcher is also very compact since it makes effective use of the meta-level capabilities of the Flora-2 system.

The rest of this paper is structured as follows. In Sect. 2, we describe the semantic specification of goals and web services in WSMO, as well as the logic we have used in the implementation of our service matching engine. In Sect. 3, we demonstrate the power and practicality of our scheme through a simple but realistic scenario (first described in [8]) and show how our solution fulfills all the requirements needed to specify and discover web services/goals. We present a short survey of related work and compare it to ours in Sect. 4. Finally, in Sect. 5 we present the conclusion and future work in the area of semantic web service discovery based on our semantic web service specification approach.

2 Semantic Specification of Goals and Web Services and the Matching Process

The functionality of a WSMO web service is defined under the *capability* tag (element) which contains four axioms: pre-condition, assumption, post-condition and effect [9]. Pre and post conditions represent the internal state of the web service, whereas assumption and effect represent the state of the outside world (environment). A WSMO web service guarantees its post-condition and effect if its pre-condition and assumption are true. This feature of WSMO web services is used for discovery and selection purposes. For the sake of simplicity, we only consider web service pre-conditions (shown by *web.pre*) and web service post-conditions (shown by *web.post*), since assumptions and effects can be handled in a similar way.

Logically, the functionality of a web service can be shown by the following formula (the arrow represents the *implication* operator).

$$\forall x : web.pre(x) \Rightarrow web.post(x) \tag{1}$$

This formula means that for all instantiations of the free variables in the formula (represented by x), if the pre-condition is true, the web service guaranties that the post-condition will also be true after the web service has finished its execution.

The definition of a logical match between a goal and a web service can be described precisely with the formula below:

$$\begin{aligned} &\forall x_i \forall y_i : ((goal.pre(x_i) \Rightarrow web.pre(y_i)) \\ &\quad \wedge \\ &(goal.pre(x_i) \wedge (web.pre(y_i) \Rightarrow web.post(y_i)) \Rightarrow goal.post(x_i)) \end{aligned} \tag{2}$$

This formula should be shown to be a valid statement in F-logic before we can say that the web service completely satisfies the functional requirements of the goal. In this formula, x_i represents the free variables in the goal and y_j represents the free variables in the web service.

Informally, the formula above checks that the goal pre-condition logically implies the pre-condition of the web service (hence guarantying that the web service has all it needs before it gets executed) and that the goal pre-condition, together with the implicit statement of the web service functionality (that the web service pre-condition implies the web service post-condition) logically implies the goal post-condition, thus guarantying that the goal will get the desired result with the execution of the web service.

In our implementation of web service specifications and the matcher, we diverge slightly from the logical definition given above in order to take advantage of the meta-logical capabilities of Flora-2 i.e. insertion of new facts which is a meta-logic operation. Specifically, the post-condition of web services can contain the *insert* predicate of Flora-2, so that the post-condition, instead of just being stated as being true, is made to be true by insertion of facts into the knowledge base. Then the post-condition of the goal can be tested against the new knowledge base.

Listing 1 depicts how our matching logic is implemented in Flora-2. Predicate `%match` in line 1 takes two variables, `?goal` representing a goal object and `?WS` representing a web service object as its parameters. In line 2, a new variable, `?module` is defined and assigned to the web service tag. In line 3, a new Flora-2 module with the same name as the web service tag is created and the description of the goal object is loaded into it. In line 4, the pre-condition of the goal (*goal.pre*) is inserted into the created module. Then in line 5, by calling `%applyWebService(?WS)` the Flora-2 reasoner attempts to prove the web service functionality specified in the form of an *if-then-else* statement in the knowledge base module. If `%applyWebService(?WS)` is proven, this means that the pre-condition of the web service is logically implied by the pre-condition of the goal, and moreover the actions specified in the post-condition of the web service have been carried out. In line 6, the variable `?gPost` is assigned to the goal post-condition, and in line 7 its validity is checked against the current knowledge base. If the check succeeds, this means that the goal post-condition is logically implied by the web service post-condition. It should be clear that the `%match` predicate indeed verifies

the validity of the formula (2) that we defined as the meaning of a successful match between a goal and a web service.

Listing 1 The `%match` and `%matcher` predicates

```
1: %match(?goal,?WS) :-
2:       ?module=?WS.tag, //Specifies the name of module
3:       %loadGoal(?goal,?module),
4:       %insertGoalPre(?goal,?module), //Inserts
                                  goal.pre into the KB.
5:       %applyWebService(?WS), // If ws.pre gets true
                                  then ws.post will be
                                  inserted into the KB.
6:       ?goal[post -> ?gPost]@?module,
7:       ?gPost. // Checks whether goal.post is implied
                   by the ws.post.
8: %applyWebService(?ws) :-
     ?X = ?ws.def, ?X. // Tries to prove the web
                          service definition (ws.def).
9: %matcher(?goal,?WS) :-
     \if %match(?goal,?WS)
     \then writeln(['Goal',?goal,
                    'matches',?WS,'.'])@\prolog
     \else writeln(['Goal',?goal,
                    'does not match',?WS,'.'])@\prolog.
```

3 Use Case: Medical Appointment Finder

In this section we show how our approach can be used to describe the scenario in [8]. We will see that same results are achievable. However, our matching engine is much simpler.

The use case scenario is as follow: A patient named *Philip* wants to make an appointment with a specialist doctor (*ophthalmologist*) in *Montpellier* hospital located in a city of *France*. His preferred dates for this appointment are the days either before 19th or after 23rd (excluded) of the month. The patient should provide some basic information about himself, as well as a description of what he desires. Listing 2 shows a sample goal for this scenario rewritten in our specification format.

The patient provides the specialty *ophthalmology*, his name *Philip*, the hospital name he wishes to get the appointment from (i.e. *Montpellier*), and his age (which

is requested by the web service) in the form of an appointment request. What he wants is an appointment date either before the 19th or after the 23rd and an available specialist doctor's name.

For the web service side, web service pre and post-conditions have been given in listing 3. The web service uses a local database of Doctor instances containing information about doctors (i.e. *doctor1* and *doctor2*) and some general facts (i.e. *Montpellier* hospital is in *Paris*), and these are also depicted in listing 3.

Listing 2 Goal specification for the appointment use case

```
1: o_G03:c_Goal. //o_G03 is an object of the concept
                   c_Goal
2: o_G03[
3: pre -> ${RequestAppointment[specialty->Opthalmology,
4:                 patientName -> Philip,
5:                 appointmentDate -> ?_,
6:                 hospitalName -> MontpellierHospital,
7:                 age -> 22],
8:             livesIn(Philip,Paris)},
9:post -> ${Appointment[appointmentDate -> ?Date,
10:              doctorName -> ?DN,
11:              patientName -> Philip,
12:              hospitalName -> MontpellierHospital
13:             ],
14:            ((?Date < 19); (?Date > 23))}].
```

The web service provides some placeholders for its inputs while checking them over some predefined criteria (like, the patient must be at least 19 years old). Moreover, it checks whether the patient lives in the same city as the location of candidate hospital. After successful unification of inputs, the web service inserts all the possible appointments into the specified module (in this case `@WS01`) which is the common knowledge base between the web service and the goal. In this example, just the doctors with the specialty of *ophthalmology* who are working in *Montpellier* hospital are inserted into this module.

By referring to listing 1 again, we can see that all these actions take place through the call to `%applyWebService(?WS)` at line 5 in the `%match` predicate. At line 7, the Flora-2 reasoning engine attempts to prove the goal post-condition. At this point, the appointment date is checked to verify that it conforms to the constraints specified by the patient in the post-condition of the goal (i.e. either before the 19th or after the 23rd). This checking filters out those doctors who are not available during the requested dates.

Listing 3 Web service specification for the appointment use case

```
1: doctor1[
2:       doctorName -> Robert,
3:       specialty -> Neurology,
4:       hospitalName -> MontpellierHospital,
5:       availableDate -> 22
6: ]:Doctor.

7: doctor2[
8:       doctorName -> Green,
9:       specialty -> Opthalmology,
10:      hospitalName -> MontpellierHospital,
11:      availableDate -> 10
12: ]:Doctor.
13: hospital(MontpellierHospital,Paris).
14: o_WS01:c_WebService[
15:  tag -> WS01,
16:  def ->
17:   ${\if (RequestAppointment[specialty -> ?DS]@WS01,
18:   RequestAppointment[patientName -> ?PN]@WS01,
19:   RequestAppointment[appointmentDate->?Date]@WS01,
20:   RequestAppointment[hospitalName -> ?HN]@WS01,
21:   RequestAppointment[age -> ?X]@WS01,
22:   (?X > 18),
23:   livesIn(?PN,?city)@WS01,
24:   hospital(?HN,?city),
25:   ?doctor:Doctor[doctorName -> ?DN,
26:                  specialty -> ?DS,
27:                  hospitalName -> ?HN,
28:                  availableDate -> ?Date])
29:   \then (
30:     ?post = ${Appointment[appointmentDate->?Date,
31:                 doctorName -> ?DN,
32:                 patientName -> ?PN,
33:                 hospitalName -> ?HN]@WS01},
34:      %insert{?post}
35:   \else \false}].
```

If we changed the last line in listing 3 to `hospital(Montpellier Hospital, Berlin)`, the match would fail since the web service pre-condition would not be satisfied. Similarly, if we changed line 11 in listing 3 to `available Date -> 21`, again the match would fail, but this time due to the fact that the goal's post-condition would not be satisfied.

4 Related Work

A good explanation of WSMO and WSML can be found in [9] and [3]. The main available resource for Flora-2 is its user's manual containing useful examples of Flora-2 code [5].

In [10], the authors propose a framework for semantic web service discovery using FIPA multi-agents. They have a broker architecture and deal with OWL-S [16] rather than WSMO, as we do. In [11] the work presented is similar to ours, however the authors use WSML to specify goals and web services, which is very verbose. Furthermore, they do not state the proof commitments that are needed for a successful match in a logical way.

The closest and most comparable work to ours is [12], where the authors use Flora-2 to present a logical framework for automated web service discovery. Moreover, they use WSMO specification as the conceptual description of web services as we do. However, their specification of web services and goals are very involved, and they resort to Transaction logic for proof commitments. Our specifications, as apparent in the realistic example given in Sect. 3, are quite intuitive and simple. Furthermore, we make use of only Frame-logic for stating our proof commitments, which itself is equivalent to first-order predicate logic [4], and is much more accessible to the reader. The simplicity of our approach can be an important facilitator in its adaptation by industry (after necessary enhancements to deal with the web environment).

There are several surveys and reviews about semantical as well as non-semantical web service discovery proposals, which give a general overview of this field of study [13–15]. Many of the surveyed proposals are based on OWL-S. As M. Kifer et al. stated in [12], such approaches rely on subsumption reasoning [17] and due to the lack of rules in OWL, they are not able to exactly guarantee goal post-conditions.

5 Conclusion and Future Work

We have demonstrated how Flora-2 can be used as a convenient and expressive way to model semantic web services conforming to WSMO. We have also shown that reasoning for semantic web service discovery can be done very effectively by relying on the underlying Flora-2 engine and its meta-level capabilities. Our work can readily be the basis of implementing strategies for semantic web service composition and choreography. We intend to concentrate our efforts for realizing this goal in the near future.

References

1. OASIS: Reference Architecture Foundation for Service Oriented Architecture Version 1.0. Committee Specification 01 (2012)
2. Bruijn, J., Bussler, C., Domingue, J., Fensel, D.: Web Service Modeling Language. http://www.w3.org/Submission/WSMO/ (2005)
3. Fensel, D., Kifer, M., Bruijn, J.: D16.1v1.0 WSML Language Reference (WSML Final Draft). http://www.wsmo.org/TR/d16/d16.1/v1.0/ (2008)
4. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object oriented and frame-based languages. J. ACM (JACM) **42**(4), 741–843 (1995)
5. Kifer, M., Yang, G., Wan, H., Zhao, C.: Flora-2: User's Manual, Version 1.0 (Cherimoya). Department of Computer Science, Stony Brook University, Stony Brook, NY 11794–4400, U.S.A. (2014)
6. Chen, W., Kifer, M., Warren, D.: HiLog: a foundation for higher-order logic programming. J. Logic Program. **15**(3), 187–230 (1993). doi:10.1016/0743-1066(93)90039-J
7. Kifer, M., Bonner, A.: Logic Programming for Database Transactions in Logics for Databases and Information Systems. Kluwer Academic Publication, Dordrecht (1998)
8. Sharifi, O., Bayram, Z.: Matching Goal and Semantic Web Services in Flora-2: A Logical Inference Based Discovery Agent. submitted for publication (2015)
9. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services, The Concepts of WSMO, pp. 63–81. Springer, Heidelberg. http://dx.doi.org/10.1007/978-3-540-34520-6_6. Accessed 1 Jan 2007
10. Neiat, A., Mohsenzadeh, M., Forsati, R., Rahmani, A.: An agent-based semantic web service discovery framework. In: International Conference on Computer Modeling and Simulation, 2009 (ICCMS '09), pp. 194–198. IEEE (2009). doi:10.1109/ICCMS.2009.75. ISBN:978-0-7695-3562-3
11. Sirbu, A., Toma, I., Roman, D.: A Logic-based Approach for Service Discovery with Composition Support, pp. 101–116. Whitestein Series in Software Agent Technologies and Autonomic Computing, Emerging Web Services Technology (2007)
12. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: Proceedings of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (2004)
13. Duy Ngan, L., Kirchberg, M., Kanagasabai, R.: Review of semantic web service discovery methods. In: 6th World Congress on Services (SERVICES-1), pp. 176–177 (2010). doi:10.1109/SERVICES.2010.85. ISBN:978-1-4244-8199-6
14. Kster, U., Lausen, H., Knig-Ries, B.: Evaluation of Semantic Service Discovery? A Survey and Directions for Future Research. Emerging Web Services Technology, Volume II, Whitestein Series in Software Agent Technologies and Autonomic Computing 2008, pp. 41–58
15. Malaimalavathani, M., Gowri, R.: A survey on semantic web service discovery. In: International Conference on Information Communication and Embedded Systems (ICICES), pp. 222–225. IEEE (2013). doi:10.1109/ICICES.2013.6508208. ISBN:978-1-4673-5786-9
16. Martin, M., Burstein, M., Hobbs, J.: OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/ (2004)
17. Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, W3C Recommendation, W3C (2014)