



WSML Deliverable
D5.2 v0.1
WSMO DISCOVERY ENGINE

WSML Working Draft – October 29, 2004

Authors:

Uwe Keller
Rubén Lara
Holger Lausen
Axel Polleres
Livia Predoiu
Ioan Toma^a

^aIn alphabetic order

Editors:

Rubén Lara
Holger Lausen

Reviewers:

This version:

<http://www.wsmo.org/2004/d5/d5.2/v0.1/20041029/>

Latest version:

<http://www.wsmo.org/2004/d5/d5.2/v0.1/>



Abstract

Semantic web services promise to add automation and dynamics to current web service technologies, considerably reducing the effort required to integrate applications, businesses and customers. One of the key tasks in the integration process is to locate services that can fulfill the application, business or customer needs. With current web service technologies, this is done mainly manually, which reduces the accuracy of the search and requires considerable effort. Semantic web services aim at providing formal descriptions of requests and web services that can be exploited to automate several tasks in the web services usage process, including dynamic discovery of services.

WSMO discovery provides a conceptual model for service discovery that exploits WSMO formal descriptions of goals and web services. In this document, and based on the theoretical foundations provided by WSMO and WSMO discovery, we investigate the implementation of a discovery engine that can provide dynamic web service discovery. We focus on the language requirements to formalize WSMO goals and web service capabilities, on the integration of different approaches to web service discovery defined by WSMO discovery, and on the definition of the interface and architecture of the discovery engine.



Contents

1	Introduction	4
2	Expressivity and Reasoning Support Required	5
2.1	Keyword-based Discovery	5
2.2	Discovery Based on Simple Descriptions of Services	6
2.2.1	Non-restricted Expressivity	6
2.2.2	DL Restrictions	6
2.3	Discovery Based on Rich Descriptions of Services	7
2.3.1	Set-based modelling	7
2.3.2	Transaction Logic	8
3	Integration	9
3.1	Description of goals and web services	9
3.1.1	Modelling Styles	9
3.1.2	WSML Variants	9
3.2	Ranking	9
4	Interface	10
5	Architecture	11
5.1	Components	11
5.2	Requirements on other components	12
6	Testing and Benchmarking	13
6.1	Use Cases Tested	13
6.2	Benchmarking	13
7	Related efforts	14
7.1	WSML Reasoning Implementation	14
7.2	WSMX	14
7.3	WSMO4J	14
8	Conclusions and Future Work	15
8.1	Future Work	15
	Bibliography	16



1 Introduction

WSMO discovery [Keller et al., 2004] defines a conceptual framework for locating services that (totally or partially) fulfill a requester goal. This conceptual framework distinguishes three major steps in discovery: goal discovery, web service discovery, and service discovery. Goal discovery is about abstracting a concrete user goal to a pre-defined, reusable, and formalized goal. Web service discovery deals with the matching of formalized goals and formalize web services, selecting web services that can potentially be used to get the desired service. The last step, service discovery, uses the web services matched in the previous step to access the real services behind such web service interfaces, finally checking what services fulfill the requester goal.

In addition, [Keller et al., 2004] discusses different approaches to the second step of the conceptual model i.e. web service discovery, which offer different complexity, accuracy levels, and efficiency. Different notions of match and their respective formalizations are discussed for these approaches, providing a theoretical basis for web service discovery.

The aim of this document is to, based on the theoretical foundation provided by [Keller et al., 2004], implement a discovery engine with well-defined semantics and that can be applied in a wide range of scenarios.

In this version of the document, we will restrict ourselves to (different forms of) web service discovery. Further theoretical developments in goal and service discovery, to be provided by future versions of WSMO discovery, will eventually lead to an engine implementation covering the complete discovery process as defined in [Keller et al., 2004].

The document is structured as follows: Chapter 2 further defines the scope of Web Service discovery and discusses the reasoning support and the language expressivity required for implementing web service discovery. Chapter 3 discusses how the different approaches to web service discovery can work together. The interface of the discovery engine is presented in Chapter 4. Chapter 5 defines the architecture of the engine and poses requirements on components outside it. The use cases and benchmarking of the implemented engine are presented in Chapter 6. The relation of our discovery engine to the WSMO reference implementation WSMX¹, the WSMO reasoning implementation, and other related efforts is discussed in Chapter 7. Finally, Chapter 8 concludes the document and presents our future work.

¹<http://www.wsmx.org/>



2 Expressivity and Reasoning Support Required

In this chapter, we discuss the language expressivity required for the different approaches to web service discovery described in [Keller et al., 2004], together with the reasoning support necessary to find a match in each case.

The WSML family of languages [de Bruijn et al., 2004] offers a strictly layered set of languages with different expressivity. The different approaches to web service discovery will require different features from the language and, therefore, will use different WSML variants.

In addition, the proof obligations associated to each of the approaches to web service discovery can require different reasoning support to find a proof for a match. Therefore, the reasoning support needed and the available tools offering such support must be identified.

Eventually, the WSML reasoning implementation [de Bruijn, 2004] will offer the reasoning support required for each of the WSML variants, providing the reasoning engine that will be used by the WSMO discovery engine. However until specific WSML reasoning engines are available, our implementation will rely on existing tools. We will orient our selves along the lines of the language layering approach and syntax proposed in [de Bruijn et al., 2004]. Translations to the actual used implementation will be initially manually.

In the following, we will discuss issues for each of the web service discovery approaches. Here, we will not cover issues such as publication of web services or necessary registries; these issues will be discussed in future versions of this document. Instead, we will assume that a goal and the set of web services to be considered for the matching process are already defined and available to the discovery engine, and we will focus on what language expressivity these require and on what reasoning support the engine needs to find a match.

2.1 Keyword-based Discovery

As described in [Keller et al., 2004], **keyword-based Discovery** is one possible approach to do web service discovery. Although it does not use well-defined semantics and is limited because of the ambiguity of natural language, keyword-based discovery can be used to filter and rank quickly the huge amount of available goal and service descriptions.

In order to design a good keyword-based discovery mechanism some questions must be answered. For example, over what properties of goal/service formalized description the keyword-based discovery must be performed? It is known that each component description in WSMO [Roman et al., 2004] has a `nonFunctionalProperties` section. A basic and intuitive approach is to match the keywords provided by the requester against the keywords from `dc:subject` or against extracted keywords from natural language descriptions provided in `dc:description` elements of `nonFunctionalProperties`. The same approach can be performed by considering other parts of goal/service description, for example `dc:description` elements of axioms that describe goal postconditions or service capabilities and postconditions. One aspect that must be taken into account is that `nonFunctionalProperties` descriptions are not mandatory. So in case the author of goal/service dose not provide any `nonFunctionalProperties`



description, this goal/service can not be discovered using the approach that we described above.

Another approach for keyword-based discovery is to find relations between keywords provided by the user and concepts from the ontologies that are used to formalize goals/services. This basically implies to match keywords from the user request against the concept names from the ontologies.

Finally, another approach that can be considered is a keyword-based discovery over the logical formulae that are used in goal and service descriptions. Usually the names of predicates are chosen having in mind the semantics of what these predicate actually do. In essence, a keyword-based match can be performed given the keywords that the user specified on the one hand and the names of the predicates on the other hand.

Another question that must be answered is what tools can be used in keyword-based discovery. One tool that offers complete natural language processing functionalities is GATE [Cunningham et al., 2004].

I will add more about tools in the next version.

Keyword-based discovery uses basically textual descriptions that are not expressed using logics. In consequence this approach does not pose any requirement on the WSML variant to be used.

2.2 Discovery Based on Simple Descriptions of Services

In this section we consider the approaches for semantics-based discovery for simple semantic annotations of web services that are discussed in [Keller et al., 2004].

2.2.1 Non-restricted Expressivity

In [Keller et al., 2004] we discussed a set based modelling approach for web services and outlined its formalization in First-order languages. There we did not impose any restrictions on the language to be used for defining web services and goal. Thus, it is in general required to use a fully-fledged theorem prover for First-order languages in order to check the proof obligations that have been given in that document. Hence, the candidate WSML variants that we consider for this approach basically are WSML Core, WSML DL as well as the monotonic part of WSML Full.

There are a bunch of candidate reasoners available, that could be used for a prototype implementation. We will have a closer look at two specific systems, namely Vampire and SNARK, and the features they support.

Future versions of this document will elaborate on the selection of concrete theorem prover for a prototype and specific aspects of how to implement the checks for the proof obligations given in [Keller et al., 2004].

2.2.2 DL Restrictions

In the previous section, no restriction is posed on the allowed expressivity to describe goals and web services and, therefore, a first-order theorem prover is in principle required.

However, if we restrict the expressivity to WSML DL i.e. to a syntactical variant of OWL DL, we can efficiently exploit subsumption reasoning. Available



DL reasoners such as RACER¹ or FACT++² provide efficient subsumption reasoning for \mathcal{SHIQ} with incomplete reasoning with nominals, and for $\mathcal{SHIF}(\mathcal{D})$, respectively.

In [Li and Horrocks, 2003], RACER is used to classify web services in the T-Box, which is time-consuming but can be done off-line when publishing them. Once the T-Box is classified, experimental results show that checking the subsumption relation between the user request and the web services in the T-Box can be done within 20 milliseconds.

These results clearly suggest that using DL reasoners to index web services and/or pre-defined goals can be a useful and efficient mechanism to restrict more detailed (and probably more expensive) checking to a (ideally small) subset of all the available web services.

The use of pre-defined, formalized goals is expected in WSMO discovery. If we restrict such goals to be described in WSML DL, we can off-line classify them in a DL reasoner T-Box. When publishing a web service, its subsumption relationship with the classified pre-defined goals can be checked, and a wgMediator[Roman et al., 2004] can be generated to link the web service to the pre-defined goal it (totally or partially, depending on the subsumption relation computed) fulfills. In this way, and as goal discovery should result on a (parameterized) pre-defined goal, and web service discovery consists of matching these pre-defined goals against published web services, web service discovery is reduced to exploring the wgMediators associated to such pre-defined goal.

Another option would be to classify the web services when they are published. However, it is expected that the number of web services available will be considerably higher than the number of pre-defined goals and, therefore, we would have to deal with bigger T-Boxes and worse classification times. Furthermore, in this case the subsumption relation between a pre-defined goal and the classified web services would have to be computed for each discovery request, while in the previous solution subsumption checking is only required once for each web service, and only at publication time.

For this approach, the candidate WSML variant is WSML DL, and the candidate reasoners are RACER and FACT++. From these two, RACER provides the closest features to the requirements for an WSML DL reasoner, namely reasoning for WSML DL with limited support for nominals, while FACT++ does not support neither nominals nor number restrictions.

Future versions of this document will investigate in more detail the modelling effort required to describe pre-defined goals and web services in this way, and the modelling and testing of some simple examples.

2.3 Discovery Based on Rich Descriptions of Services

2.3.1 Set-based modelling

In [Keller et al., 2004] we discussed the extension of the set based modelling approach for web services to rich semantic descriptions which capture the actual relationship between inputs and outputs/effects of web service executions as well and gave the formalization in First-order languages.

In principle, the language expressivity needed for expressing the discussed proof obligations is the same as in the case of simple semantic annotations (without restrictions on the service and goal descriptions), that means we have

¹<http://www.sts.tu-harburg.de/~r.f.moeller/racer/>

²<http://owl.man.ac.uk/factplusplus/>



to do reasoning over First-order formulae. Thus, it is in general required to use a fully-fledged theorem prover for First-order languages in order to check the proof obligations that have been given in that document. Basically, the very same techniques and systems that we discussed in Section 2.2.1 can be used for an implementation here.

The candidate WSML variant that we consider for this approach basically is the monotonic part of WSML Full.

2.3.2 Transaction Logic

The use of transaction logic [Bonner and Kifer, 1998] for web service discovery has been described in [Keller et al., 2004]. An implementation of a slightly different proof obligation for web service discovery using transaction logic has been presented in [Kifer et al., 2004].

At the moment, is not completely clear what WSML variant will be required. Based on the work done in [Kifer et al., 2004], we envision that built-in equality and meta-modelling will be required. However it is not clear whether non-stratified negation will be needed. In addition, rules reification will be most likely required for the description of pre-defined goals and web services. This is not explicitly considered in any of the WSML variants, but it could be added to WSML-Rule.

Therefore, we estimate that WSML-Flight or, more likely, WSML-Rule will be used. Additionally, transaction logic will be obviously used to prove the proof obligation defined in [Keller et al., 2004]. However, this does not pose any extra requirement on the language needed to describe goals and web services, but only on the language supported by the prover. Currently, *FLORA-2*³ is the only prover which supports transaction logic.

Future versions of this document will further explore the need for rule reification in WSML-Rule and will include the modelling and testing of some simple examples with *FLORA-2*.

³<http://flora.sourceforge.net>



3 Integration

As has been seen in the previous chapter, different approaches to web service discovery will require different WSML variants and different reasoning support. In this chapter, we discuss how the different approaches can work together.

3.1 Description of goals and web services

Goals and web services might be described in a different way depending on the kind of web service discovery to be used. For example, the postconditions and effects will not include the relation to the input when using simple semantic annotations for discovery, while they need to be described if the functionality of the web service and not only the kind of results it provides is to be considered. In addition, the approaches discussed in the previous chapter will use different WSML variants. This section discusses these issues.

3.1.1 Modelling Styles

Here, we will discuss whether it is possible to use a single goal and web service description for all the kinds of web service discovery and, if so, what kind of translation or preprocessing is needed to adapt the single description to the needs of the concrete approach used.

3.1.2 WSML Variants

In future versions of this deliverable, we will discuss how the different WSML variants used for the different kinds of web service discovery can be made compatible.

Any kind of web service discovery will in principle use the same domain ontologies. Therefore, it is very relevant to be able to describe such ontologies only once, without requiring the duplication of the ontologies to fit the needs of the different approaches. It is foreseen that, provided that such ontologies are described in WSML-Core, a single definition of the ontology will be sufficient. Furthermore, and according to the study presented in [Volz, 2004], most of the currently available ontologies will be expressible in WSML-Core.

However, the descriptions themselves will require different WSML variants. We will study in the future what the implications of this are.

3.2 Ranking

If the different approaches to web service discovery are to be combined, a ranking of the web services matched should be clearly defined. This requires not only to rank web services matched using e.g. keyword-based discovery, but also to rank these with respect to the matching results given by e.g. discovery based on transaction logic.

Future versions of this document will investigate this issue.



4 Interface

In this chapter, we discuss what interface the discovery engine will use. We restrict ourselves to the interface for web service discovery. When goal discovery is added, this interface will be subject to changes.

We expect the interface to adhere to the following rough description:

Given a formalized goal, an approach to web service discovery (keyword-based, simple, rich), and the notion of match expected to be used at that level (exact, plug-in, etc.), the engine should return a list of web services that are a (total or partial) match, together with a ranking of such web services.

The detailed interface will be described in future versions of this document.

We will have a look at already developed interfaces in different projects, i.e. Semantic Web Fred (SWF)¹, WSMX², Semantic Web enabled Web Services (SWWS)³ and take the approaches into account when making a more detailed design.

¹<http://www.deri.at/research/projects/swf/>

²<http://www.wsmx.org/>

³<http://sws.semanticweb.org/>



5 Architecture

In this chapter, we outline the architecture of the discovery engine, which at the moment only includes the components necessary for web service discovery. Future versions will extend it to goal discovery and service discovery.

5.1 Components

The high-level architecture of the discovery engine is depicted in figure 5.1.

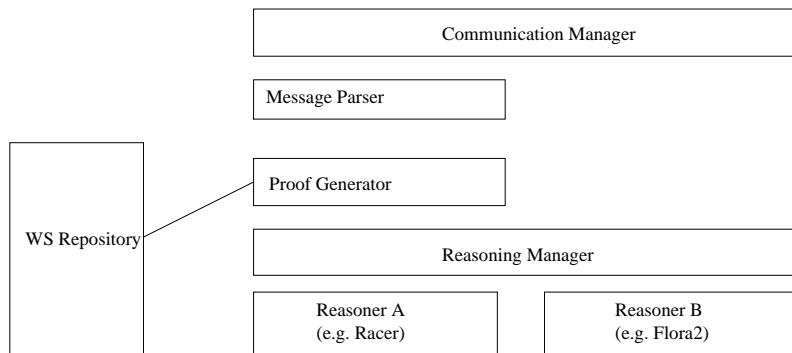


Figure 5.1: Discovery Engine Architecture

The Communication Manager handles the incoming requests, that can be for example goals, the exact messages will be defined in 4. The Message Parser analyzes the incoming message and extracts the message components such as postconditions or effects and hands them to the Proof Generator.

Note: Here we might add a translation step, which transforms the logical expression contained in the incoming WSML messages to the target format of the reasoner. It could be argued that this step is better done within the reasoning manager, however it will be more efficient to construct the proof obligation in the native format of the target reasoner.

Depending on the requested type of match the Proof Generator constructs logical formulas e.g. a subsumption check between two class description for a plug-in match. This step already needs the web service descriptions. Those will also be translated into the format required by the target reasoner.

The Reasoner Manager abstracts from the different interfaces of the reasoners and offers a common API to the Proof Generator. Furthermore such manager is responsible for load balancing the reasoners and network protocol related issues.

Notice that in the architecture the web services repository is part of the discovery engine i.e. we see the discovery engine integrated with the registry. The reason is that if the registry does something more than giving all the web services (at the same time or one by one) to the discovery engine, it already performs some kind of discovery e.g. keyword-based discovery such as UDDI.

Another option is to assume a registry that contains the complete descriptions of the web services together with a unique identifier for every of them, with an API that provides the complete description of a web service given its identifier. Then, the discovery engine would only need to store a copy of the relevant



information for discovery of every web service in the registry, together with its identifier. Complete descriptions would only be retrieved from the registry for matching web services. However, this approach needs to solve synchronization problems between the registry and the (partial) copy stored in the discovery engine.

5.2 Requirements on other components

Here we describe the requirements posed by the discovery engine, both at the interface and functionality level, on other components e.g. registry or ontology repository.



6 Testing and Benchmarking

6.1 Use Cases Tested

Future versions of this document will describe the use cases for testing.

6.2 Benchmarking

The benchmarking of the engine considering the different types of web service discovery and the use cases above will be included in this section.



7 Related efforts

In this chapter, we outline what the efforts related to the discovery engine are, and what is the nature of such relation.

7.1 WSML Reasoning Implementation

In [de Bruijn, 2004] an ongoing effort to provide a reasoning implementation for the different WSML variants is presented. The reasoner that will result from this work is of high relevance for the implementation of the discovery engine. As discussed in previous sections, our first outline of the discovery engine considers different existing reasoners for providing the reasoning support required for web service discovery. However, as [de Bruijn, 2004] will provide an integrated reasoner for all the WSML variants, it is our intention to use this reasoner for the final implementation of the discovery engine.

7.2 WSMX

The Web Services Execution Environment (WSMX)¹ is an execution environment for dynamic mediation, selection and invocation of web services. WSMX is the reference implementation of Web Service Modeling Ontology (WSMO) [Roman et al., 2004].

The current version of WSMX includes a very basic discovery implementation. Part of the current discovery functionality in WSMX is provided by WSMX Matchmaker. This component implements a very simple matchmaking algorithm based on string matching.

The integration of the discovery engine with WSMX will be studied in the future version of this document.

7.3 WSMO4J

WSMO4J² will provide a data model in Java for WSML and (de-)serializers for the different WSML syntaxes. It will additionally provide wrappers for different Reasoners, the first one according to the current plans will be KAON2³. Others are not yet scheduled by the core development team, however also 3rd parties can extend this open source API.

The discovery engine will rely on WSMO4J for the parsing of incoming messages and use its Java model for internal processing. This might also include the transformation of logical formulas, however the current API does not yet support fine grained access on WSML logical expressions.

¹<http://www.wsmo.org/wsmx/>

²<http://wsmo4j.sourceforge.net/>

³<http://sourceforge.net/projects/kaon2/>



8 Conclusions and Future Work

In this deliverable we have provided a first overview of how the WSMO discovery engine will perform web service discovery, and of the WSML variants and reasoning support required. We have briefly outlined the problems that are to be solved to integrate the different approaches to web service discovery. In addition, we have presented our first ideas on the interface and architecture of the engine. Finally, we have identified related efforts and how they are expected to contribute to the development of our discovery engine.

8.1 Future Work

The future work will include a more detailed evaluation of the languages required, further ideas on how the different approaches to web service discovery can be integrated, and more detailed interface and architecture definitions. The modelling of some examples for the different approaches and the testing of the reasoners required will be also included in future versions of this document. The evolution of WSMO discovery [Keller et al., 2004] will be closely followed and this document updated accordingly. Finally, our work will lead to an implemented engine for WSMO discovery.

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, and Esperanto; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program.

The editors would like to thank to all the members of the WSML working group for their advice and input into this document.



Bibliography

- [Bonner and Kifer, 1998] Bonner, A. and Kifer, M. (1998). A logic for programming database transactions. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers.
- [Cunningham et al., 2004] Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., Dimitrov, M., Dowman, M., and Aswani, N. (2004). Developing language processing components with GATE. Technical report.
- [de Bruijn, 2004] de Bruijn, J., editor (2004). *WSML Reasoning Implementation*. WSMO Working Draft D16.2v0.1. Available from <http://www.wsmo.org/2004/d16/d16.2/v0.1/>.
- [de Bruijn et al., 2004] de Bruijn, J., Foxvog, D., Lausen, H., Oren, E., Roman, D., and Fensel, D. (2004). The WSML Family of Representation Languages. Deliverable D16v0.2, WSMO, <http://www.wsmo.org/wsml/>. Available from <http://www.wsmo.org/2004/d16/v0.2/>.
- [Keller et al., 2004] Keller, U., Lara, R., and Polleres, A., editors (2004). *WSMO Discovery*. WSMO Working Draft D5.1v0.1. Available from <http://www.wsmo.org/2004/d5/d5.1/v0.1/>.
- [Kifer et al., 2004] Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., and Fensel, D. (2004). A logical framework for web service discovery. In *Workshop on Semantic Web Services at ISWC 2004*.
- [Li and Horrocks, 2003] Li, L. and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, Budapest, Hungary.
- [Roman et al., 2004] Roman, D., Lausen, H., and Keller, U. (2004). Web service modeling ontology standard (WSMO-standard). Working Draft D2v1.0, WSMO. Available from <http://www.wsmo.org/2004/d2/v1.0/>.
- [Volz, 2004] Volz, R. (2004). *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe.