WSML Deliverable
D5.2 v0.1
Inferencing Support for Semantic Web Services:
Tools for Semantic Support
WSML Working Draft – April 19, 2004

Authors:
Uwe Keller

Editors:
Uwe Keller
Axel Polleres

This version:
http://www.wsmo.org/d5/d5.2/v0.1/20040419
Latest version:
http://www.wsmo.org/d5/d5.2/v0.1/
Previous version:
http://www.wsmo.org/d5/d5.2/v0.1/20040405

Copyright © 2004 DERI®, All Rights Reserved.
DERI liability, trademark, document use, and software licensing rules apply.
Abstract

In this document we present and discuss the tool suite that implements the needed support for automated and semi-automated checks of semantic properties of Semantic Web Services described by means of the Web Service Modeling Ontology (WSMO) and the Web Service Modeling Language (WSML).

The tools suite is develop for the purpose of rigorously proving the formal statements defined and described in the corresponding deliverable D5.1, so-called \textit{proof obligations}.

In particular, we describe the underlying principles for verifying proof obligations automatically as well as semi-automatically and explain the tools and techniques we have applied and developed for that purpose.

Together, WSMO and WSML as well as our tool suite constitute the necessary semantic infrastructure that is needed for making semantic web services come true. Only such an infrastructure allows us to exploit the full-potential of web service technology in the context of Enterprise Application Integration (EAI) and fully-dynamic eCommerce.

The current version of this document is in many ways incomplete. Therefore, please consider this only as a living ,,draft” of the deliverable that will continuously be extended and enhanced in the course of the project.
Contents

1 Introduction ........................................ 4
   1.1 Motivation ........................................ 4
   1.2 Goal of this document ................................ 6
   1.3 Overview of the paper ................................ 6

2 Semantic support in the context of WSMO and WSML .......... 7
   2.1 Where can semantic annotation be exploited? ............ 7
      2.1.1 Service Discovery .................................... 7
      2.1.2 Service Composition ................................... 7
      2.1.3 Service Invocation .................................... 7
      2.1.4 Service Mediation on the data, protocol and process level .... 7
      2.1.5 Service Execution Monitoring .......................... 7
      2.1.6 Service Compensation ................................. 7
   2.2 The general Principle: Proof Obligations .................. 7
      2.2.1 What is a proof obligation? ............................ 7
      2.2.2 Benefits of the Proof Obligation Approach .............. 8

3 Tools for Semantic Support .................................. 9
   3.1 Service Discovery ....................................... 9
      3.1.1 Goal-Capability-Matching .............................. 9

4 Conclusions ............................................. 13
   4.1 What have we achieved so far? ............................ 13
   4.2 Future Work ........................................... 13

5 Acknowledgements ......................................... 15
1 Introduction

1.1 Motivation

The World Wide Web (WWW) has been invented in 1989 by Tim Berners-Lee [BLFD99] and since then changed the way people gather and access knowledge in everyday life drastically. At its invention, nobody was able to foresee the effects and influences of this new promising technology. In fact, the WWW can be considered as one of the most influential technologies that have been invented in the 20th century.

Nowadays the World Wide Web is certainly the biggest information repository on earth and contains a significant amount of the knowledge of mankind. But the major bottleneck when it comes to exploiting the represented information is how to find the required knowledge: the Web is still a mainly syntactically-driven information repository, where information is represented in form of web pages that actually lack to capture the semantics of their contents. As a consequence, the content of the pages is not easily accessible by computers but instead exclusively intended for consumption by human beings. Given the size and the rapid growth of the WWW, this approach doesn’t scale up.

This has lead to significant efforts on the invention of the so-called Semantic Web [BLHL01, DFvH03, FHLW03], that will enable computerized agents to easily gather information in the Web and exploit this knowledge on behalf of their human clients. By enriching the information repository WWW with machine-processable semantics, the annoying bottleneck shall become overcome.

From Web Services to Semantic Web Services. Web services add a new level of abstraction on the current Web. Whereas at the moment the Web mainly is constituted by simple documents (e.g. static and dynamic web pages), in the future the Web might become a repository of electronic (web) services. Each service represents a computational entity that can be accessed and invoked over the Internet. Thus, the Web might become a huge, distributed, heterogeneous and shared computational device.

Recently, web services gain more and more interest in the industry as well as among academics. This is partly due to what they promise: a uniform infrastructure for accessing software-systems over the Internet which can be used for integration of legacy systems – even across enterprise boundaries. Nonetheless, though web service simplify enterprise application integration drastically, the technologies around web services – namely SOAP, WSDL and UDDI – do not solve the integration problem by themselves. Again, the integration technologies are exclusively syntactical means that lack the consideration of semantics.

Moreover, given the growing industrial interest in web services, one can expect a huge amount of web services for various tasks and computations. Hence, the same bottleneck as for the conventional WWW arises: How can we find a web service that solves one of our problem at hand? The corresponding technologies SOAP, WSDL and UDDI can resolve the bottleneck only partially. Again, machine-processable semantics for web-services is needed in order to ensure access of web services for agents without human interaction. Adding semantics to web services is the main prerequisite for an approach that scales.

This semantically enriched web services are called Semantic Web Services. In a sense, we can use the following equation in order to describe the relationship between web services and semantic web services symbolically:

\[
\text{Semantic Web Service} = \text{Web Service} + \text{Semantic Annotation}
\]
And what do we do with all that Semantics? The major benefit of any form of specification is that by explicitly documenting what an artifact actually does, it becomes a lot easier for humans (who don’t know that artifact before) to comprehend what this thing actually can be used for.

Clearly, with a formal representation of any specification, the use of an artifact becomes more precise\(^1\) and eventually accessible for computerized agents.

Knowledge about the capabilities of single objects can be exploited for combining these objects to systems which solve more complex problems than each of their elements.

In general, the system-construction process itself is a highly creative task – at least it requires an understanding of the semantics of the single elements that can be combined to a system. Regarding the design of software architectures for instance [AG97a, AG97b] show the benefit of formal specification of architectural elements (in particular so-called connectors) for automated and semi-automated validation of architecture designs and the potential impact on software quality in early phases of the software development process. In these works for example formal descriptions of interactions are exploited to check whether given elements of an architecture that are supposed to interact can actually interact without any problems.

Each software system is build for some purpose: to solve a specific well-defined problem of a client or organization. Software architectures themselves are the conceptual means for describing how the functionality of a system is provided and how the given requirements are actually satisfied. In this sense software architectures [GS96, BCK03] bridge the gap between the user requirements and the level of computational units that provide some well-defined functionality and interact and communicated with each other.

In a world of rapidly changing requirements and incomplete knowledge (such as the Semantic Web with it’s open and heterogeneous user community), the problems to be solved most likely occur rather ad-hoc and can’t be forseen in detail. In this case, the construction of a system that solves the problem at hand, in general can’t be done in advance, but has to be done on the fly instead. Thus, software-systems composed by services in the Web constitute a particularly interesting and challenging problem for the theory and practice of software architecture: how to dynamically construct systems out of existing ones in a goal-driven way.

In case of a formal specification, the knowledge about the possible elements of a system to solve a problem is represented in a machine-processable manner.

On the other hand – when considering the semantic web –, all efforts in semantic annotation (resp. specification or description) of web services are far from being actually effective as long as there is no way for computerized agents (instead of human beings) to actually process this kind of information about the services. Only then, the dynamic composition of software-systems could be achieved.

In this context, „processing“ means to apply the formalized knowledge intelligently to solve a given problem, that is, to actually exploit the semantics of the given description. In one way or the other, this drills down to performing simple inference steps over existing knowledge until enough knowledge has been generated to solve the task.

Thus, agents need support for reasoning about semantic web service descriptions in order to effectively utilize semantic annotations.

The big picture. Together, WSMO and WSML as well as the semantic tool suite developed in the context of deliverable D5 constitute the necessary se-

---

\(^1\) Usually even unambiguous in a mathematical sense!
mantic infrastructure that is needed for making semantic web services come true. Such an infrastructure is a basic building block for semantically discovering services and dynamic composition of services. Thus, it is a major necessary prerequisite for enabling the exploitation of web service technology’s full potential in the context of Enterprise Application Integration and Electronic Commerce\(^2\) [Fen03].

### 1.2 Goal of this document

In deliverable D5.1 of WSML we identify a set of tasks for which we want to exploit semantics annotation of web services. There, we especially analyze the different tasks and define the corresponding proof obligations.

In this document we present and discuss briefly how we actually check (resp. prove) the proof obligations that are defined in deliverable D5.1. For this purpose, we describe the various tools that implement means for formal checking of proof obligations and discuss, why we have followed the described approach.

Eventually, we evaluate the tool suite developed in the course of the project during case-studies and report about the results and conclusions here. Of course, since we are still in a very early stage for this deliverable and the use-case studies, this part of the document can only be included in the future.

### 1.3 Overview of the paper

Section 2 recapitulates briefly the various areas in the process of dynamically building web-service-based applications where information about the semantics of a service as well as knowledge about his properties can be fruitfully applied. Furthermore, we explain the general principle on which our tool suite is based: The principle of

In section 3 we describe the various tools that are meant to support the reasoning tasks on web service descriptions in the context of WSMO and WSML.

Finally, in section 4 we summarize our current achievements and sketch future work and ideas.

---

\(^2\)One imagines for instance the potential benefit of dynamic configuration of supply chains for manufacturing companies under consideration of the current market state.
2 Semantic support in the context of WSMO and WSML

This section recapitulates briefly the various areas in the process of dynamically building web-service-based applications where information about the semantics of a service as well as knowledge about its properties can be fruitfully applied.

Afterwards, we then explain the general principle that has been applied in deliverable D5.1 and which is underlying the tools suite and the benefits that we expect to achieve by applying this principle for the design of tools for semantic support.

2.1 Where can semantic annotation be exploited?

According to deliverable D5.1, we have to consider the following areas where elements of semantic descriptions of web services can be exploited. In each of these areas, we are looking for possible concrete reasoning tasks.

2.1.1 Service Discovery
2.1.2 Service Composition
2.1.3 Service Invocation
2.1.4 Service Mediation on the data, protocol and process level
2.1.5 Service Execution Monitoring
2.1.6 Service Compensation

Current restrictions. For the moment, we only consider WSMO-Standard [KLR04]. This excludes some important aspects that are included in the WSMO extension WSMO-Full. These aspects are mainly concerned with the business or application layer of WSMO, e.g. concepts like contract, contract templates, negotiations and protocols for the pre-negotiation phase and the post-negotiation phase etc.

Currently, WSMO-Full is in a very early and rapidly evolving stage. In the final version of WSMO-Full it is our goal to cover all the related problems in detail. But at a first attempt, we focus here on specific reasoning tasks within WSMO-Standard, e.g. a goal-driven service discovery.

Moreover, in this version of the document, we focus on one specific aspect of the whole process: the service discovery task; and in particular on one specific aspect therein: the goal-capability-matching which lies in the very heart of the service discovery and is the most fundamental element of service discovery where automation is highly desired.

2.2 The general Principle: Proof Obligations

2.2.1 What is a proof obligation?

Given a set of semantic web services, our desire is to check and become sure about particular “real-world” properties of elements of this set, of compositions of such elements as well as the relationship between such elements. As mentioned in section 1, we principally aim at automated or semi-automated construction of
dynamic software-systems with well-understood properties, whose architectural
paradigm is exclusively based on web-services.

The question whether a given composition of web service fulfills a requested
task or whether two web services can interact with each other (and thus can
actually be connected architecturally within a system) are examples for such
real-world properties of interest.

As a first step, we have to come up with a precise mathematically-based
definition of the properties of interest.

Afterwards, we principally have to translate this definition in statement of
some formal language, that allows us to check the statement by using algorithms.
The translation has to preserve the semantics of the original definition or can be
strengthened\(^3\). Strengthening of the definition can be necessary because of the
available information about the single elements or computational complexity
resp. efficiency of algorithmic checks, but should be kept to a minimum level.

The resulting statement in a formal language is called \textit{proof obligation}.

\subsection{2.2.2 Benefits of the Proof Obligation Approach}

In this section, we argue that proof obligations not only represent a nice
conceptual approach for dealing with reasoning tasks, but that their explicit
use in the design of the architecture of semantic tools within our tools suite can
be rather beneficial in some cases (wrt. a practical setting).

\begin{quote}
The basic idea: Formulate proof obligations as independent as possible from
the concrete reasoning tool that has to check the PO. Consider the language
used for formulating the PO as an intermediate language and translated the PO
in a separate step into the input language of some (perhaps pre-existing tool).
This allows us to use different tools for checking the same PO. Basically we are
free to decide which tool we want to use for a specific.

In many cases, the translation into the input language of the tool might be
straightforward, in other cases it might be more involved and perhaps not be
possible for every PO. Example: Logic Programming Systems vs. Full reasoners
for F-Logic.
\end{quote}

\(^3\)More precisely, this means finding some criterion \(P\) with the property that if this criterion
is valid then the original defined criterion \(C\) is valid. Please note, that it’s not required that
the set of models is of \(P\) is subset of the models of \(C\). Both criterion might also use different
formal languages
3 Tools for Semantic Support

3.1 Service Discovery

3.1.1 Goal-Capability-Matching

In this section we discuss our approach for dealing with the discovery subtask Goal-Capability-Matching.

As discussed in deliverable D5.1 the proof obligation for Goal-Capability-Matching is as follows:

Given a goal

\[ G = (\Phi^{post}, \Phi^{eff}, O_G, M_G) \]

and a capability

\[ C = (\Psi^{pre}, \Psi^{ass}, \Psi^{post}, \Psi^{eff}, O_C, M_C) \]

the proof obligation for Goal-Capability-Matching is to prove the following relation

\[ PO^{gcm}(G, C) \equiv \{O_C, O_G, M_C, M_G\} \vdash \text{Cl}_3(\exists in_1, \ldots, in_n: (\Psi^{pre} \land (\Psi^{post} \rightarrow \Phi^{post}) \land (\Psi^{eff} \rightarrow \Phi^{eff}))) \]

By using sound and complete deduction calculus \( C \) that defines the provability relation \( \vdash \), this is equivalent to show within the calculus that the set of conclusions can be derived from the set of premisses:

\[ PO^{gcm}(G, C) \equiv \{O_C, O_G, M_C, M_G\} \vdash \text{Cl}_3(\exists in_1, \ldots, in_n: (\Psi^{pre} \land (\Psi^{post} \rightarrow \Phi^{post}) \land (\Psi^{eff} \rightarrow \Phi^{eff}))) \]

Such a calculus for full F-Logic is discussed in detail in [KLW95]. Hence, we can deal with the proofobligation \( PO^{gcm}(G, C) \) (and thus with this reasoning task) in an automated way, as it is needed for our purpose. Nevertheless, we have to keep in mind some practical issues.

Practical concerns. F-Logic is at least as expressive as classical First-Order Logic for which logical entailment is not decidable. It’s straightforward to see that thus logical entailment for full F-Logic is not decidable, too. But because there is a sound and complete proof system for logical entailment, this relation is recursively-enumerable (resp. semi-decidable).

This has the following consequence for practical applications: Given a capability \( C \) and a goal \( G \) for which we want to know whether \( C \) matches \( G \). We can proceed as follows: Compute the proof-obligation \( PO^{gcm}(G, C) \) and start an automated theorem prover (ATP) for full F-Logic. If \( PO^{gcm}(G, C) \) holds, then the ATP will find a proof for \( PO^{gcm}(G, C) \) in finite time. On the other hand, if \( PO^{gcm}(G, C) \) doesn’t hold, then the ATP won’t stop in general\(^4\). Unfortunately, even in the first case, the time for getting the answer from the ATP is undefined, that means there might be cases, where the prover needs a very long time to find a proof. In general, it’s not possible to distinguish both cases, if we don’t get an answer for a long time.

\(^4\)But it might stop for some particular cases.
How can we deal with this situation? One pragmatic approach to this problem is to set and use a time-limit, for finding the answer. If the proof obligation has not been established within that period of time, we stop the prover and assume that the capability doesn’t match the goal (resp. at least we don’t assume that they match!). Obviously, this again restricts our notion of Goal-Capability-Matching further and is only applicable, when a suitable number of actual matchings can be detected in that way.

A second approach is to restrict the language for specifying goals and capabilities in such a way, that we end up with a language where our specific proofobligation $PO_{gcm}(G, C)$ is decidable (for all $G$ and $C$ in the restricted language)
. This approach might be limited by the application domain requirements on the expressiveness of the language for specifying semantic services.

The following third approach can be seen as an intermediate approach between the two mentioned first: We restricts the syntax of the language for in such a way, that our proof-obligation is not guaranteed to be decidable but very efficient and specialized proof-search techniques can be applied. Thus we expect to find (existing) proofs several orders-of-magnitude faster in average.

In our opinion all theory is grey and the feasibility resp. the infeasibility of one approach or the other has to be analyzed by some case-studies.

Reasoning Support For now, we don’t want to restrict the language for expressing goals and capabilities unless this approach turns out to be totally unfeasible. That means, that arbitrary F-Logic formulas can be used for the definition of this descriptive elements of WSMO and WSML.

Applying the Logic Programming Paradigm.

Alternative Approaches. As mentioned above, we can use a sound and complete proof calculus for full F-Logic for establishing the goal-capability-matching. Such a calculus for instance has been described in [KLW95].

To the best of our knowledge, there is no currently no implementation of a reasoning system for full F-Logic. The only implemented reasoning systems based on F-Logic (e.g. Flora2 [YK], Florid [FHK+97], Ontobroker [DEFS99]), are logic programming or knowledge-base systems, but no full reasoners for F-Logic. Thus, we would like to implement an inference engine for full F-Logic.

Basically, there are two possible approaches:

1. Implement a reasoner directly based on F-Logic. That means we have build a reasoner form scratch. The difficulty here lies not in implementing a system itself, but building a system that performs it’s proof-search very efficiently!

   The calculus given in [KLW95] has about 14 different inference rules, among them there are also some rather complex ones. Thus, it’s likely that a naive implementation of this calculus would be horrible inefficient. We expect that a lot of effort would have to be invested in order to come up with a implementation with high performance.

2. Translate a F-Logic into First-Order Logic and use an existing ATP. On the other hand, there are a couple of rather sophisticated ATP systems

\[\text{Please note that this is different from having a language for which logical entailment is always decidable! Indeed, this would be even more restrictive.}\]

\[\text{For classical logic such an example would be SLD-Resolution which exploits syntactical characteristics of the the Horn fragment of First-Order Logic in order to get rid of non-determinism when searching for a proof. This method is sound and complete for the Horn fragment, but not complete for arbitrary formulas in CNF.}\]
for classical First-Order Logic (with and without Equality) like Vampire [RV], E-Setheo [MIL+97], E [Sch01], Gandalf [Tam97], DCTP [Ste02], Otter [MW97], Scott [Sla93] etc. In some cases these systems have been developed and tuned over many years.

Furthermore, F-Logic formulas can be translated into FOL formulas. The basic principle underlying the translation can be found in [FDES98], but a formal correctness argument is missing. It should not be a difficult task to define the correctness criterion and formally prove that our translation is correct.

Then we could easily create a reasoning engine that can deal with arbitrary F-Logic formulas by translating them into corresponding FOL formulas (and additional axioms for defining the semantics) and using one of the existing FOL inference engines to actually perform the proof-search.

The main effort here lies in the implementation of the translation from F-Logic into First-Order Logic (including the correctness proof), and the translation of the generated formulas into the proprietary format used by the specific prover.

Both approaches have advantages as well as drawbacks:

1. **Approach 1.**
   - ⊕ Full control over source code and development.
   - ⊕ F-Logic specific enhancements and tuning possible.
   - ⊕ First available implementation of a reasoner for full F-Logic.
   - ⊕ Application-specific extensions extensions possible, e.g. handling of concrete domains like Strings, Integers.
   - ⊕ Most likely a lot of effort and time is needed in order to come up with an efficient system.
   - ⊕ We need to have an expert in the field to build an efficient system.
   - ⊕ Pure reasoning approaches for dealing with Integers are not a good idea in general.

2. **Approach 2.**
   - ⊕ We can build a reasoner for our task rather quickly.
   - ⊕ Most likely we’ll get a reasoner that is several orders-of-magnitude faster than a self-made one for quite some time.
   - ⊕ The most complicated task (proof-search) is designed and maintained by experienced experts in the field.
   - ⊕ First available implementation of a reasoner for full F-Logic.
   - ⊖ No control over source code and development.
   - ⊖ Modification of existing systems hard to do (no documentation, very tricky techniques etc.)
   - ⊖ F-Logic specific enhancements and tuning not easy to add.
   - ⊖ Application-specific extensions extensions hardly possible, e.g. specific handling of concrete domains like Strings, Integers.
   - ⊖ Semantic characteristics of F-Logic that could be applied in the calculus are not explicit anymore after the translation.
   - ⊖ Efficient integration of concrete domains not supported: Pure reasoning approaches for dealing with Integers are not a good idea in general.
How to proceed? At the moment we want to proceed as follows: We prefer and implement approach 2 and evaluate the result. This is likely to be achieved in a rather short period of time. The evaluation will show, whether a the full language approach is reasonable at all and whether the system is fast enough. If a restriction of the language not possible or not needed, but we need severe optimizations for the existing system, we should also start to develop a separate direct implementation of F-Logic in parallel and see, whether this can be turned into a feasible solution. If this also seems to be infeasible within a reasonable period of time than we should strive for restricting the language as far as possible in order to get the proof searches more restricted and the inference engines far more efficient.
4 Conclusions

4.1 What have we achieved so far?

Currently we mainly addressed the problem of service discovery in the context of WSMO, that is, given a description of what the requester wants to achieve (Goal) as well as a (possibly huge) set of descriptions of services that precisely state what the corresponding services provide as their functionality (Capability).

For service discovery, we focussed on the most fundamental problem of Goal-Capability-Matching. We described the basic architecture of the corresponding reasoner and discussed benefits as well as drawbacks from taking this approach.

The implementation of the used reasoner is still ongoing and will be finished in the near future. Afterwards, we can perform first case studies and study the efficiency and feasibility of this approach at least in a laboratory setting.

4.2 Future Work

Obviously, the next step is the completion of the implementation of the Goal-Capability-Matching reasoner and the evaluation of this tool.

An interesting question then is to investigate how and to which extend we can apply logic programming systems for the same task. A comparison of both alternative approaches to this specific reasoning task will then be performed. At the moment, we are not sure about the applicability of logic programming techniques in the context of service discovery as described in deliverable D5.1. Even, if these techniques are not applicable in general, a combination of this approach with a full reasoner might be relevant and very useful for practical setting.

Naturally, service discovery is one of the most fundamental of these domains. Additionally, the corresponding description means in WSMO are fully defined and in a sense well-understood. For this reasons, we started with the investigation of this domain.

Clearly, we will look all other domains as well – but a necessary prerequisite for this is some mature intermediate state of the corresponding deliverables in WSMO. In particular, we aim at automated (or at least computer-supported) analysis of global and local properties of a software-system, which is composed by semantic web services.

But even for service discovery, there are some aspects besides the consideration of the pure functionality (Goal-Capability-Matching) that we haven’t treated in detail until now. In particular, we want to look at the „dynamic part” of web service descriptions in WSMO, that is the choreography, and see how we can include this information in a service discovery request which constraints the choreography of accepted services. This issue will mainly be relevant on the application level of WSMO, namely WSMO-Full.

7For instance, the description of orchestration and choreography of web services are currently not as well-developed!
Similarly, the corresponding techniques are also relevant for checking whether two given web services which are intended to interact with each other – for instance by dynamically composing them within a service-based system – can actually interact with each other (and thus can indeed be composed) which is a fundamental task for automated and semi-automated service composition.

To sum up, until now we covered only a tiny aspect of the whole picture. Certainly, a lot of interesting questions and difficult problems will arise along our way when we try to complete our investigation of the various areas where semantic annotation of web services can be exploited beneficially and we’re excited to see how (and to what extent) we can cope with them (in a practical setting).
5 Acknowledgements

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

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

References


