



WSML Deliverable
D5.1 v0.1
INFERENCE SUPPORT FOR
SEMANTIC WEB SERVICES:
PROOF OBLIGATIONS

WSML Working Draft – April 5, 2004

Authors:

Uwe Keller, Rubén Lara, Axel Polleres, Holger Lausen

Editors:

Uwe Keller

This version:

<http://www.wsmo.org/2004/d5/d5.1/v0.1/20040405>

Latest version:

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

Previous version:

<http://www.wsmo.org/2004/d5/d5.1/v0.1/20040301>



Abstract

The *Web Service Modeling Ontology* (WSMO) provides the conceptual framework for semantically describing web services and their specific properties. Based on WSMO the *Web Service Modeling Language* (WSML) implements this conceptual framework in a formal language for annotating web services with semantic information.

In this document we present and discuss how semantic descriptions of web services based on WSMO and WSML can be exploited for various tasks in the design and dynamical composition of web-service-based software-systems.

In particular, we investigate and analyze what kind of statements need to be formally checked (resp. proven) in order to support the various tasks occurring during this dynamic system-construction process mechanically and allow these processes to access the semantic information captured in the description. These formal statements are called *proof obligations*.

Together, WSMO and WSML as well as the semantic tool suite described and developed within deliverable D5 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.



Contents

1	Introduction	4
1.1	Motivation	4
1.2	Goal of this document	5
1.3	Overview of the paper	5
2	Semantic support in the context of WSMO and WSML	7
2.1	Motivation	7
2.2	Where can semantic annotation be exploited?	7
3	What do we want to prove?	9
3.1	What is a proof obligation?	9
3.2	Service Discovery	9
3.2.1	Goal-Capability-Matching	10
3.2.2	Exploiting Mediators for Web Service Discovery	15
4	Concrete Proof Obligations	17
4.1	Proof obligations in the context of Service Discovery	17
4.1.1	Capability matches Goal	17
5	Conclusions	23
5.1	What have we achieved so far?	23
5.2	Future Work	23
6	Acknowledgements	24



1 Introduction

1.1 Motivation

The Web as a huge information repository A huge amount of knowledge of mankind is accessible in the Web.

From Web Services to Semantic Web Services. - Web Services - Software architectures based on web services - Semantic Web Services

Describing Semantic Web Services. - WSMF

- WSMO/WSML the (*Web Service Modeling Ontology (WSMO)*) [KLR04]
- WSMO Semantics

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 precise¹ 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 foreseen 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.

¹Usually even unambiguous in a mathematical sense!



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 semantic 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 this document we present and discuss how semantic descriptions of web services based on WSMO and WSML can be exploited for various tasks in the design and dynamic composition of web-service-based software-systems.

We identify the different phases (resp. domains) within the (dynamic) construction of a software system based on web services, where semantic information can be beneficially used to support a client in solving a problem at hand with a computerized agent. For the various domains, we describe what kind of reasoning task we have to cope with and what form of reasoning has to be carried out.

In particular, for all these domains we investigate and analyze what kind of statements need to be formally checked (resp. proven) in order to support the various tasks occurring during this dynamic system-construction process mechanically and allow these processes to access the semantic information captured in the description. These formal statements are called *proof obligations*.

1.3 Overview of the paper

Section 2 identifies 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.

In section 4 we define and describe the different reasoning tasks that have to be supported by a suite of inferencing tools in the context of WSMO and

²One imagines for instance the potential benefit of dynamic configuration of supply chains for manufacturing companies under consideration of the current market state.



WSML. In each case we extract the corresponding statements that have to be proven formally and give a precise definition.

Finally, in section 5 we summarize our current achievements and sketch future work and ideas.



2 Semantic support in the context of WSMO and WSML

This section identifies 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.

2.1 Motivation

Having the overall vision of the Web Service Modeling Framework [FB02] in mind, let's briefly look at a possible scenario of using semantic web services in order to solve a problem.

An example scenario.

Our scenario reveals the major application areas of semantic description of web services and shows what elements of these semantic descriptions can be exploited in what manner. Precisely here is where reasoning comes into play: Each time when we want to access and exploit the information contained in the semantic annotation of web services, we'll most likely need to perform some form of reasoning.

Therefore, by abstracting from the specific details of the given scenario and analyzing the resulting generic model we are able to identify the major proof obligations that we have deal with.

An abstract view on this scenario. *Roughly: Description of needs by goal, as some agent to look up services that can potentially provide a solution (capabilities). Find out whether the agent can communicate with and actually invoke the service. Make sure that nothing bad happen during the execution.*

2.2 Where can semantic annotation be exploited?

According to this abstraction, we have to consider the following areas where elements of semantic descriptions of web services can be exploited. In each of these areas, we'll then identify possible concrete reasoning tasks.

- Service Discovery
- Service Composition
- Service Invocation
- Service Mediation on the data, protocol and process level
- Service Execution Monitoring
- Service Compensation

Remark (Completeness of the list). Please note, that we don't consider this list to be complete in any sense. During the course of this project, we are going to extend and possibly refine it as far as this is relevant for the use cases that we aim to address.



Furthermore, in future versions of this document we will only consider those areas wherein we are able to identify substantial or important concrete reasoning task. ◉

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.



3 What do we want to prove?

Now we want to take a closer look into each of the application areas that have been revealed by our abstract model and for which semantic annotation has to be processed and exploited. For each of these domains, we want to clarify how we can actually apply the knowledge contained in the description of semantic web services within this context. Consequently, we derive the corresponding proof obligations in an informal or semi-formal way. A rigorous definition of these proof obligations can be found in section 4.

3.1 What is a proof obligation?

to be written !

Semantics of WSMO/WSML vs. Proof Obligation. *Roughly: Point out the difference between the formal language used to capture the precise semantics of WSMO/WSML (see Language Comparison) and the Language used for formally establishing proof obligations.*

3.2 Service Discovery

According to the above mentioned scenario, we have in principle two parties between which a matching has to be established: A requester who is looking for a concrete service that solves a specific problem and a set of services that offer specific functionality to their clients. Both parties are basically interested in interacting with each other.

Then, *Service Discovery* is the process of identifying possible candidates of services that can solve the requester's problems. The identification of valid candidate services will at least take into account the the functionality that is offered by a service. Furthermore, an additional selection function might be applied in a filtering sub-process. This selection function for instance might be based on non-functional characteristics of services (like response times, level of trust or reliability) and user preferences.

Remark (Service Discovery vs. Requester-Provider-Matchmaking). Please note, that when we are considering a business environment the service discovery process is only one single element of the overall matchmaking process between service requester and service provider, since whether a service requester can – or is willing to – actually use one of the discovered services in order to solve his problem depends also additional information besides the functionality offered by a service which can't be advertised in general beforehand but instead depends on the specific role of the requester within the interaction and the current state of the world when the service requester is looking for a service, e.g. financial aspects or a certain degree of trust between the two interacting parties. In particular, in a business setting this involves *negotiation* between the service requester and the service providers.

Requester-Provider-Matchmaking is an important area that we have to address when we consider WSMO-Full. ☺

Thus, service discovery is basically based on two ingredients: A precise description of the needs of a client that requests some abstract service (whereby



this service must not necessarily exist in a materialized form in the real-world) as well as precise descriptions of what the available and materialized services can provide.

The most fundamental aspect of service discovery then is the exclusive consideration of the *principle functionality* that is expected by requester and provided by a service (thus ignoring all non-functional characteristics). We call this subtask *Goal-Capability-Matching* and explore it in detail in the following section.

3.2.1 Goal-Capability-Matching

Goals. In WSMO-Standard [KLR04] the main means for a service requester to express his desires are so-called *goals*. There, the definition of the term *goal* basically coincides with the one given in [FB02]:

A goal specifies the objectives that a client may have when he consults a web service.

By invoking a service, the requester wants to achieve something: either he wants to acquire additional knowledge (by analyzing the result resp. output of the invoked service) or the state of the real-world should be changed in some way. Indeed, both reasons can be considered as some kind of state change: The former can be considered as a state change in the information space whereas the latter can be considered as a state change in the real-world.

In contrast to [FB02], where a goal formally is defined by two logical expressions describing requirements on the pre-state *and* the post-state, in WSMO a goal only refers to the state that should be reached after invoking a service. In that state, the desires of the requester have to be satisfied.

To sum up, we consider a goal here as a description of possible states of the world (as well as the states of the information space) that a client wants to have reached after invoking a service. Of course, the basic assumption here is that the requester wants to use a service in order to reach his goal.

Therefore in WSMO a goal G is represented by a *postcondition* (for the state of the information space) and an *effect* (for the state of the real-world). Formally, postconditions and effects are represented by formulas in F-Logic [KLW95] (over some signature Σ): Φ^{post} and Φ^{eff} .

In other words, by using some service the user wants to reach a state (in the real-world) where both the specified postcondition as well as the specified effects are satisfied. All such states are considered as valid with respect to the requesters goal, that is the goal is considered as being achieved, when the service execution results in such a particular state; the goal itself can be identified with the set of states that satisfy the requirements stated by postcondition and effects.

The basic vocabulary for the definition of these constraints on the state that has to be reached after the service execution originates from a set of domain ontologies $\mathcal{O} = \{O_1, \dots, O_n\}$, e.g. $\Sigma_{\mathcal{O}} \subseteq \Sigma$. Since each ontology O_i formally describes the interrelationship between the terms used for expressing the goal, they also have to be considered in this reasoning task. Without loss of generality, we consider an ontology O_i as a set of F-Logic formulas (over signature $\Sigma_{\mathcal{O}}$).



In general, the ontologies that are used for describing the goal can't be integrated by simply joining them into one ontology: Usually, there will be conflicts, e.g. due to slight differences in the interpretation of terms, or some concepts that have actually a similar semantics in two or more ontologies are considered to be different because of different names. These problems with integration the imported ontologies are treated by one or more *ontology mediators* M . We denote the set of used ontology mediators by \mathcal{M} and assume again that the mediation functionality provides by these mediator M is represented by a set of formulas.

A goal then can be represented as a tuple of (sets of) F-Logic formulas (over some signature Σ):

$$G = (\Phi^{post}, \Phi^{eff}, \mathcal{O}, \mathcal{M})$$

Capabilities. The main concept in WSMO-Standard for describing what a service actually provides as it's functionality is the so-called *capability*. A capability basically consists of description what the service expects from it's input (*pre-condition*) and the state of the world (*assumption*) at the invocation in order to provide it's service properly, as well as the description of the output (*postcondition*) it provides and the state of the real-world that is going to be reached afterwards (*effects*).

Formally, each of these descriptions represents a constraint of a state (of the information space or the real-world) and is thus expressed by a F-Logic formula: $\Psi^{pre}, \Psi^{ass}, \Psi^{post}, \Psi^{eff}$. As in the case of goals, for any capability C these formulas are expressed with respect to a set of ontologies \mathcal{O}_C that have to be integrated by using some ontology mediators \mathcal{M}_C . Again, we consider both elements to be represented by a set (resp. sets) of F-Logic formulas.

Hence, a capability can be considered as a tuple of (sets of) F-Logic formulas too:

$$C = (\Psi^{pre}, \Psi^{ass}, \Psi^{post}, \Psi^{eff}, \mathcal{O}_C, \mathcal{M}_C)$$

Remark (Capabilities and actual Service Functionality). Let us briefly discuss the relationship between the terms „capability” and „service”.

In WSMO each semantic web service specifies its capability. This means that the service provider announces that this web service provides a certain type of functionality. The actual functionality provided by the the web service indeed is defined by the implementation of the service.

Semantically speaking, a service implementation defines a specific relation between our abstract state-space³.

On the other hand, from a semantic point of view, a capability can also be seen as a characterization of a set of services with similar behaviour; it constrains the „allowed” state-changes when executing these services. In this sense capabilities and actual service functionalities are principally independent of each other.

More formally, we can give a formula in a Dynamic F-Logic⁴ which describes whether a service „respects” some capability: Given a concrete implemented

³The set of all states. Each state in this space represents knowledge about the real-world as well as the information space.

⁴This logic can be seen as a logic that combines two orthogonal aspects in a single logical framework: the modelling primitives of F-Logic [KLW95] for states and a propositional Dynamic Logic [HKT00] for state-transitions. The details are not relevant here, since throughout this section, we use this formal language as a notation with precise semantics.



service s , that means a fixed relation $I(s)$ over the state space. A service s provides a capability C if the following formula (in a Dynamic F-Logic) is valid (holds in all states wrt. this fixed interpretation of s):

$$(\Psi_C^{pre} \wedge \Psi_C^{ass}) \rightarrow (\langle s \rangle (\Psi_C^{post} \wedge \Psi_C^{eff}) \wedge [s] (\Psi_C^{post} \wedge \Psi_C^{eff}))$$

The formula can be read as follows: If the precondition and the assumption of the capabilities are met then every terminating execution⁵ of service s stops in a state that satisfies the postcondition and the effects specified by the capability C (and such an execution exists).

Please note, that in WSMF and WSMO capability descriptions are separate from a *specific* services. A capability can characterize more than one service. Thus, by looking at a capability, we don't know anything about a concrete service providing this capability – we even don't care about that!

Every service instead has to specify its capability and at least there is exactly one such capability specification for each available service. \odot

What does matching now precisely mean? We consider the *functionality* of service s – expressed by means of its capability C – to *match* the needs of the requester – expressed in terms of a goal G – if the service – according to C – can be executed in a way such that the poststate that is reached afterwards satisfies the requirements of the requester.

SERVICE MATCHES
GOAL

At the level of capabilities (instead of services) this results in the following definition: For every (*possible*) service s that offers C as its capability it holds that the service s can be executed such that in the poststate of this execution the goal of the requester is met.

CAPABILITY
MATCHES GOAL

In the model of WSMO-Standard, by using a goal the requester only specifies what he expects to achieve and not what he's willing or able to provide, whereas whether „there is an execution of the service” by which the needed poststate is reached depends on the state of the world and the input when the service is invoked. In that sense, our notion of matching between a goal and a capability is a *relative one*: The match can only be considered as *valid* (or meaningful) when the *requester* is able ensure the environment described by the preconditions and the assumptions in the capability. But as soon as this is guaranteed, the service can provide its functionality and thus there is such an execution. In that sense, the matching – though being relative to the prestate requirements – is actually established by the poststate requirements only.

WHAT ABOUT THE
PRECONDITIONS?
– MATCHING AS A
RELATIVE NOTION

Remark (Preconditions & assumptions in the context of matching). As mentioned, the introduced notion of matching is a relative one – it's relative to the ability of the requester to ensure the preconditions and the assumptions. The match is valid only if user ensures that the precondition and assumptions are met when actually invoking the service. What consequences has this fact? Indeed, the given explanation is a bit too sloppy: How is the user able to ensure these conditions at all?

With preconditions it's straightforward: Preconditions are defined as conditions that the input values of a user have to respect at invocation time in order to allow the service to work properly. Such a condition could be the requirement that a particular integer-valued input parameter that represents the day during

⁵In general, we consider s to be nondeterministic.



a month of a specific date only takes values between 1 and 31. Since these conditions are explicitly mentioned and they only refer directly to the user input, the user can indeed easily ensure that the requirements are met by his input.

With assumptions the situation in general is more involved: Assumptions are constraints on the state of the world that the service imposes. Only when the assumptions are satisfied at invocation time, the service guarantees to work properly. Since assumptions refer to the „current” state of the world, the user only has limited (or no) possibilities to influence the properties of the state of the world. In most cases, the user is not even interested in actively changing the state of the world such that the assumptions are met: Imagine for example the assumption of a book-selling service that enough copies of the desired book are available in the stock; here the user himself can not change the state of the stock. Even worse, he might not be able to get any information about the state of the book-sellers stock. If the user is really lucky, then he is indeed able (perhaps by using some additional services) to check whether the assumptions are fulfilled at invocation time. In general, this will not be the case.

But what consequence has this issue: According to our definition of matching between goal and capability, a service that provides the given capability would match the goal although (when it comes to the actual invocation) the service indeed can *not* be used to fulfill my goal, *because the assumptions are not satisfied!* And this discussion also shows, that if it's not possible to automatically ensure the assumptions (by checking them) at invocation time, it's not possible to automatically discover and invoke a service which offers this capability without having doubts that the service can't be used for the intended purpose. That means fully-automated and dynamic system-construction at runtime is not possible in a reliable way under these circumstances. But it still is possible, if all assumption can be checked at invocation time⁶

Naturally, we have to ask ourselves whether our definition for goal-capability-matching is indeed adequate for our purpose:

The discussion above clearly shows that the property of our criterion as being „only” relative to the precondition (and not explicitly taking into account the precondition for the decision on matching) *is not a flaw* of our definition but somehow inherent to all such formal criterions:

Discovery is a process that chronologically occurs before the corresponding service invocation. It's never possible to have both at the same time. Thus, during the period of time between both activities the world might change and the results of the discovery process might become invalid (wrt. to the new state of the world – that is, the current one at invocation time).

Moreover, during service discovery usually not all information that (of course) is available when the user is actually invoking a service (actual state of the world at invocation time as well as all input values) are known at discovery time. For instance, one might think of an additional input parameter of a candidate service for which its concrete value at invocation time can not be determined by just using the goal specification given by the requester.

Thus, by taking preconditions and assumptions into account at discovery time, we basically have the same problem, since preconditions and assumptions have to be evaluated with respect to the concrete state of the world (and information space resp. input values) at the actual invocation time but this point in time is different from the point in time when discovery happens.

Nevertheless, checking the preconditions and assumptions at invocation time is a very important if we want to construct reliable and correct software-systems *automatically* on runtime by composing web-services. ◉

⁶Of course, all the problems with a changing world with agents that act concurrently that are well-known from information systems occur in this context as well!



Remark (Formal description of a goal-driven discovery request). Above, we have not given a formal semantics of the client’s discovery request. We want to do this here: Given a goal G by the user, the corresponding (goal-driven) discovery request R_G is:

Find some service s (which has been described by means of a capacity C) for which the following formula is valid:

$$(\Psi_C^{pre} \wedge \Psi_C^{ass}) \rightarrow (\langle \mathbf{s} \rangle (\Psi_G^{post} \wedge \Psi_G^{eff}) \wedge [\mathbf{s}] (\Phi_G^{post} \wedge \Phi_G^{eff}))$$

All known semantic web services s_i (in a registry) are described by means of their capabilities C_i which semantically can be seen as the following valid formula:

$$(\Psi_{C_i}^{pre} \wedge \Psi_{C_i}^{ass}) \rightarrow (\langle \mathbf{s}_i \rangle (\Psi_{C_i}^{post} \wedge \Psi_{C_i}^{eff}) \wedge [\mathbf{s}_i] (\Phi_{C_i}^{post} \wedge \Phi_{C_i}^{eff}))$$

Please note, that this representation of the goal-capability-matching obviously suggest a natural criterion on the specification elements (goal and capabilities). We will discuss later on whether this criterion is the only one, and if not in which relation this criterion stands to an alternative one (e.g. if it’s stronger or weaker etc.) \odot

In the simplest case, we assume that there is a *single* service that fits our needs. Then, we are looking for *exact* matches between the description of the requested service (the goal) and the specification of what is offered by existing and known web services (the capabilities). Thus, we have to support *goal-capability-matching*.

However, in many cases, we will not be able to find a *single* service that *exactly* provides what we need. In order to resolve this deficiency, in principle there are two natural approaches (which themselves might not be considered directly as service discovery tasks):

- *Server-sided resolution: service composition* – Given the specification of the requesters needs, we try to combine several existing web services at design-time (represented by means of proxies) or on the fly at runtime in order to create a new web service that is adequate with respect to the request.
- *Client-sided resolution: compute and describe mismatch* – Given the specification of the requesters needs and the description of a possible (e.g. almost matching) web service, we compute the existing mismatch and return it to the client. Afterwards it’s the client’s responsibility to use this information in a proper way to close the gap.

Clearly, the technique underlying the latter approach can also be used order to support the first approach: When we precisely know about the mismatch between what we need and what we have, it’s much easier to find the missing piece in the puzzle.

Therefore, we can consider the *computation of a goal-capability-mismatch* as a quite useful and relevant task that we would love to be supported – at least to some extent – by our framework. In principle, expect this problem to be an undecidable problem in most cases, but there might be interesting cases too, that be dealt with algorithmically. There might even be cases, when the description can be used in a constructive sense: for the *automated generation of a mediator* that bridges the gap.



In fact, this problem is a generalization of the the capability-goal-matching problem: If we can compute the mismatch between a goal and a capability then we can also solve the matching problem by testing whether there is *no* mismatch at all. Thus, it's clearly computationally harder.

Remark (Relation to WSMO-Layering). The computational complexity of these questions heavily depends on the notion of *service request* that one applies (or the language that can be used in order to specify such a *request* respectively). For instance, in WSMO-Full – which addresses the business layer of underlying abstract web service architecture model – the specification of a request most likely will represent a high-level and rather rough process specification, which is far more complex than a simple description of the requirements of a state of the world. Therefore, the complexity and tractability of the reasoning tasks will vary in general with respect to the single layers of WSMO. ©

3.2.2 Exploiting Mediators for Web Service Discovery

In the previous section, we have only taken into account the description of the capabilities of available services and the description of the requester's goal. Only these description elements are exploited in order to locate services that fulfill the requirements expressed by the requester.

In this section, we discuss how to exploit the mediators defined in WSMO to provide an alternative approach to Web Service discovery. The mediators that are to be considered in this approach are ggMediators and wgMediators. We will first discuss the use of wgMediators and then we will analyze the role of ggMediators in the discovery process.

Use of wgMediators WSMO-Standard version 0.2 defines wgMediators as elements that link Web Services to goals, explicitly stating the difference (called reduction) between the two components and mapping different vocabularies. The most interesting point of this definition is that wgMediators provide an explicit link between Web Service capabilities and requester goals. The mediation between different vocabularies is not of interest for our discussion.

The approach presented in the previous section does not consider these explicit links between requester needs and Web Service offers. Clearly, these explicit links can be taken into account when locating suitable Web Services, as they describe a well-defined relationship between the request and the offer. In order to exploit these links, we will make two assumptions:

- **Use of pre-defined goals:** Explicit links can only be exploited if they are provided at design-time i.e. if they are available for the discovery process. In order to define such links at design-time, the goals and capabilities involved also have to be defined before-hand. In the case of capabilities it is clear that the service will describe its capability to be advertised. However, it can be argued that goals can be defined by the user right before submitting the query for suitable services, and therefore explicit links would not be available. Therefore, our assumption is that some pre-defined goals are described at design-time and links are established between them and the capabilities of the published services.
- **Definition of reductions by the requester:** The pre-defined goals will be customized (if necessary) by the requester to describe her needs. This can be done by using ggMediators i.e. the user expresses its request by refining existing goals, and this refinement is expressed as the reduction



in a ggMediator. Therefore, we assume that the requester will use pre-defined goals (with well-established links to the available services) and refine them using ggMediators.

We believe that these assumptions can be met in a real application. In fact, defining libraries of pre-defined goals will support the user in defining her needs for the following reasons:

- **Ease-of-use:** The requester can inspect available goals and, helped by non-functional properties of the goal such as the textual description, will locate the goals related to her request. This process is, specially for non-experts in logics, much easier than defining postconditions and effects from the scratch.
- **Reduction of effort:** The requester only has to specify postconditions and effects to refine existing goals. This effort is obviously lower than defining a complete new goal.
- **Reduction of errors:** The use of pre-defined and appropriately tested goals improves accuracy when defining the requester needs. Goals define from the scratch by the requester are more likely to be wrongly expressed.

For these reasons, we believe that the use of pre-defined goals is not only a reasonable assumption but also a feature that might be required by users.

Once pre-defined goals, wgMediators, and the reductions to the selected pre-defined goal are performed by the user, we can use this information for locating suitable services. In contrast to the scenario in section 3.2.1, now the search space is not the entire set of available services, but only those ones linked via a wgMediator to the pre-defined goal selected (and possibly refined) by the requester. In this way, we can considerably cut-off the search space in the discovery process.

Use of ggMediators So far, we have ignored ggMediators, that enable the definition of goals by reusing and refining existing ones. Nevertheless, this is an important point that deserves additional discussion in order to ground our alternative approach. As a goal can be described as successive refinements of other goals, it can happen that services linked to the goal(s) used to describe the selected goal can fulfill the user request. For our discussion, we will assume that the ggMediators define a non-directed acyclic graph, being the nodes of the graph the different goals defined, and the arcs the reductions captured in the ggMediators. We do not consider cycles as these would be an error when modelling goals. We consider the graph as non-directed, as the reduction expressed in the ggMediator is direction-independent.

WSMO-Standard version 0.2 [KLR04] defines the links established by wgMediators and ggMediators as capturing the difference between the linked components. However, this difference can reflect a reduction of functionality or an augmentation of functionality. The existence of both possibilities implies that the discovery process has to follow every path linked to the selected goal.

If we impose that the reductions expressed in ggMediators and wgMediators always capture a functionality reduction, then we have a tree instead of a graph. In this case, only the services linked to the selected goal and its parent goals (more general functionalities) have to be considered. Nevertheless, this does not completely fit to the definition of reduction given in WSMO-Standard version 0.2. Therefore, we will not make this assumption in order to be consistent with the definitions given in the specification of WSMO.

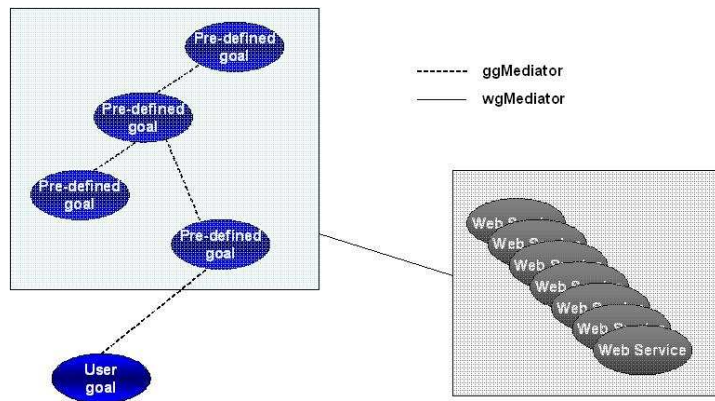


Figure 1: Search space in the presence of mediators

In this context, the search space is the one depicted in Figure 1. The box on the left represents the goals to be taken into account i.e. the ones linked to the selected goal via a `ggMediator`. The box on the right represents all the services linked to these goals via a `wgMediator`, that constitute the search space.

When comparing the use of mediators for Web Service discovery to the discover process without mediators, positive and negative aspects can be found for the former approach:

- ⊕ Reduced search space
- ⊕ Supports the user in defining her goals
- ⊖ The approach is less general

But, both approaches are compatible i.e. mediators can be exploited if a pre-defined goal is selected and refined, and the discovery in absence of mediators can be applied otherwise.

4 Concrete Proof Obligations

In section 3 we described the concrete reasoning tasks that we want to support on a precise but still semi-formal level. Now, this section is concerned with the precise formal definition of the corresponding proof obligations.

4.1 Proof obligations in the context of Service Discovery

In the following sections we give precise definitions of the proof obligations that belong to the single reasoning tasks identified for web service discovery in section 3.2.

4.1.1 Capability matches Goal

Let G be a goal and C be a capability. According to our definition above, C matches G if and only if for any service s that offers capability C there is an

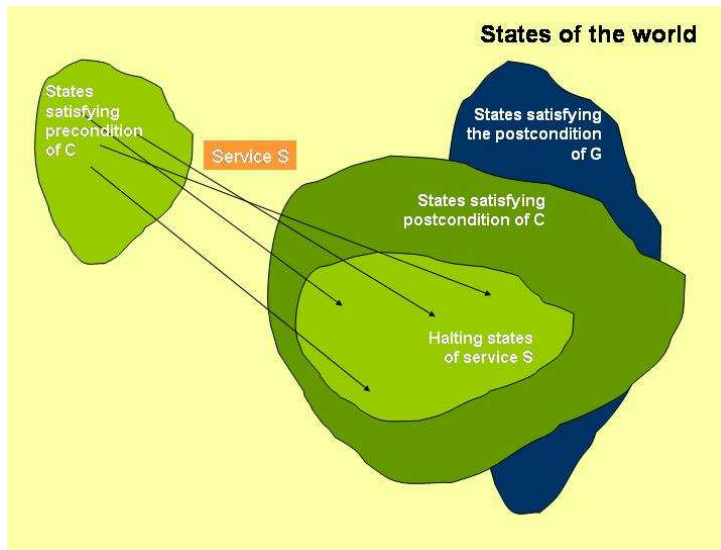


Figure 2: Semantical relationship between capability, goals and services

execution of the service⁷, which stops in a state that satisfies the requirements stated in the goal. According to semantics of a capability, the resulting state – the post-state of the execution of service s – satisfies at least the requirements stated as the postconditions and the effects of the capability.

Thus, given a specific service s that provides a capability C , it's obvious that the set of states which are reachable by executing s from any state satisfying the precondition of C is indeed a subset of the set of *all* states that satisfy the postcondition of C . This situation is illustrated in figure 2.

If for the given service s the set of halting states has a common element with the set of states where the goal is satisfied, then the service can be used by the requester in order to achieve his goal.

Since in the definition of matching between capabilities and goals we universally quantify over all possible services which providing a capability C , we have to consider especially the two extreme cases: a service s that satisfies capability C and for which the set of possible halting states is minimal⁸ (wrt. set-inclusion) or maximal. Regarding the matching, the first case is the worst possible case, whereas the second is the best – in the optimal case the set of halting states and the set of states satisfying the postcondition coincide.

But at the capability level, we don't have any information about specific services s that offer capability C , hence, we don't know anything about the possible sets of halting states of the services s . The only information related to the set of halting states that can be read off the capability description itself is the postcondition. Thus, this is the only available information for developing the proof obligation for the goal-capability-matching task.

⁷Please note, that the actual execution – and thus the state reached by this specific execution – depends on the concrete state of the world (in particular the input) where the service is being invoked!

⁸Note that this set is non-empty iff the precondition of the capability is satisfiable.



If we can show, that the set of states satisfying the postcondition of a capability C is a subset of the set of states that fulfill the goal, then this also holds for all services s providing C , and the matching is established.

Clearly, this is a (much) stronger criterion than the actual required one, e.g. there are some cases where a capability actually matches a goal, but this criterion is not satisfied. But it's the *weakest* criterion that can be established with the given information on services by a capability, that means every weaker criterion would need *specific* information about the services s that can provide the capability C .

Thus, we'll use this *stronger* criterion (as needed when we would know about the actual semantics of the services) when defining the proof obligation for goal-capability-matching: Whatever state we consider, when the state satisfies the post-state-conditions of the capability, then it satisfies the goal. Since states are represented by F-Logic-Interpretations I (over signature Σ) we get as our proof obligation nothing else than a logical entailment between the requirements on the poststate by the capabilities and the goal: $\Psi^{post} \wedge \Psi^{eff} \models \Phi^{post} \wedge \Phi^{eff}$.

In summary, we have achieved the following: If C matches G (by proving the proof obligation) and if the requester is able to ensure the requirements on the pre-state when calling any service with capability C , then (according to the semantics of the capability) this service will stop in a state that fulfills the postcondition and thus (because of our stronger criterion) the requirements specified in the goals are satisfied. In other words: The requester can use a service with capability C in order to achieve his goal!

Again, we have to take imported ontologies \mathcal{O} and an ontology mediator M also into account: In the proof obligation, they have to be considered as premisses, since the axioms of the ontologies as well as the description of the mediation constrain (or define) the basic semantics of the symbols in the ontologies and the definition of the capability and the goal in turn relies on this particular semantics.

Remark (Goal-Goal-Mediators). In WSMO a goal can be defined in terms of another goal and a refinement condition. For this definition a *Goal-Goal-Mediator* M_G^{gg} is used. These mediator M_G^{gg} can be dealt with in the same way as for ontology mediators. \odot

Proof Obligation Given a goal

$$G = (\Phi^{post}, \Phi^{eff}, \mathcal{O}_G, M_G^{oo}, M_G^{gg})$$

be a goal and

$$C = (\Psi^{pre}, \Psi^{ass}, \Psi^{post}, \Psi^{eff}, \mathcal{O}_C, M_C)$$

be a capability. Then the *proof obligation for goal-capability-matching* is to

$$PO_{gcm}^*(G, C) \equiv (\{\mathcal{O}_C, \mathcal{O}_G, M_C, M_G^{oo}, M_G^{gg}, \Psi^{post}, \Psi^{eff}\} \models (\Phi^{post} \wedge \Phi^{eff}))$$

By using sound and complete deduction calculus 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:

$$PO_{gcm}(G, C) \equiv (\{\mathcal{O}_C, \mathcal{O}_G, M_C, M_G^{oo}, M_G^{gg}, \Psi^{post}, \Psi^{eff}\} \vdash (\Phi^{post} \wedge \Phi^{eff}))$$



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 the following:

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 general⁹. 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.

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. 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)¹⁰. 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 techniques¹¹ 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 unfeasibility of one approach or the other has to be analysed 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.

⁹But it might stop for some particular cases.

¹⁰Please note that this is different from having a language for which logical entailment is always decidable! Indeed, this would be even more restrictive.

¹¹For 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.



Alternative Approaches. As mentioned in section 4.1.1, 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 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 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.



5 Conclusions

5.1 What have we achieved so far?

5.2 Future Work

Formal methods in the context of protocol specifications. Model Checking. Additional proof-obligations: Deadlockfreeness, Livelockfreeness. Maybe other formal techniques (different from deduction) are more efficient for this task.

Open questions: What is decidable? What can be done automatically? What requires human interaction?



6 Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperanto, 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

- [AG97a] Robert Allen and David Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, July 1997.
- [AG97b] Robert John Allen and David Garlan. *A formal approach to software architecture*. CMU-CS-97-144, School of Computer Science, Carnegie Mellon University, Pittsburg, USA, 1997.
- [BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley Pub Co, 2nd edition, 2003.
- [DEFS99] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *DS-8*, pages 351–369, 1999.
- [FB02] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.
- [FDES98] Dieter Fensel, Stefan Decker, Michael Erdmann, and Rudi Studer. Ontobroker in a nutshell. In *European Conference on Digital Libraries*, pages 663–664, 1998.
- [Fen03] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer-Verlag, Berlin, 2003.
- [FHK⁺97] Jurgen Frohn, Rainer Himmeroder, Paul-Thomas Kandzia, Georg Lausen, and Christian Schleppehorst. FLORID: A prototype for f-logic. In *ICDE*, page 583, 1997.
- [GS96] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.
- [KLR04] Uwe Keller, Holger Lausen, and Dimitriu Roman (*editors*). Web Service Modeling Ontology – Standard (WSMO-Standard). Working draft, Digital Enterprise Research Institute (DERI), March 2004. Look at <http://www.wsmo.org/2004/d2/v0.2>.
- [KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.



- [MIL⁺97] Max Moser, Ortrun Ibens, Reinhold Letz, Joachim Steinbach, Christoph Goller, Johann Schumann, and Klaus Mayr. SETHEO and e-SETHEO - the CADE-13 systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.
- [MW97] William McCune and Larry Vos. Otter - the CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.
- [RV] Alexandre Riazanov and Andrei Voronkov. System description: Vampire 1.0.
- [Sch01] S. Schulz. System Abstract: E 0.61. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the 1st IJCAR, Siena*, number 2083 in LNAI, pages 370–375. Springer, 2001.
- [Sla93] John K. Slaney. SCOTT: A model-guided theorem prover. In *IJCAI*, pages 109–115, 1993.
- [Ste02] G. Stenz. Dctp 1.2 - system abstract. In U. Egly and C. G. Fermüller, editors, *Proc. of TABLEAUX'02*, volume 2381 of LNAI, pages 335–340. Springer, 2002.
- [Tam97] Tanel Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.
- [YK] Guizhen Yang and Michael Kifer. Flora-2: User's manual.