



# D5.1v01 Inferencing Support for the Semantic Web

Reasoning about Semantic Web Service Descriptions in WSMO and WSML

DERI Working Draft 1 March 2004

**This version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d5/d51/v01/20040301>

**Latest version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d5/d51/v01/20040301>

**Previous version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d5/d51/v01/20040301>

**Editors:**

Uwe Keller

This document is also available in non-normative [PDF](#) version.

Copyright (C) 2004 [DERI](#) (R), All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

---

## Contents

- [1. Introduction](#)
  - [1.1 Motivation](#)
  - [1.2 Goal](#)
- [2. Deduction in the context of WSMO and WSML](#)
  - [2.1 Motivation](#)
  - [2.2 What do we want to prove?](#)
    - [2.2.1 Service Discovery](#)
  - [2.3 Web Service Discovery](#)
    - [2.3.1 Capability matches Goal](#)
- [3. Conclusions](#)
  - [3.1 Future Work](#)
    - [3.1.1 Web Service Invocation](#)
    - [3.1.2 Web Service Mediation](#)

### ■ [3.1.3 Web Service Execution](#)

- [4. Acknowledgements](#)
  - [A. WSMO F-Logic Reasoner](#)
    - [A.1 Overall Architecture](#)
    - [A.2 Translating F-Logic into First-Order Logic](#)
    - [A.3 Implementation Aspects](#)
  - [B. Formalizing the WSMO Ontology in F-Logic](#)
    - [B.1 Representing the Ontology in F-Logic](#)
    - [B.2 Applying F-Logic Programming Environments for Querys](#)
  - [Bibliography](#)
- 

## Abstract

In this document we present and discuss the formal framework and tool suite that has been (resp. is in the process of being) developed along with *Web Service Modeling Ontology* (WSMO) and the *Web Service Modeling Language* (WSML) in order to provide a inferencing support in the area of semantic web services.

In particular, we investigate and analyze what kind of formal statements - so-called *proof obligations* - need to be formally established in the context of semantic description of web services by means of WSMO and the corresponding WSML.

Along our way, we identify and use well-established methods and tools that help to achieve our goals.

Eventually, we describe the underlying principles for verifying proof obligations 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 you have to consider this only as a first draft of the deliverable that will continuously be extended and enhanced.

## 1. Introduction

### 1.1 Motivation

## ***From Web Services to Semantic Web Services.***

*to be written.*

## ***Describing Semantic Web Services.***

*to be written.*

## ***And how do we make use of this semantics?***

The major benefit of any form of specification is that by explicitly documenting how some artifact actually behaves and what kind of competence it provides, 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 a specification, the purpose and use of an artifact becomes precise, unambiguous and even 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. In a world of rapidly changing requirements and incomplete knowledge (such as the Semantic Web with its 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. 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/description) of web services are far from being really effective as long as there is no way for *computerized* agents to *actually process* this kind of information about the services. In this context, processing means to apply the formalized knowledge *intelligently* to solve specific problems, 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 problem. 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 our tool suite 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 (EAI) and Electronic Commerce<sup>1.1</sup> [Fen03].

## **1.2 Goal**

In this document we present and discuss the formal framework and tool suite that has been developed along with WSMO and WSML in order to provide a inferencing

support in the area of semantic web services.

In particular, investigate and analyze what kind of formal statements - so-called *proof obligations* - need to be formally established in the context of formal description of web services by means of WSMO and the corresponding *Web Service Modeling Language (WSML)* and identify well-established methods and tools that help to achieve our goals.

Eventually, we describe the underlying principles for verifying proof obligations automatically and explain the tools and techniques we have applied and developed for that purpose.

## 2. Deduction in the context of WSMO and WSML

This section defines and describes the different reasoning tasks that have to be supported by a suite of inferencing tools in the context of WSMO and WSML. In each case we extract the corresponding statements that have to be proven formally and give a precise definition. Eventually, we describe our approach for formally establishing these statements in a detail and justify our decision.

### 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.

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.

***A simple scenario.***

***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.*

According to this abstraction, we have to consider the following application areas where elements of semantic descriptions of web services can be exploited. In each of

these areas, we'll then identify possible proof obligations.

### ***Where can semantic annotation be exploited?***

- Service Discovery
- Service Invocation
- Service Mediation (Data, Protocol, Process)
- Service Execution
- Additionally: querying the WSMO Ontology
- Additionally: building virtual enterprises

### ***Current restrictions.***

Currently, we only consider WSMO-Standard [KLR04]. This excludes some important aspects that will be included in the extension WSMO-Full. These aspects will be mainly concerned with the business or application layer of WSMO, e.g. most likely concepts like contract, contract templates, negotiations and protocols for the pre-negotiation phase and the post-negotiation phase etc. But at the moment, WSMO-Full is not defined yet, and therefore we can't address this issues here. Moreover, in this version of the document, we focus on the service discovery task, in particular on specific aspect therein: the capability-goal-matching.

## 2.2 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 the following sections.

### ***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.*

### 2.2.1 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.

Please note, that this process is only one single element of the overall matchmaking process, 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, 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.

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.

### **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.

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  $\Sigma$ ):  $\Phi^{post}$  and  $\Phi^{eff}$ . 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  $O$  as a set of F-Logic formulas (over signature  $\Sigma_O$ ).

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 of the imported ontologies are treated by an *ontology mediator*  $M$ . We again assume that the mediation functionality provided by this 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  $\Sigma$ ):

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

### Capabilities.

The main concept in WSMO-Standard for describing what a service actually provides as its functionality is the so-called *capability*. A capability basically consists of a description of what the service expects from its input (*precondition*) and the state of the world (*assumption*) at the invocation in order to provide its 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  $O_C$  that have to be integrated by using some ontology mediator  $M_C$ . Again, we consider both elements to be represented by a set of F-Logic formulas.

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

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

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: If a service  $s$  provides a capability  $C$  then the following formula (in a Dynamic F-Logic) is valid (wrt. this fixed interpretation of  $s$ ):

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

Please note, that in WSMF and WSMO capability descriptions are separate from a *specific* service. 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.

### **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. 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. 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.

**Remark Explicit formal definition of a requestleer(Explicit formal definition of a request).** Above, we have not given a formal semantics of the client's request. Perhaps it would be a more appealing and elegant to include this semantic definition of a client's request in this section too: Given a goal  $G$  by the user, *explicitly* represent a request rigorously as follows: Find some service  $s$  (which has been described by means of a capability  $C$ ) for which the following formula is valid (whereby  $s$  is assumed to be deterministic!):

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

This will be clarified soon. In particular, this would allow us to describe more formally what matching then means and precisely derive our given criterion from that. ◀

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:

- *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 run-time 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 should be supported - at least to some extent - by our framework. In principle, this might 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-Layeringleer(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 represents 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.

## 2.3 Web 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 [2.2.1](#). Furthermore, we discuss our approach to establish these proof obligations.

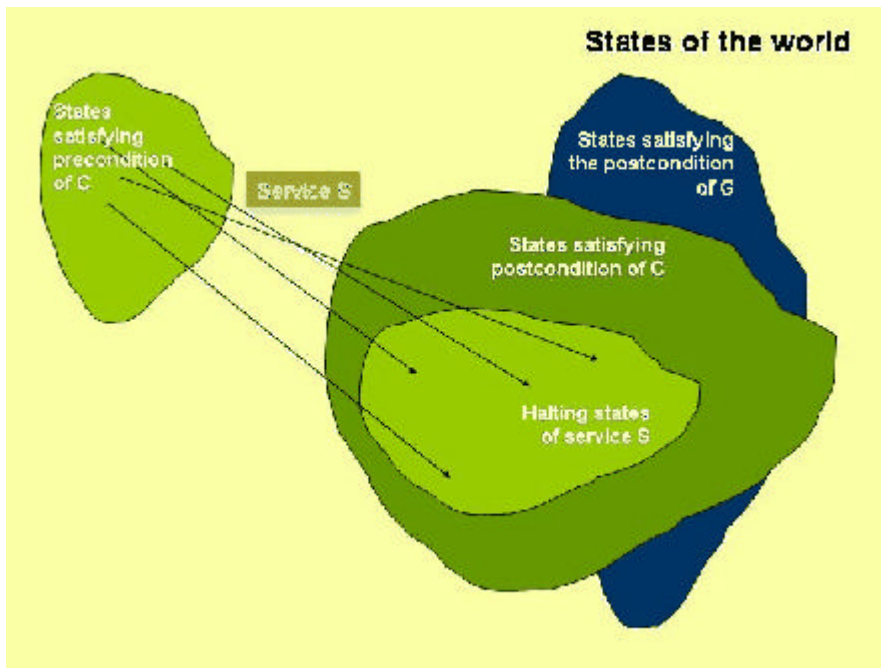
### 2.3.1 Capability matches Goal

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 execution of the [service<sup>2.1</sup>](#), 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 2.1.

**Figure 2.1:** Semantical relationship between capability, goals and services



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 minimal<sup>2.2</sup> (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.

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  $\Sigma$ ) we get as our proof obligation nothing else than a logical entailment between the requirements on the

$$\Psi^{post} \wedge \Psi^{eff} \models \Phi^{post} \wedge \Phi^{eff}$$

poststate by the capabilities and the goal:

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-Mediatorsleer(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*  $M^{gg}_G$  is used. These mediator  $M^{gg}_G$  can be dealt with in the same way as for ontology mediators.

### 2.3.1.1 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 general<sup>2.3</sup>. 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)<sup>2.4</sup>. 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<sup>2.5</sup> 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.

#### **2.3.1.2 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.

### **Why Logic Programming is not adequate in general.**

#### **Alternative Approaches.**

As mentioned in section [2.3.1.1](#), 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<sup>+</sup>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<sup>+</sup>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 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 seperate 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.

## 3. Conclusions

### 3.1 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?

#### 3.1.1 Web Service Invocation

### 3.1.2 Web Service Mediation

### 3.1.3 Web Service Execution

## 4. Acknowledgements

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

## A. WSMO F-Logic Reasoner

### A.1 Overall Architecture

### A.2 Translating F-Logic into First-Order Logic

### A.3 Implementation Aspects

## B. Formalizing the WSMO Ontology in F-Logic

### B.1 Representing the Ontology in F-Logic

### B.2 Applying F-Logic Programming Environments for Querys

## Bibliography

### [DEFS99]

Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer.  
Ontobroker: Ontology based access to distributed and semi-structured  
unformation.

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: A 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 Schlepphorst.  
FLORID: A prototype for f-logic.  
In *ICDE*, page 583, 1997.
- [KLR04]**  
Uwe Keller, Holger Lausen, and Dimitriu Roman (*editors*).  
Web Service Modeling Ontology -- Deliverable 2: WSMO-standard, version 0.1 final.  
Working draft, Digital Enterprise Research Institute (DERI), February 2004.  
Look at <http://www.nextwebgeneration.org/projects/wsmo/>.
- [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 Wos.  
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. Gore, 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. Fermuller, 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.

---

## Footnotes

### ... **Commerce**1.1

One imagines for instance the potential benefit of dynamic configuration of supply chains for manufacturing companies.

### ... **service**2.1

Please 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!

### ... **minimal**2.2

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

### ... **general**2.3

But it might stop for some particular cases.

### ... **language**2.4

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

### ... **techniques**2.5

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.

---