WOOGLE meets Semantic Web Fred

Uwe Keller, Michael Stollberg, Dieter Fensel

Digital Enterprise Research Institute (DERI), University of Innsbruck, Austria
{uwe.keller, michael.stollberg, dieter.fensel}@deri.org

Abstract. A major merit of the Web Service Modeling Ontology WSMO is the well-structured and unambiguous definition of the description elements for its components. This allows developing concise, generic inference mechanisms for basic Semantic Web Service technologies like Web Service Discovery as the detection of suitable Wed Services for solving a Goal. This paper introduces WOOGLE, a basic but powerful and generic WSMO-enabled Web Service discovery mechanism that is developed in the course of the Semantic Web Fred project. We outline the underlying conceptual model for Web Service Discovery in WSMO, discuss the WOOGLE functionality in the context of Semantic Web Fred, and explain its realization on top of a First-order Logic theorem prover.

1 Introduction

The aim of research efforts around Semantic Web Services is to facilitate automated handling of Web Services. Initial Web Service efforts failed to make the promise of automatically interacting, dynamically composed Web Services become reality. The reason is that the Web Service technology stack around SOAP, WSDL, and UDDI does not supply sufficient means for describing Web Service in a way that supports generic mechanisms for discovering, composing, and executing Web Services. Thus, the concept of Semantic Web Services has been established: based on concise and unambiguous semantic description frameworks for Web Services and related aspects, generic inference mechanisms shall be developed for handling Semantic Web Services. Concerning overall frameworks for Semantic Web Services, the most relevant ones existing at this point in time are OWL-S [9] and the Web Service Modelling Ontology WSMO [8]; in comparison, WSMO is defined more precisely and unambiguously, and it provides a more complete framework for aspects and challenges arising within Semantic Web Services [4].

Web Service Discovery is concerned with detecting suitable Web Services for solving a Goal, i.e. an objective that a client wants to get resolved when consulting a Web Service. Discovery is a core technology for Semantic Web Services. After several discussions on the conceptual model and means for realization, Web Service Discovery in WSMO has been separated into three main steps:

1) matching of Goal postconditions and effects with those of Web Service Capabilities in order to detect a set of Web Services that might can solve the Goal without regard to data that is submitted as input to a Web Service

2) detection of those Web Services (out of the possibly usable Web Services in the result of (1)) which can really solve the Goal with regard to the preconditions and assumptions of the Web Service, the data a Goal owner is able to submit as input to

the Service, as well as attending the Choreography Interface, i.e. the behavioral interface of a Web Service for consuming its functionality

3) selection of one Web Service (out of the result of (2)) that is to be really used for solving the Goal under consideration of user preferences and Web Service specific non-functional properties.

The reason for this separation is to decompose the complexity for the overall Web Service Discovery process and to allow combination of different discovery mechanisms within WSMO, thus not restricting the realization of WSMO-enabled technologies to only one solution. WOOGLE is the realization of the 1st step of Web Service Discovery within WSMO that provides some functionality similar to GOOGLE, but on a semantic level: a set Web Service identifiers (URLs) is returned that can possibly solve a given Goal. In fact, only WOOGLE is intended to be a standardized Web Service Discovery technology within WSMO. The subsequent steps are application-dependent, and thus are not part of WSMO which aims at a general framework that is not limited to specific realization technologies not to particular application fields.

The aim of this paper is to explain the framework for Web Service Discovery intended in WSMO and elaborate the conceptual model for WOOGLE along with a prototypical realization. For showcasing integration of WSMO-enabled Semantic Web Service technology into complex systems, we outline the realization of the Service Discovery Module that includes WOOGLE in Semantic Web Fred, an environment for automated agent cooperation with WSMO-enabled dynamic service usage.

The paper is structured as follows: Section 2 explains the conceptual model for Web Service Discovery in WSMO; Section 3 elaborates the theoretical basis of WOOGLE and presents a prototypical realization; Section 4 describes the usage of WOOGLE within SWF and explains the realization of the SWF Service Discovery Module; finally, Section 5 concludes the paper.

2 Web Service Discovery in WSMO

Due to its overall structure and the concise definitions of description elements for the components, WSMO provides the foundation for generic and precisely defined inference mechanisms for Semantic Web Services technologies [8]; in fact, WSMO is designed to support generic, ontology-based inference mechanisms for realizing Semantic Web Service technologies, whereof Web Service Discovery is a core technology.

The conceptual model of Web Service Discovery in WSMO is elaborated in detail in [5]; however, as this document is under construction at the time of writing, the following briefly outlines the approach for Web Service Discovery in WSMO with special attention to the conceptual model of WOOGLE which might serve as a basis for further elaboration of the Web Discovery framework. As stated above, the discovery process in WSMO is separated into three subsequent steps: we explain these along with their position in WSMO, and point out possible solutions.

2.1 Canonical Discovery – WOOGLE

The first step in WSMO Web Service Discovery is to detect (on a semantic basis) a set of Web Services that might be able to solve a given Goal without regard to data that are submitted as input to the Web Service or any other related aspects. The idea is to establish a pre-selection of possibly usable Services to solve a Goal out of a potentially huge set of available Services, thus providing a general facility for Web Service enabled applications to pre-determine the Web Services to deal with. The result of this basic discovery mechanism is a set of WSMO Web Services Identifiers (which are URLs) of available Services that can possibly solve the Goal.

The matchmaking in WOOGLE relies on proving a suitable proof obligation, i.e. a logical formula that is composed by the logical representation of the Goal descriptions as well as the description of the Web Services Capability. Thus, unlike current Internet search engines WOOGLE is a semantic operation which takes into account the semantic descriptions of what services provide and what a client wants to achieve.

Since this first step in the overall discovery process does not take into account concrete input provided by a client on runtime, it is in principle possible to pre-compute matching results for this step if goals and service capabilities are known in advance[1] or at least cache the results of frequently used goals and services. The pre-computed or cached results can then be stored in a database and reused for discovery; hence, the number of actually performed proof attempts (which are comparably expensive operations with regard to simple database queries) can be reduced drastically without losing the characteristics of WOOGLE to be a semantic operation. Moreover, the pre-computed information can be gathered and maintained incrementally as new Service and/or Goal descriptions are stored in a registry.

The WOOGLE mechanism is completely WSMO-compliant as it does not require additional information beyond what is specified in WSMO component descriptions. Furthermore, it is a generic discovery facility independent of specific technologies or application domains, and thus will be provided as a standardized Web Service Discovery mechanism within WSMO.

2.2 Detection of usable Services

The WOOGLE mechanism identifies only Web Service that can possibly solve the given Goal according to the logical proof obligation. The WOOGLE result set might contain Services that can not be used for various reasons, e.g. spatial availability, too unspecific service descriptions, or requested information the client can not or does not want to provide. Thus, the aim of the 2nd step of Web Service Discovery in the proposed model is to determine the set of Web Services which can actually be used to solve the given Goal with regard to all related aspects. In general, this second step is not necessarily based on logic-techniques and reasoning. Here, the main aim is to eventually increase the precision of the discovery process drastically, for instance by consideration of the input data that is actually provided by the client.

There are several possibilities for realization of this functionality. We give a brief overview of proposals that have been discussed with the WSMO working group along with the deficiencies of each proposal:

- ***Hypothetical Execution of the real-world Service:*** the strictest method is to actually execute all possible Web Services – thus obviously detecting only usable Ser-

[1] In fact, this can often happen if we assume that clients mostly use predefined goal templates in goal repositories to express their desires and refine them by providing concrete input values. Service capability descriptions are advertisements for services and thus will not be generated for single invocations but rather for long-time use.

vices. But this method is certainly not the most pleasant solution, as it brings up several problems: a Web Service might expects critical input data, the execution of a service might be very time consuming and all the executions have to be rolled back after the hypothetical execution. Furthermore, service providers have to be willing to support potentially numerous rollbacks without actual use for the provider and checking the usability of a single service is service-dependent in general; for instance, one has to be able to determine algorithmically that the actual results and outputs of the service execution satisfy the Goal specification.

- ***Hypothetical Execution of the Service Description:*** an exhaustive framework based on a transaction-logic approach has been defined in [6]. This approach basically executes a service on a logical level (rather than in the real-world) according to their specifications: based on a notion of Input that is transmitted from the Goal to the Web Service, the Capability precondition is checked to be satisfied; if so, the service postcondition and effects are assumed to hold and the Goal postcondition and effects are validated. If both are satisfied the Service is considered to match the Goal. This approach obviously contradicts the design of WSMO Goals; besides, it only works on the Capability description of a Web Service. This presumes that the Web Service Capability completely and correctly describes the functionality of a Web Service, which might be considered as an unrealistic assumption for many application domains like Semantic Web Service based eCommerce[2].
- ***Additional Information Facilities in the Service Interface:*** another solution proposed with regard to the eCommerce-related services is to define additional 'information facilities' for Semantic Web Services, that is a set of additional (and perhaps standardized) functions that are invocable as part of the Web Service Interface in order to gather the needed information that is missing in the service description (and acquired in the other approaches by service execution). An example would be a `getProductPrice()`-method that retrieves the price of a certain product out of the internal product database. This approach is an application-specific technique that contradicts the design of WSMO components as well.

It is obvious that this step of Web Service Discovery is very much dependent on specific application scenarios or technologies, or it requires additional description elements which are not defined in WSMO (and are not by intention). We do not claim the above mentioned efforts to be wrong or not usable: they might serve as ideas and starting points for more elaborated mechanisms for this step of Web Service Discovery. In any way, as there seems not to be any generic solution, this step is considered not to be part of the general Web Service Discovery provided within WSMO.

[2] Imagine the Amazon.com Web Service and a Goal "buying a certain book for less than 10$": the Service Capability Description can not list the price for each product of Amazon, therefore one can not be sure that the Amazon-Web Service can solve this Goal when following this purely logical approach. For the sake of accuracy and simplicity of service descriptions, some sort of real-world interaction with the service (or its provider) will have to take place.

2.3 Selection of a Service to be used

The final step of WSMO Web Service Discovery is to select one Web Service out of the 2nd step result set that shall be used to solve the clients Goal. This selection shall be based on user preferences in relation to Web Service specific non-functional properties as defined in WSMO Standard [8]. Although these aspects have not been elaborated in detail yet, the WSMO reference implementation specifies a specific component "Selector" for this discovery step [10].

3 A prototypical Implementation of WOOGLE

After outlining the overall conceptual model of Web Service Discovery within WSMO, this section discusses the prototypical realization of WOOGLE on top of the First-order Logic theorem prover VAMPIRE [7].

3.1 WOOGLE with VAMPIRE

VAMPIRE is a saturation based theorem prover for First-order Logic with equality and since several years now one of the most powerful theorem provers for this kind of logic. In our case, we want to reason about proof obligations in the WSML Service Description Language. The precise proof obligation for WOOGLE will be discussed in detail in [13]. In order to being able to use VAMPIRE for our problem, we need a translation of the proof obligation (and hence WSML expressions) to First-order Logic with equality. In fact, this is not a difficult task: The WSML language is inspired by and based on F-Logic [11], which in wide parts (i.e. the non-monotonic part) can be mapped to First-order Logic [12]. Since WSML does not contain features of F-.Logic which give rise to non-monotonic behavior during inferences, the same translation can be used in our case.

In the following we briefly outline the formalization that is used in SWF. Since the formal modeling of services and goals in WSMO is under development at present, this modeling might be adapted in the future. Clearly, this will not present a serious problem for our approach, since it is rather unlikely that the proof obligations that we eventually have to deal with are not expressible in First-order Logic with equality.

A goal is represented by two unary predicates g-post(x) which defines the set of instances a user is interested in and g-eff(x) which specifies the set of desired effects. A service is represented in the same way by two unary predicates ws-post(x) which defines the set of instances that can be delivered by the service and ws-eff(x) which specifies the set of effects that are caused by the service execution. All these four unary predicates P(x) are formally defined by a formula $\forall x.(P(x) \leftrightarrow \phi(x))$, where $\phi(x)$ is an arbitrary F-Logic formula which (possibly) contains x.

A service is considered to (canonically) match a goal, if the sets of desired instances and effects are all *completely* delivered by the services. Formally, this means that we prove $\forall x.(\text{gs-post}(x) \rightarrow \text{ws-post}(x)) \wedge \forall x.(\text{gs-eff}(x) \rightarrow \text{ws-eff}(x))$ with respect to some collection of ontologies. Often, this kind of match is called *plug-in match.*

Furthermore, the canonical definition of matching immediately suggests further and slightly different notions of matching, namely: A service is considered to match (*strong contributing match*) if it delivers parts of what the user wants to have (as specified in the

Goal) as the output and effects (but *no* other kinds of information or effects). Formally, this means $\forall x.(\text{ws-post}(x) \rightarrow \text{gs-post}(x)) \wedge \forall x.(\text{ws-eff}(x) \rightarrow \text{gs-eff}(x))$. In this case the client would have to consult additionally a couple of matching services (in this sense), if he is actually interested in getting all the instances and effects that the Goal predicates specify.

An even weaker notion than the strong contribution match (commonly referred to as intersection match) drops the requirement that the service does not deliver instances and causes effects apart from those mentioned in the Goal. Formally, this is expressed by $\exists x.(\text{ws-post}(x) \wedge \text{gs-post}(x)) \wedge \exists x.(\text{ws-eff}(x) \wedge \text{gs-eff}(x))$. We call this notion *(weak) contribution match*. In particular, this notion is appropriate, if the user is only interested in getting some instance and effects that are specified in the goal. Clearly, the weak contribution match is the weakest possible notion of matching in our approach of formal modeling of Services and Goals.

As the informal descriptions already suggest, the appropriate proof obligations to be used for a specific discovery request depends on how the user interprets the sets of instances and effects that he formalized as part of his request: If his intuition for this request is that all elements of the set are needed in order to claim a match , then the plugin-criterion is the appropriate one, whereas if his intuition is that the set of elements represents the set of valid solutions to his problem and that only some of these elements are actually needed to resolve his Goal, then the (weak) contribution matches would be the suitable notions for modeling his desire. Furthermore, we can imagine that it is also useful to allow the usage of different of these notions to the postcondition part and the effect part in the same proof obligation. Please note, that all this kinds of notions (and any combined notions as well) can principally be supported by our prototype.

In summary, this means that a discovery request in our modeling approach not only consits of a Goal specification, but additionally the user gives some additional information about his intention with the given modeling by explicitly (or indirectly) selecting the appropriate notion of matching that is to be applied when processing his discovery request.

3.2 Advantages and Shortcomings

We came up with a rather general approach that allows to deal with a very rich modeling language (WSML-FOL, as well as everything that can be mapped into WSML-FOL, e.g. WSML-Core and WSML-DL). This allows us to deal with WSML descriptions on a very general level with the least possible efforts. In particular, the most difficult component, the deduction system has been reused almost without any efforts. Now, we are in a suitable position to look at actual use cases and evaluate this rather naïve approach for implementation of WOOGLE in a concrete application context.

On the other hand, First-order Logic is a very expressive logic and thus there are no computational guarantees for finding proofs (or recognizing that such a proof can not be found) like in less expressive formalisms, e.g. Description Logics. This might be a problem in general, but does not necessarily have to be a problem in our specific applications. For instance, in [15] it is shown that VAMPIRE can indeed be used for Description Logic reasoning without extreme drawbacks in comparison to specialized and highly optimized inference systems. Indeed, one needs to perform actual and realistic experiments to make a meaningful statement in this regard.

The architecture of VAMPIRE at present does not seem to support the maintenance of a knowledge base. That means that all ontologies to which Goal and Service descriptions refer to have to be (re)loaded for every proof task. Clearly, this is not desirable and has to be resolved for complex applications where we have to deal with big ontologies.

Most Description Logic and Logic Programming systems support so-called concrete domains (like integers and strings) and operations on these domains (+, <, substring) by an extra-logical component in order to compute these operations efficiently. Such a component is not present in any First-order Logic theorem provers and integrating such a component is not straightforward. Instead, dealing with concrete domains is realized by axiomatizing these domains. Unfortunately, this method (although more general) is less efficient for many standard tasks, e.g. computing the value of the product of two given integers.

At present, we only implemented a straightforward translation of WSML-FOL to First-order Logic. On the other hand, as experience in translation-based theorem proving for logics like Modal Logics or Description Logics show, the efficiency of a theorem prover can significantly be improved by using a suitable (clever) translation. For instance in [15] it is demonstrated that some sort of preprocessing and filtering during the translation process can drastically prune the search space for the prover. Clearly, our translation has to be carefully analyzed and optimized in this respect in the course of a concrete use case.

In conclusion, we can summarize that we have chosen this translation-based approach (and VAMPIRE) for building a reasoner for WSML-FOL to primarily come up quickly with a system that allows us to actually experiment with service discovery using WSML (in particular WSML-FOL) and that is flexible enough to easily allow to adapt to changes in the style of formal modeling of Goals and Services and thus the proof obligations that have to be established by the system. Moreover, VAMPIRE is the most powerful deduction system for First-order Logics at present and thus it makes perfect sense to start with this system as the underlying deduction system for our reasoner and the discovery component. It is important to note that we conceptually have only a loose coupling of our system to VAMPIRE: the proof obligation actually is serialized according to TPTP format [14], which is the de-facto standard for automated theorem proving systems and can be used as an input format for all available major ATP systems for First-order Logic. Thus, we principally can exchange VAMPIRE at any point in time without any efforts.

4 WOOGLE meets Semantic Web Fred

As an overview of the background on the WOOGLE development in Semantic Web Fred, we outline the usage of WOOGLE in SWF and explain its realization within the SWF GS Discovery Module as an example for integration into a WSMO-enabled application

4.1 SWF Overview

The aim of the SWF project is to develop advanced mechanisms for cooperative goal resolution within the FRED system, and agent platform developed by Net Dynamics (www.netdynamics-tech.com), and to align these with Semantic Web Service technologies emerging around WSMO. In SWF, Goals are solved within cooperations between agents (called *Freds*): each Fred has a Goal to be solved, and it needs Services to automate the goal resolution. SWF distinguishes between Goal Schemas as predefined Goals, whereof Goal Instances can be created that are assigned to Freds for resolution. Goal Schemas are WSMO Goals, while Goal Instances hold additional information (a ‘submission’ as the data that are submitted as Input to a Service, the Goal Owner id, and resolution status information). Services in SWF are described as WSMO Web Services. Similar to WMSO, ontologies are used as the data model throughout the whole system, and WSMO Mediators can be included to resolve possibly occurring heterogeneities.

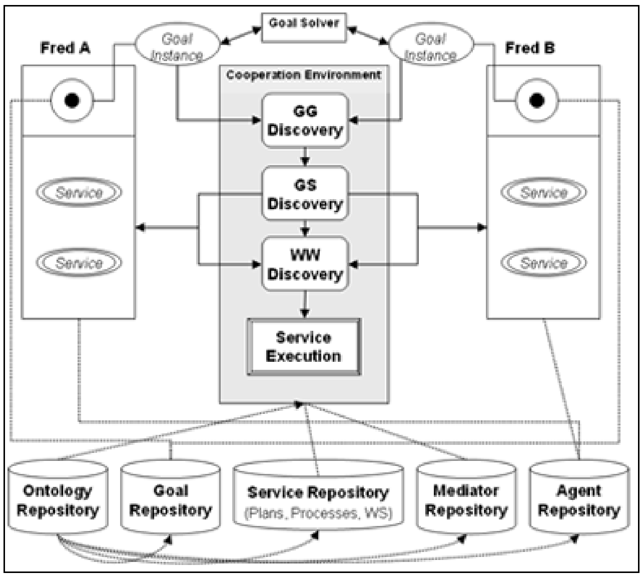

Figure 1. SWF Architecture

Figure 1 shows the overall architecture of SWF, wherein three discovery mechanisms establish cooperative goal resolution between Freds. After a successful pass through all of these mechanisms, the Freds are sent into a Meeting where the cooperation contract is executed.

- **GG Discovery.** At first, potential cooperation partners have to be detected by matching Goal Instances assigned to Freds. Goals are considered to be compatible if their objects of interest (i.e. the postconditions and effects) match, and when the cooperation roles of the Goal Instance owners are compatible.
- **GS Discovery.** Next, each partner has to find the Services that he needs for the cooperation. Therefore, GS Discovery detects suitable Services available in the system similar to Web Service Discovery.
- **WW Discovery.** For automated execution, the Services detected in GS Discovery need to have compatible Choreography Interfaces. WW Discovery checks this, resulting in a choreographed service execution model.

In fact, these mechanisms increase the rate of productive meetings (i.e. wherein Goals are really resolved): they do not predict success of a meeting because some aspects will never be determinable by description only – never mind how elaborated discovery mechanisms are.

4.2 The GS Discovery Module of Semantic Web Fred

The WOOGLE facility is incorporated into the SWF GS Discovery Module. This is realized as an open-source Java-Module which consists of the following elements that are needed for every system integration of WOOGLE[3].

Registry API. Interface to retrieve component descriptions out of a Registry / Repository. Here, this is realized for the WSMO Registry [3].

WSML Parser and Translator. Out of the components' descriptions in WSML [1,8], and abstract representation is derived which is then transformed into the format required by the theorem prover, e.g. the TPTP format supported by all major FOL theorem provers like VAMPIRE.

Prover and ProverConnector. The component which actually tries to establish a proof for given proof obligations and the component which abstracts away from how to locate and talk to a specific deduction system. The prover connector for instance can be used realize easily a distributed network of provers which can dynamically grow and shrink (on top of a Tuple-space blackboard infrastructure) in order to address scalability of the system.

GS Discoverer. Defines the control structure for the GS Discovery process, i.e. receives a Goal and Service descriptions as input and determine the result set of the discovery. The SWF GS Discoverer integrates WOOGLE as well as a solution for the 2nd step of Web Service Discovery as outlined above.

5 Conclusions and Future Work

In this paper we have outlined the general approach for Web Service Discovery in WSMO, and we have defined the conceptual model as well as means for realization of WOOGLE. As a basic but powerful discovery mechanism for Semantic Web Service, WOOGLE is intended to be provided as a standard discoverer within WSMO. Although the overall Web Service Discovery is comprised of further and extendable steps, WOOGLE is a generic solution that is compliant with the WSMO. We also outlined the position of WOOGLE in SWF as an example for integration into a more complex Semantic Web Services system

Future work is to evaluate the performance of our prototype implementation of WOOGLE, to identify and realize potential for optimization, and to provide a WOOGLE prototype with a Web Service Interface for seamless integration into other WSMO-enabled systems on the WSMO Web Server for demonstration purposes. Besides, the work presented here is considered as a working proposal for definition of a final overall framework for Web Service Discovery in WSMO.

[3] All sources of the GS Discovery Module along with a Use Case for SWF and some documentation are available at the web interface of the SWF CVS at: http://cvs.deri.at/cgi-bin/viewcvs.cgi/swf/.

Acknowledgement

The work presented is funded by the Austrian government under the CoOperate programme 2003, and by the European Commission under the IST projects DIP and SWWS. We like to thank the WSMO working group for fruitful input; special thanks go to Michael Kifer, Holger Lausen, Ruben Lara, and Axel Polleres for intensive and helpful discussions and to Net-Dynamics for likewise fruitful input and support for the realization Service Discovery within Semantic Web Fred.

References

1. de Bruijn, J. (ed.): *WSML-Core*. WSML Work Draft D16.7, 13 August 2004.
2. Fensel, D.; Bussler, C.: *The Web Service Modeling Framework WSMF*. Electronic Commerce Research and Applications, 1(2), 2002.
3. Herzog, R.; Zugmann, P.; Stollberg, M.; Roman, D.: *WSMO Registry*, WSMO Working Draft D10, 26 April 2004.
4. Lara, R.; Roman, D.; Polleres, A.; Fensel. D.: *A Conceptual Comparison of WSMO and OWL-S.* To appear in Proceedings of the European Conference on Web Services (ECOWS'04).
5. Keller, U.; Lara, R. (ed.): *Inferencing support for Semantic Web Services. Proof Obligations.* WSML Working draft D5.1, August 2nd, 2004.
6. Kifer, M.; Lara, R.; Polleres, A..; Zhao, C.; Fensel, D.; Keller, U.; Lausen, H.; Stollberg, M.: *A Logical Framework for Web Service Discovery*. Submitted to the ISWC 2004 Workshop on Semantic Web Services: Meeting the World of Business Applications.
7. Riazanov, A.; Voronkov, A.: *The design and implementation of VAMPIRE.* AI Communications 15(2), Special issue on CASC, pp. 91 -110, 2002.
8. Roman, D.; Lausen, H.; Keller, U. (eds.): *Web Service Modeling Ontology - Standard (WSMO - Standard) v 1.0*, WSMO Working Draft D2, 16 August 2004.
9. The OWL Services Coalition: OWL-S: Semantic Markup for Web Services, version 1.1, 2004.
10. Zaremba, M. (ed): *WSMX Architecture.* WSMX Working Draft D13.4, 22 June 2004.
11. Kifer M. ; Lausen G.; Wu J.: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM 42(4), pp. 741-843, 1995.
12. Fensel D. ; Decker S. ; Erdmann M. ; Studer R.: *Ontobroker: How to make the WWW intelligent*, Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems, Workshop (KAW98) (Banff, Canada), 1998.
13. Keller, U.; Lara, R; Polleres A.. (ed.): *Discovery of Semantic Web Services.* WSML Working draft D5.1, to appear at www.wmso.org/wsml.
14. Sutcliffe G.; Suttner C.: *The TPTP Problem Library for Automated Theorem Proving*. See: http://www.cs.miami.edu/~tptp/
15. Tsarkov D.; Horrocks I.: *DL Reasoner vs. First-order Prover*. In *Proc. of the 2003 Description Logic Workshop (DL 2003),* volume 81 of *CEUR,* pages 152-159, 2003.
16. Bachmair L.; Ganzinger H.: *Resolution Theorem Proving*, Chapter 2 in Handbook of Automated Reasoning, Vol. I, A. Robinson and A. Voronkov (Eds.), pp. 19-99, 2001