

Exploiting Semantic Technology in Computational Logic-based Service Contracting

Marco Alberti¹, Massimiliano Cattafi², Marco Gavanelli², and Evelina Lamma²

¹ CENTRIA, Universidade Nova de Lisboa
Quinta da Torre - 2825-144 Caparica (Portugal)

`m.alberti@fct.unl.pt`

² ENDIF, Università di Ferrara

Via Saragat, 1 - 44100 Ferrara (Italy)

`massimiliano.cattafi@unife.it`

`marco.gavanelli@unife.it`

`evelina.lamma@unife.it`

Abstract. Dynamic composition of web services requires an automated step of contracting, i.e., the computation of a possibly fruitful interaction between two (or more) services, based on their policies and goals. In previous work, the *SCIFF* abductive logic language was used to represent the services' policies, and the associated proof procedure to perform the contracting.

In this paper, we build on that work in order to exploit the results of the Description Logics research area to represent domain specific knowledge, either by importing the knowledge encoded in an ontology into a *SCIFF* knowledge base, or by interfacing the *SCIFF* proof procedure to an existing ontological reasoner.

1 Introduction

Service Oriented Architectures, exploiting the Internet as a means of communication, are emerging as a simple and effective paradigm for distributed application development. Heterogeneous entities, in terms of hardware and software settings, can interoperate effectively following well established communication standards.

Service providers can expose their policies, in order for potential customers to evaluate the feasibility of a fruitful interaction. Such an evaluation can be performed manually, if the service's interface is bound not to change over time.

However, a more promising approach is for customers to choose the appropriate service provider at run-time, based on the service provider's exposed policies.

This approach requires a reasoning engine that reasons on the provider and customer's goals and policies, in order to devise a sequence of actions that lets both achieve their goals, while respecting their policies.

In [1], Alberti et al. proposed a contracting architecture based on the *SCIFF* abductive logic framework [2]. In that work, the policies were expressed by means of integrity constraints, and the domain knowledge was expressed in *SCIFF*'s logic clauses. The sound and complete *SCIFF* proof procedure was employed to find, if possible, a course of action that was satisfactory for both.

However, in a practical perspective, we envisage a possible improvement in representing the domain specific knowledge exploiting the vast body of results from the Knowledge Representation field and, in particular, Description Logics. In this way, we can address the Ontology layer of the Semantic Web stack.

In this paper, we propose two different approaches to let *SCIFF* access existing knowledge, expressed by means of an ontology:

- by translating (part of) an ontology into *SCIFF* clauses, which can subsequently be handled by the *SCIFF* proof procedure in the usual way;
- by interfacing *SCIFF* with ontological reasoners, distinguishing syntactically the ontology-related predicates and delegating their computation to the external reasoners.

This approach bears two main practical advantages: first, it makes the knowledge encoded in ontologies available to *SCIFF*, and second (with the second approach) it takes advantage of the formal properties enjoyed by the existing ontological reasoners for the associated computational tasks, in particular concerning decidability and efficiency.

The paper is structured as follows: in Sect. 2 we briefly recall the *SCIFF* approach to contracting, in Sect. 3 we show how the relevant domain knowledge can be expressed by means of an ontology, and in Sect. 4 we describe the two different approaches to access ontologies from *SCIFF*: importing an ontology into a *SCIFF* knowledge base (Sect. 4.1), and interfacing the *SCIFF* proof procedure with the Pellet [3] ontological reasoner (Sect. 4.2). A brief discussion of related work and conclusions follow.

2 Contracting with *SCIFF*

In this section, we briefly recall the *SCIFF*-based contracting framework [1].

2.1 The *SCIFF* framework

A web service's policy is defined, in the *SCIFF* language, as an abductive logic program (ALP). An ALP is defined as the triplet $\langle KB_S, \mathcal{A}, IC \rangle$, where KB_S is a logic program in which the clauses can contain special atoms, that belong to the set \mathcal{A} and are called *abducibles*. Such atoms are not defined by means of clauses in the KB_S , and, as such, they cannot be proved: their truth value can be only hypothesized. In order to avoid unconstrained hypotheses, a set of *integrity constraints* (IC) must always be satisfied. Integrity constraints, in our language, are in the form of implications, and can relate abducible literals, defined literals, as well as constraints with Constraint Logic Programming [4] semantics.

In particular, the integrity constraints can relate the web services' information exchanges with the expected input from peer web services. The possible relations can be of various types, and include temporal relations, such as deadlines, linear constraints, inequalities and equalities, all defined by means of constraints. The definitions stated in this way are then used to make assumptions on the possible evolutions of the interaction.

A web service can reason about *happened* events, denoted with $\mathbf{H}(S, R, M, T)$, meaning that a Sender S sends message M to a Receiver R at some time T . All the parameters can be logical terms, or variables, possibly subject to constraints. Also, the policy might describe that a peer should provide some input, by sending a message. Such *expectation* of the incoming message is represented as $\mathbf{E}(S, R, M, T)$, and it will be satisfied in case the message, in the form of \mathbf{H} atom, actually arrives. In some cases, the *expectation* could be *negative*, as in $\mathbf{EN}(S, R, M, T)$: the policy says in this case that a matching \mathbf{H} event would be unwelcome, as it would contradict other assumptions.

For example, an integrity constraint can state that, if a peer web service sends me a request, then I will accept:

$$\mathbf{H}(S, me, request(X), T) \rightarrow \mathbf{H}(me, S, accept(X), T_a).$$

Another rule can say that, if I send a request, I expect the peer either to accept or to refuse, within 5 time units:

$$\begin{aligned} \mathbf{H}(me, S, request(X), T) \rightarrow & \mathbf{E}(S, me, accept(X), T_a), T_a < T + 5 \\ & \mathbf{E}(S, me, refuse(X), T_r), T_r < T + 5 \end{aligned}$$

Once the specification of a policy is defined by means of integrity constraints and expectations, they are processed by the *SCIFF* proof procedure, the operational counterpart of the language. The proof-procedure performs abductive reasoning, i.e., it can make assumptions and generate hypotheses. In particular, it can hypothesize that some message will be sent, or that some expectation will be raised. In case the expectations are matched by corresponding events, the abductive process succeeds, otherwise the current branch will fail, and another alternative will be selected (if there exists one). In this way, the *SCIFF* proof-procedure is able to find if there exists at least a set of events that satisfies a given ALP, and provides in output both the abduced events and expectations.

In contracting applications, the potential interacting services expose their policies (in a format resulting from the extension of RuleML), and *SCIFF* operates on the merging of the policies. A successful computation yields a sequence of actions that satisfies all the parties' policies.

2.2 A contracting scenario

In [1] the *SCIFF* framework is applied to contracting in an e-commerce scenario, which we extend in this paper in order to demonstrate our approach to integration.

The two actors are *eShop* (which wants to sell a device) and *alice* (a potential customer for that device), each with policies expressed as *SCIFF* ICs. *SCIFF* is used as a reasoner in a component that acts as a mediator, considering the actors' policies and trying to devise a course of action that will let both reach their goals. The two actors' policies, expressed in *SCIFF* integrity constraints, are as follows.

alice's policy. "If the shop asks me to pay cash, I will, but if the shop asks me to pay by credit card, I will require evidence of the shop's affiliation to the Better Business Bureau."

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Shop}, \text{alice}, \text{ask}(\text{pay}(\text{Item}, \text{cc}))), \text{Ta}) \rightarrow \\ & \mathbf{H}(\text{tell}(\text{alice}, \text{Shop}, \text{request_guar}(\text{BBB})), \text{Trg}) \wedge \\ & \mathbf{E}(\text{tell}(\text{Shop}, \text{alice}, \text{give_guar}(\text{BBB})), \text{Tg}) \wedge \text{Tg} > \text{Trg} \wedge \text{Trg} > \text{Ta}. \end{aligned}$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Shop}, \text{alice}, \text{ask}(\text{pay}(\text{Item}, \text{cc}))), \text{Ta}) \wedge \\ & \mathbf{H}(\text{tell}(\text{Shop}, \text{alice}, \text{give_guar}(\text{BBB})), \text{Tg}) \rightarrow \\ & \mathbf{H}(\text{tell}(\text{alice}, \text{Shop}, \text{pay}(\text{Item}, \text{cc})), \text{Tp}) \wedge \text{Tp} > \text{Ta} \wedge \text{Tp} > \text{Tg}. \end{aligned} \tag{1}$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Shop}, \text{alice}, \text{ask}(\text{pay}(\text{Item}, \text{cash}))), \text{Ta}) \rightarrow \\ & \mathbf{H}(\text{tell}(\text{alice}, \text{eShop}, \text{pay}(\text{Item}, \text{cash})), \text{Tr}) \wedge \text{Ta} < \text{Tr}. \end{aligned}$$

eShop's policy. “If an acceptable customer requests an item from me, then I expect the customer to pay for the item with an acceptable means of payment. If the customer is not acceptable, I will inform him/her of the failure. If an acceptable customer pays with an acceptable means of payment, I will deliver the item. If a customer requests evidence of my affiliation to the Better Business Bureau (BBB), I will provide it.”

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Customer}, \text{eShop}, \text{request}(\text{Item})), \text{Tr}) \wedge \text{accepted_payment}(\text{How}) \rightarrow \\ & \text{accepted_customer}(\text{Customer}) \wedge \mathbf{H}(\text{tell}(\text{eShop}, \text{Customer}, \text{ask}(\text{pay}(\text{Item}, \text{How}))), \text{Ta}) \wedge \\ & \mathbf{E}(\text{tell}(\text{Customer}, \text{eShop}, \text{pay}(\text{Item}, \text{How})), \text{Tp}) \wedge \text{Tp} > \text{Ta} \wedge \text{Ta} > \text{Tr} \\ & \vee \text{rejected_customer}(\text{Customer}) \wedge \mathbf{H}(\text{tell}(\text{eShop}, \text{Customer}, \text{inform}(\text{fail})), \text{Ti}) \wedge \text{Ti} > \text{Tr}. \end{aligned}$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Customer}, \text{eShop}, \text{pay}(\text{Item}, \text{How})), \text{Tp}) \\ & \wedge \text{accepted_customer}(\text{Customer}) \wedge \text{accepted_payment}(\text{How}) \rightarrow \\ & \mathbf{H}(\text{tell}(\text{eShop}, \text{Customer}, \text{deliver}(\text{Item})), \text{Td}) \wedge \text{Td} > \text{Tp}. \end{aligned}$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(\text{Customer}, \text{eShop}, \text{request_guar}(\text{BBB})), \text{Trg}) \rightarrow \\ & \mathbf{H}(\text{tell}(\text{eShop}, \text{Customer}, \text{give_guar}(\text{BBB})), \text{Tg}) \wedge \text{Tg} > \text{Trg}. \end{aligned}$$

(2)

The notion of acceptability for customers and payment methods from *eShop's* view-point, defined by the *accepted_customer/I* and *accepted_payment/I* predicates, is defined in *eShop's* knowledge base. In [1], only EU residents are accepted customers, as defined by the following clauses:

```
accepted_customer (Customer) :-
    resident_in (Customer, Location),
    accepted_destination (Location).
rejected_customer (Customer) :-
    resident_in (Customer, Location),
```

```
not_accepted_destination(Location).
accepted_destination(european_union).
accepted_payment(cc).
accepted_payment(cash).
```

This knowledge is merged with that provided by the customer, for example

```
resident_in(alice,european_union).
```

so that in the resulting knowledge base, on which *SCIFF* operates, `accepted_customer(alice)` is true.

However, this method would not work in case the customer declared `resident_in(alice,italy)`, as *italy* does not unify with *european_union*. One could, of course, add to the knowledge base `accepted_destination/1` facts for all the current EU members, but such knowledge should be updated when new countries join the EU. In other cases, acceptability could be defined by a transitive, symmetric relation, which could introduce cycles in the knowledge base, possibly leading to loops.

3 Representing domain knowledge with ontologies

An alternative way to represent (part of) the domain specific knowledge is to use technology and concepts developed, with focus on this very purpose, in the Knowledge Representation field, and to rely, in particular, on ontologies. The W3C recommendation for ontology representation on the Web is the Web Ontology Language (OWL) [5] based on the well established semantics of Description Logics [6] and on XML and RDF syntax. Using OWL for domain knowledge representation improves expressiveness (with such features as stating sub-class relations, constructing classes on property restrictions or by set operators, defining transitive properties and so on) yet keeping decidability (if using OWL Lite or OWL DL) in a straight-forward and domain modeling-oriented notation. Moreover, since OWL is tailored for the Web, it provides support for expressing knowledge in distributed contexts (identified by URIs) and its recognized standard status is a warranty on interoperability and reuse issues. As a plus, it can be mentioned that community driven development of Semantic Web tools already provides good support for OWL ontology management tasks such as editing [7] also for not KR-skilled users.

For instance, in Fig. 1 we show a possible ontological representation of *eShop*'s policies concerning acceptable customers and means of payments, merged with *alice*'s own knowledge.

The following OWL axiom says that the `acceptedCustomer` class is a subclass of the `potentialCustomer` class, and that it is disjoint from the `rejectedCustomer` class:

```
<owl:Class rdf:about="#acceptedCustomer">
  <rdfs:subClassOf rdf:resource="#potentialCustomer" />
  <owl:disjointWith rdf:resource="#rejectedCustomer" />
</owl:Class>
```

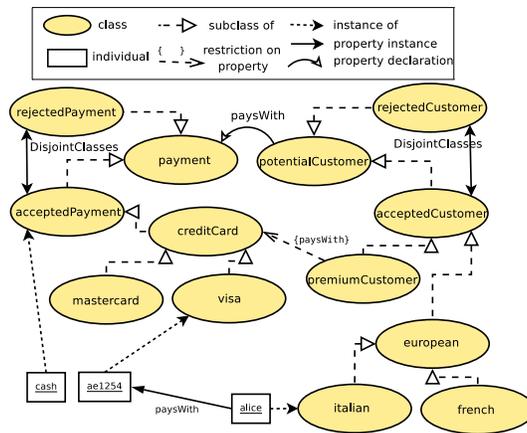


Fig. 1. A graphical representation of the ontology

The following fact states that `cash` is an instance of the `acceptedPayment` class:

```
<owl:Thing rdf:about="#cash">
  <rdf:type rdf:resource="#acceptedPayment" />
</owl:Thing>
```

The following axiom declares the `paysWith` property:

```
<owl:ObjectProperty rdf:ID="paysWith">
  <rdfs:domain rdf:resource="#potentialCustomer" />
  <rdfs:range rdf:resource="#payment" />
</owl:ObjectProperty>
```

The following fact states that `alice` is an instance of `italian`, with value `ae1254` for the `paysWith` property:

```
<owl:Thing rdf:about="#alice">
  <rdf:type rdf:resource="#italian" />
  <paysWith rdf:resource="#ae1254" />
</owl:Thing>
```

It can be noticed that the ontological notation for the KB makes it possible to infer that `alice` is European (and therefore an accepted customer) even if she just states being Italian, while if we had put `resident_in(alice, italy)` instead of `resident_in(alice, europe)` in the KB at the end of Sect. 2.2 she would not have been recognized as such.

Moreover, we defined a class `premiumCustomer` representing the accepted customers who pay with a credit card, and which we could use to add refinement to policies (for instance providing a faster delivery). Since `alice` is an accepted customer

and pays with her credit card, the ontological reasoning allows to recognize her as a `premiumCustomer`.

4 Handling semantic knowledge with *SCIFF*

We implemented the *SCIFF* access to OWL ontologies in two different ways, that we describe in this section. In both cases it should be noticed that even if OWL relies on the Open World Assumption, when *SCIFF* comes to reasoning involving it, the logic programming peculiar Closed World Assumption is reintroduced, thus assuming to have all (relevant) information on the domain available at that time and providing usual features such as negation as failure. In this section we present the two approaches, and their experimental evaluation.

4.1 Importing ontologies into the *SCIFF* KB_S

Considering that Logic Programming (LP) and Description Logics (DL) have a common root in First Order Logic, a first approach is to find their intersection and to translate ontologies to LP clauses. These two problems have already been addressed by Grosz et al. [8] who named this intersection DLP (Description Logics Program) and by Hustadt et al. [9] who proposed a method for translation. On these basis, the `dlpconvert` [10] tool was developed and made available by the Karlsruhe University, which converts (the DLP fragment of) OWL ontologies to datalog clauses. We used `dlpconvert` to translate domain knowledge described in OWL documents to *SCIFF* clauses. Reasoning is then performed by *SCIFF* in the usual way. However, this solution limits ontological expressivity. First of all, since the DLP fragment is a proper subset of DL, some OWL axioms are not included. For instance, out of the features mentioned as available in [11] by Vrandečić et al., `DisjointClasses` and the important DL (and OWL) feature of class definition by restriction on properties are missing. Moreover, some axioms' translation is not actually suitable for reasoning with goal-driven operational semantics, such as resolution or unfolding, employed in *SCIFF*, because it leads to loops. As an example, consider

```
<owl:Class rdf:about="#danaan">
  <owl:equivalentClass rdf:resource="#achaeen"/>
</owl:Class>
```

whose translation is

```
danaan(X) :- achaeen(X).
achaeen(X) :- danaan(X).
```

and

```
<owl:ObjectProperty rdf:ID="ancestorOf">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="#person" />
  <rdfs:range rdf:resource="#person" />
</owl:ObjectProperty>
```

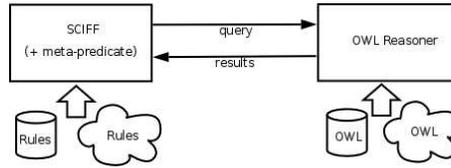


Fig. 2. Integration architecture

whose translation is

```
ancestorOf(X, Z) :- ancestorOf(X, Y), ancestorOf(Y, Z).
```

For all these reasons the e-commerce ontology must be modified, in particular removing the premiumCustomer class, and the DisjointClasses axioms.

4.2 Interfacing *SCIFF* and ontological reasoners

An approach that aims to overcome the limitations described in Sect. 4.1 consists of interfacing *SCIFF* with an external specific ontology-focused component which can be, when necessary, queried by *SCIFF* and which performs the actual ontological reasoning and gives back results. As represented in Fig. 2, the architecture for this solution involves a Prolog meta-predicate which invokes the ontological reasoning on desired goals, an intercommunication interface from *SCIFF* to the external component (which incorporates a query and results translation schema) and the actual reasoning module. Both modules can access both local and networked knowledge.

Goals given to the meta-predicate are handled like suggested in [9] and [11] considering single arity predicates as “belongs to class (with same name of predicate)” queries and double arity ones as “are related by property (with same name of predicate)” queries. To reduce the overhead caused by external communication, our implementation of the meta-predicate provides a caching mechanism: it is first checked if a similar query (i.e., involving the same predicate) has been performed before and, only if not, the external reasoner is invoked and answers are cached by storing them as Prolog facts (by means of `asserta/1`). The OWL reasoning module uses the Pellet [3] API, while the communication interface uses the Jasper Prolog-Java library [12]. This solution enables access to the full OWL(-DL) expressivity, including features such as equivalence of classes and properties, transitive properties, declaration of classes on property restriction and property-based individual classification.

4.3 Experimental evaluation

Both approaches were tested successfully in simple reasoning scenarios, such as the one depicted in Fig. 1 (as far as allowed by the expressiveness limitations of the ontology import approach discussed in Sect. 4.1).

Performance comparisons of the two approaches are hardly significant, as they mainly differ in expressive power. However, to assess their scalability, we experimented

N	Ontology import			Interface with Pellet		
	Load time	Query time	Total	Load time	Query time	Total
100	3.4	~ 0	3.4	~ 0	~ 0	~ 0
500	5.8	~ 0	5.8	1.0	~ 0	1.0
1000	8.2	~ 0	8.2	1.0	~ 0	1.0
5000	14.9	~ 0	14.9	2.0	1.2	3.2
10000	26.6	~ 0	26.6	4.0	2.8	6.8

Table 1. Performance comparison (all times are in seconds, average over 50 runs)

with randomly generated ontologies. Each ontology, composed of N classes, was built starting from its root node, and recursively attempting, for each node, five attempts of child generation, each with probability $1/3$.

The reasoners were queried about the belonging of an entity to the hierarchy root class. For both approaches, we report in Tab. 1 the time spent for loading the ontology into the reasoner³ and for the actual query. Both approaches appear to scale reasonably (on PC equipped with an Intel Celeron 2.4 GHz CPU). In both cases, the loading time is higher than the query time; for the importing approach, this is particularly noticeable, and encouraging, as in real applications the ontology would be imported only once.

5 Related work

One of the first works about the ontological representation of knowledge in a logic programming context is F-Logic [13] (implemented in Flora-2 [14]). F-Logic ontologies are similar to the formalism of frames and provide an attribute-value and taxonomical (with inheritance) structure for objects and classes. However they lack the feature of class definition based on restriction on attributes (or properties), which is a peculiar characteristic of DL (and OWL). Even if the Semantic Web community adopted the DL paradigm for the ontology layer of the Semantic Web cake, the urge for introducing rules for improving expressivity, for instance to model Semantic Web Services, partially made F-Logic come back as a suitable candidate, like in the case of WSMO (Web Services Modeling Ontology [15]) framework. An extensive study of how rules and ontologies can be integrated, with a specific focus on Semantic Web, can be found in [16] where a language, WSML, is proposed to be used as a Web Service Modeling Language in WSMO. Kifer et al. in [17] propose a comprehensive solution based on Flora-2, in which F-logic is used to express ontologies and Transaction Logic is used to model declaratively how services behave.

On the other hand, work is being done for the foundations of a tight integration of Description Logic and Logic Programming and one of the most important issues is how to deal with the different assumptions (Closed World for LP, Open World for DL) made by these two families of languages. In [18] decidability and complexity results are

³ For the importing approach, the load time is the time spent for translating the ontology and parsing the resulting clauses and ICs, while for the Pellet-based approach it is the time spent for parsing ICs and loading the ontology into a persistent OWLOntology object.

presented for an integration of DL and Disjunctive Datalog, while in [19] an integration of DL and LP is based on the Minimal Knowledge and Negation as Failure (MKNF) logic [20]. In [21] this proposal is extended providing a three-valued semantics. In our work, we apply the closed world assumption to ontological predicates, when computing their truth value in a SCIFF derivation; this simplification has proved acceptable for the applications that we have considered.

6 Conclusions

In this paper, we proposed an integration of the previously developed SCIFF reasoning framework for service contracting with Description Logics standards. We proposed two approaches: first, to translate ontological knowledge into the SCIFF formalism; second, to interface the SCIFF reasoner to ontological reasoners, delegating the ontological reasoning to them. In this way, we make the results (in terms of representation capability and computational efficiency) obtained in the Description Logics area available to the SCIFF-based contracting framework.

In future work, we intend to experiment with the integration of ontological reasoning to other applications of the SCIFF framework, such as agent programming [22], following the approach proposed by Moreira et al. [23].

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: Web service contracting: specification and reasoning with SCIFF. In Franconi, E., Kifer, M., May, W., eds.: ESWC. Volume 4519 of LNAI. (2007)
2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logics* **9** (2008)
3. Parsia, B., Sirin, E.: Pellet: An OWL DL reasoner. In van Harmelen, F., ed.: ISWC 2004. (2004)
4. Jaffar, J., Maher, M.: Constraint logic programming: a survey. *J. of Logic Programming* **19-20** (1994) 503–582
5. : W3C recommendation: OWL, web ontology language. <http://www.w3.org/TR/owl-guide/> (2004)
6. Lutz, C.: Description logic resources. dl.kr.org (2008)
7. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Ferguson, R.W., Musen, M.A.: Creating semantic web contents with Protégé-2000. *IEEE Int. Systems* **16** (2001) 60–71
8. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: WWW '03, ACM (2003) 48–57
9. Hustadt, U., Motik, B., Sattler, U.: Reducing $SHIQ^-$ description logic to disjunctive datalog programs. (In Dubois, D., Welty, C., Williams, M.A., eds.: KR2004)
10. Motik, B., Vrandečić, D., Hitzler, P., Sure, Y., Studer, R.: Dlpconvert - Converting OWL DLP statements to logic programs. System Demo at the 2nd ESWC (2005)
11. Vrandečić, D., Haase, P., Hitzler, P., Sure, Y., Studer, R.: DLP-an introduction. Tech.Rep., Univ. Karlsruhe (2006)
12. : (SICStus Prolog) <http://www.sics.se/sicstus>.

13. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object oriented and frame based languages. *Journal of the Association for Computing Machinery* **42** (1995) 741–843
14. : Flora-2: An object-oriented knowledge base language. (<http://flora.sourceforge.net/>)
15. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., M.Stollberg, A.Polleres, C.Feier, C.Bussler, Fensel, D.: Web service modeling ontology. *Appl. Ontology* **1(1)** (2005)
16. de Bruijn, J.: Semantic Web Language Layering with Ontologies, Rules, and Meta-Modeling. PhD thesis, Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck (2008)
17. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In Martin, D., Lara, R., Yamaguchi, T., eds.: *SWS*. Volume 119 of *CEUR Workshop Proc.* (2004)
18. Rosati, R.: *DI+log*: Tight integration of description logics and disjunctive datalog. In Doherty, P., Mylopoulos, J., Welty, C.A., eds.: *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, AAAI Press (2006) 68–78
19. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In Veloso, M.M., ed.: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. (2007) 477–482
20. Lifschitz, V.: Nonmonotonic databases and epistemic queries. In Mylopoulos, J., Reiter, R., eds.: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI), Sidney, Australia, Morgan Kaufmann (1991)* 381–386
21. Knorr, M., Alferes, J.J., Hitzler, P.: Towards tractable local closed world reasoning for the semantic web. In Neves, J., Santos, M.F., Machado, J., eds.: *Progress in Artificial Intelligence, 13th Portuguese Conference on Artificial Intelligence, EPIA 2007, Workshops: GAIW, AIASTS, ALEA, AMITA, BAOSW, BI, CMBSB, IROBOT, MASTA, STCS, and TEMA, Guimarães, Portugal, December 3-7, 2007, Proceedings*. Volume 4874 of *Lecture Notes in Computer Science.*, Springer (2007) 3–14
22. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: A verifiable logic-based agent architecture. In Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G., eds.: *Foundations of Intelligent Systems - 16th International Symposium, ISMIS 2006 Bari, Italy, September 27-29, 2006 Proceedings*. Volume 4203 of *Lecture Notes in Artificial Intelligence.*, Berlin Heidelberg, Springer-Verlag (2006) 188–197
23. Moreira, Á.F., Vieira, R., Bordini, R.H., Hübner, J.F.: Agent-oriented programming with underlying ontological reasoning. In Baldoni, M., Endriss, U., Omicini, A., Torroni, P., eds.: *Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*. Volume 3904 of *Lecture Notes in Computer Science.*, Springer (2005) 155–170