pDAML+OIL: A probabilistic extension to DAML+OIL based on probabilistic Datalog

Henrik Nottelmann
University of Duisburg-Essen
47048 Duisburg, Germany
nottelmann@uni-duisburg.de

Norbert Fuhr
University of Duisburg-Essen
47048 Duisburg, Germany
fuhr@uni-duisburg.de

Abstract

As a representative of ontology languages, DAML+OIL is well-suited for describing objects and their properties. We show that real-life problems require uncertain facts and general rules, which are not supported by DAML+OIL. We extend DAML+OIL accordingly while preserving as much of the original semantics as possible, by mapping DAML+OIL models onto probabilistic Datalog.

Keywords: DAML+OIL, probability theory, logics

1 Introduction

The Semantic Web aims at transforming the currently text-based web into a network of information with a well-defined meaning, which allows for sharing and reusing it across applications and communities. One major focus lies on defining standards for representing knowledge by means of ontologies. Turning the vision of a web with semantics into something practical is a long-term goal, where a lot of problems of the current approaches have to be overcome.

One of the Semantic Web languages based on description logics is DAML+OIL [11], built on top of RDF and RDF Schema.

RDF [10] is a specification language for objects and their properties via statements (also called triples): Its subject is an RDF object, the predicate is a property, and the object is another RDF object or a literal (a constant). The vocabulary can be defined with the schema specification language RDF Schema (RDFS for short) [12].

DAML+OIL enriches RDFS with more advanced primitives: E.g. the range of a property can be specified w.r.t. the domain (in RDFS, it can only be specified globally).

However, the DAML+OIL language has two significant restrictions:

1. As nearly all ontology languages, DAML+OIL is only able to specify and infer deterministic knowledge (which is either true or false). However, not all knowledge in the world is deterministic, uncertainty often plays an important role, and weights are required. They can be used to model assumptions about missing knowledge, e.g. the probability that someone is the father of a child unless this is certified by a test. Weights can also be used for specifying vague information needs.

2. DAML+OIL does not support rules (besides properties like `rdfs:subClassOf` which could be viewed as rules). DAML-L is intended to add rules to DAML+OIL, but this language is not yet completed. Rules can be used as a powerful query language, and are mandatory for Semantic Web Services. E.g., a travel planning system needs rules for deciding whether a specific flight and a specific hotel matches the user needs for a specific trip. They can also be used as a declarative way for defining schema mapping rules, when documents (or queries) have to be transformed from one schema into another.

Reification (statements about statements) is used in [6] for supporting uncertainty. Then, the property `prob` of a statement specifies the probability that this statement is correct. Reification is supported by RDFS, but DAML+OIL tools cannot give any semantics to such triples.

In contrast to this approach, we now extend the language itself to solve both problems. First we map DAML+OIL onto probabilistic Horn logics, namely Datalog, while preserving the semantics. We then employ its probabilistic extension, probabilistic Datalog (pDatalog for short) [2] for defining the semantics of probabilistic DAML+OIL, and for introducing rules. This approach has the advantage of using an existing framework for which tools (HySpirit [3], or a mapping onto relational databases) are already available. This is important because our major goal is to develop actual applications for our approach.

This paper is organised as follows. First we give a brief introduction in probabilistic Datalog. Section 3 shows how DAML+OIL models can be mapped onto pDatalog programs. Uncertainty and rules are added to DAML+OIL in Sec. 4. Then, an example (a travel planning system) shows how pDatalog and rules can be used in practice.

2 Probabilistic Datalog

This sections gives a brief overview over Datalog and its probabilistic extensions two-valued and four-valued probabilistic Datalog.

2.1 Datalog

Datalog [8] is a variant of predicate logic based on function-free Horn clauses. Negation is allowed, but its use is limited to achieve a correct and complete model. Rules have the form $h \leftarrow b_1 \wedge \cdots \wedge b_n$, where h (the "head") and b_i (the subgoals of the "body") denote literals[1] with variables and constants as arguments. A rule can be seen as a clause $\{h, \neg b_1, \ldots, \neg b_n\}$:

```
father(X,Y) :- parent(X,Y) & male(X).
```

[1]Literals in logics are different from literals in DAML+OIL!

This denotes that `father(x,y)` is true for two constants `x` and `y` if both `parent(x,y)` and `male(x)` are true. This rule has the head `father(X,Y)` and two body literals (considered as a conjunction) `parent(X,Y)` and `male(X)`.

In addition, negated literals start with an exclamation mark. Variables start with an uppercase character, constants with a lowercase character. Thus the rule expresses the semantics behind the `rdfs:subClassOf` property.

A fact is a rule with only constants in the head and an empty body:

```
parent(jo,mary).
```

The semantics are defined by well-founded models [9], which are based on the notion of the greatest unfounded set. Informally speaking, given a partial interpretation of a program, this is the maximum set of ground literals that can be assumed to be false.

Only modularly stratified programs [7] are allowed. Since the definition of modular stratification is rather complicated, we only give a simple explanation: In contrast to global stratification, modular stratification is formulated w.r.t. the instatiation of a program for its Herbrand universe. The program is modularly stratified if there is an assignment of ordinal levels to ground atoms such that whenever a ground atom appears negatively in the body of a rule, the ground atom in the head of that rule is of strictly higher level, and whenever a ground atom appears positively in the body of a rule, the ground atom in the head has at least that level.

2.2 Probabilistic Datalog

In probabilistic Datalog (see [2]), every fact or rule has a probabilistic weight attached, prepended to the fact or rule:

```
0.5 male(X) :- person(X).
0.5 male(jo).
```

Semantics of pDatalog programs are defined as follows: The pDatalog program is modelled as a probability distribution over the set of all "possible worlds". A possible world is the well-founded model of a possible deterministic program, which is formed by the deterministic part of the program and a subset of the indeterministic part. As for deterministic Datalog, only modularly stratified programs are allowed.

Computation of the probabilities is based on the notion of event keys and event expressions, which allow for recognising duplicate or disjoint events when computing a probabilistic weight.

Facts and instantiated rules are basic events, each of them has assigned a unique event key. Each derived fact is associated with an event expression that is a Boolean combination of the event keys of the underlying basic events. E.g., the event expressions of the subgoals of a rule form a conjunction. If there are multiple rules for the same head, the event expressions corresponding to the rule bodies form a disjunction. By default, events are assumed to be independent, so the probabilities can be multiplied.

2.3 Four-valued Probabilistic Datalog

In four-valued probabilistic Datalog [3], we switch from the closed-world assumption in Datalog to an open-world assumption: If we don't have any information, the fact has the truth value "unknown". On the other hand, a fact has the truth value "inconsistent" if there is evidence for both true and false.

For facts, probabilities for true, false and inconsistent have to be specified manually, the probability for unknown can then be derived easily.

```
0.6/0.2/0.2 male(jo).
0.9 female(maria).
!male(laura).
!male(X) :- female(X).
```

Jo is a man with probability 0.6, not a man with probability 0.2, and the knowledge is inconsistent with probability 0.2. For Maria, the "0.9" is a short form for "0.9/0.1/0". Laura is not a man, and women are not men (here, the negated head stands for the probabilities "0/1/0").

If positive and negative rules are true for a fact in the head, then this fact is inconsistent.

A model of a deterministic Datalog program can contain— for a specific fact—the positive atom, the negated atom, both the positive and the negated atom or neither of them. This corresponds to the four truth values. A model of a probabilistic program consists of a set of possible worlds, where each world is a model for the corresponding deterministic Datalog program, together with a probability distribution on the set of possible worlds.

Programs in four-valued probabilistic Datalog can easily be transformed into equivalent two-valued pDatalog programs: For each predicate, two distinct predicates for positive (true or inconsistent) and negative knowledge (false or inconsistent) are created in the corresponding two-valued program. For each fact, three facts are derived, namely one for each of the truth values true, false and inconsistent. In order to ensure proper computation of probabilities, we declare identical facts with different truth values as disjoint events.

3 Mapping DAML+OIL onto Datalog

Our goal is to define DAML+OIL axiomatic semantics [4] by mapping them onto Datalog. The semantics are originally defined in KIF [1]; we aim at preserving as much as possible in Datalog.

The resulting Datalog programs will later be used for adding probabilistic weights and rules to DAML+OIL.

3.1 Basic idea

We do not give a complete mapping from DAML+OIL onto Datalog. In contrast, we concentrate on the most important (and the most difficult) parts of the DAML+OIL definition.

The DAML+OIL universe consists of classes, properties and ground facts. Following [4], we map DAML+OIL classes onto unary predicates (with the class name being the predicate name), and properties onto binary predicates:

```
<daml:Class rdf:id="flight:Airport"/>
<flight:Airport rdf:ID="#DUS">
  <flight:close rdf:resource="#Dusseldorf"/>
</flight:Airport>
```

```
daml:Class(flight:Airport).
flight:Airport(#DUS).
flight:close(#DUS,#Dusseldorf).
```

As shown in the two previous facts, we slightly modify the Datalog language: Literals like `"2003-12-23"` will be enclosed in quotation marks. Resources are specified by their URI, where both absolute and relative (the latter starting with a hash sign #) URIs are allowed. For abbreviation, also namespaces can be used (as in element and attribute definitions in XML), e.g. daml:Class.

DAML+OIL modelling primitives are transformed into sets of Datalog rules:

```
<daml:Class rdf:id="flight:Airport">
  <rdfs:subClassOf rdf:resource=
                   "travel:Location"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="#parent">
  <daml:inverseOf rdf:resource="#child"/>
</daml:ObjectProperty>


travel:Location(O) :- flight:Airport(O).
travel:Location(O) :- travel:City(O).

child(X,Y) :- parent(Y,X).
parent(X,Y) :- child(Y,X).
```

Most of the other modelling primitives are handled in a similar way.

3.2 Modelling primitives with higher complexity

Some modelling primitives have a higher complexity and cannot be handled by pure Datalog alone.[2] E.g., we can model airports and cities as disjoint subclasses of locations:

```
<daml:Class rdf:id="travel:City">
  <rdfs:subClassOf rdf:resource=
                   "travel:Location"/>
  <daml:disjointWith rdf:resource=
                    "flight:Airport"/>
</daml:Class>


!flight:Airport(O) :- travel:City(O).
!travel:City(O) :- flight:Airport(O).
```

Here, negated rule heads are required, a feature which is not supported in pure Datalog. Thus, it is a natural decision to switch to four-valued logics. Incorrect models can then be detected easily by checking for inconsistent facts.

A similar approach can be used e.g. for disjointWith, unionOf, complementOf or uniqueProperty.

[2]In fact, these modelling primitives are excluded from OWL Lite due to their complexity.

Another example which cannot be covered by pure Datalog are cardinality constraints, as Datalog does not allow for counting. However, also cardinality constraints can be verified with a couple of rules:

```
<daml:Class rdf:ID="Person">
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource=
                       "#hasParent"/>
      <daml:cardinality>2</daml:cardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>


hasParent_diff3(O) :- hasParent(O,Y1) &
                      hasParent(O,Y2) &
                      hasParent(O,Y3) &
                      <>(Y1,Y2) &
                      <>(Y1,Y3) &
                      <>(Y2,Y3).
!Person(O) :- Person(O) & hasParent_diff3(O).
!Person(O) :- Person(O) & !hasParent_diff2(O).
```

Here, hasParent_diff3 contains all instances which have at least three different role-fillers for the property hasParent. The other two rules check for inconsistencies, i.e. that there are at least three fillers or that is does not have two distinct fillers. Note that this set of rules is only feasible for a small cardinality number n, as the auxiliary rules contains $O(n^2)$ literals.

Auxiliary rules also have to be used e.g. for qualified cardinality restrictions, hasClass and hasValue.

4 Probabilistic DAML+OIL

So far, we demonstrated how we can map DAML+OIL models onto Datalog programs. By using additional rules, we can model the DAML+OIL semantics by creating new knowledge (e.g. for subClassOf) and by verifying semantic constraints (e.g. disjointWith).

DAML+OIL and Datalog allow for deterministic (Boolean) facts only. Often, however, we need to incorporate uncertainty in DAML+OIL (then called pDAML+OIL). E.g., the property travel:close indicates if an airport is close to a city. Obviously, there is no strict definition of “closeness” involved. Furthermore, we add rules to probabilistic DAML+OIL itself which can be used by planning systems.

4.1 Probabilistic facts

As we already mapped DAML+OIL onto Datalog, now it is easy to add uncertainty by switching to four-valued probabilistic Datalog. The semantics of pDAML+OIL programs will be given by the corresponding pDatalog program.

As a consequence, every DAML+OIL statement can be weighted, and this weight obviously is the weight of the corresponding pDatalog fact. In the XML serialisation, an element defining a property simply has an additional attribute `pdaml:prob` (which can be omitted if the probability equals one):

```
<flight:Airport rdf:ID="#FLR">
  <travel:close rdf:resource="#Siena"
               pdaml:prob="0.7"/>
</flight:Airport>

<Person rdf:ID="#Bob" pdaml:prob="0.7/0.1/0.1"/>
```

The last fact describes an uncertain `rdf:type` property (it is unclear if Bob is a person).

This model can be mapped onto:

```
flight:Airport(#FLR).
0.7 travel:close(#FLR,#Siena).

0.7/0.1/0.1 Person(#Bob).
```

For modelling primitives, the situation is more difficult. As we use probabilistic Datalog as the underlying framework, the expression

```
<daml:Class rdf:id="A">
  <rdfs:subClassOf rdf:resource="B"
                   pdaml:prob="0.5"/>
</daml:Class>

0.5 subClassOf(#A,#B).
```

denotes, that the probability of an instance of `#A` being also an instance of `#B` is 50%.[3]

However, other constructs do not have clear semantics. E.g., an object is a class, or it is not a class, weights do not make sense in this case:

```
<daml:Class rdf:id="Man" pdaml:prob="0.5"/>
```

[3]Alternative interpretations which are not backed by pDatalog are: Exactly 50% of the instances of class `#A` are also instances of `#B`; or with a probability of 0.5 all instances of `#A` are also instances of `#B`.

4.2 Rules in pDAML+OIL

Advanced DAML+OIL applications require support for stating rules. An obvious solution is to map pDAML+OIL rules onto pDatalog rules.

For modelling this in pDAML+OIL, we introduce some new classes (see Fig. 1). Logical variables used in these rules are defined by `pdaml:Variable`. Literals are specified by instances of `pdaml:Literal`, a class similar to `rdf:Statement` but for rules instead of facts. Positive and negative literals, respectively, are modelled by sub-classes of `pdaml:Literal`. Instances of `pdaml:Rule` are rules with two properties `pdaml:head` (cardinality of 1, can be weighted) and `pdaml:body` (minimum cardinality of 1), both with range `pdaml:Literal`.

Complex DAML+OIL diagrams are hard to read. So, instead of describing a sample rule as a graph, we present the XML serialisation:

```
<pdaml:Variable rdf:ID="T"/>
<pdaml:Variable rdf:ID="F"/>

<pdaml:Rule>
  <pdaml:head pdaml:prob="0.6">
    <pdaml:PositiveLiteral>
      <rdf:subject rdf:resource="#T"/>
      <rdf:predicate rdf:resource=
                     "travel:candflight"/>
      <rdf:object rdf:resource="#F"/>
    </pdamlPositive:Literal>
  </pdaml:head>
  <pdaml:body>
    <pdaml:PositiveLiteral>
      <rdf:subject rdf:resource="#T"/>
      <rdf:predicate rdf:resource=
                     "travel:candflight0"/>
      <rdf:object rdf:resource="#F"/>
    </pdaml:PositiveLiteral>
  </pdaml:body>
  <pdaml:body>
    <pdaml:PositiveLiteral>
      <rdf:subject rdf:resource="#F"/>
      <rdf:predicate rdf:resource="rdf:type"/>
      <rdf:object rdf:resource=
                  "flight:TransferFlight"/>
    </pdaml:PositiveLiteral>
  </pdaml:body>
</pdaml:Rule>
```

This pDAML+OIL model of a rule is then mapped onto the pDatalog rule:

```
0.6 travel:candflight(T,F) :-
    travel:candflight0(T,F) &
    flight:TransferFlight(F).
```

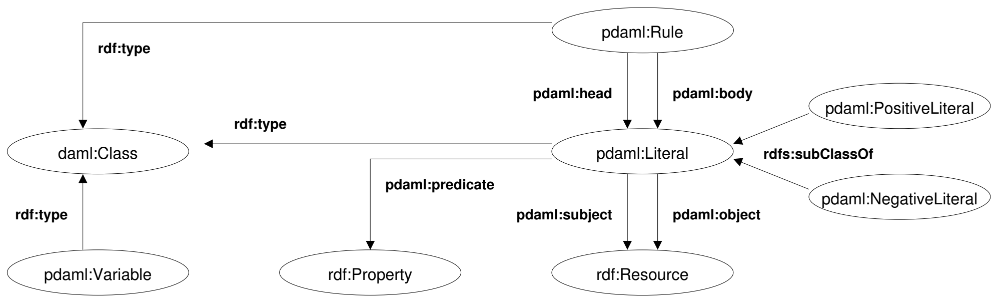

Figure 1: Rules in pDAML+OIL

Thus, the semantics are the following: If the variables `T` and `F` can be bound to instances of the classes `travel:Travel` and `flight:Flight`, respectively, so that there are facts corresponding to the body literals in the fact basis, then the additional fact corresponding to the instantiated head literal are added as intensional facts.

5 Example

In this section we present an example DAML+OIL model, a travel planning system, map it onto pDatalog, and use rules for planning the best travel details.

5.1 Scenario

In this paper, we consider a simple travel planning system. The ontology and the instances are defined in pDAML+OIL. A graphical version is depicted in Fig. 2. In the remainder, we only describe pDatalog facts due to space restrictions.

A planned travel is defined by the start and destination city, the latest possible start date and the earliest possible end date. Here, we plan a short Christmas trip from Dusseldorf to Siena from December 24 to December 25:

```
travel:Travel(#Travel).
travel:from(#Travel,#Dusseldorf).
travel:to(#Travel,#Siena).
travel:start(#Travel,"2003-12-24").
travel:end(#Travel,"2003-12-25").
```

These facts are constructed individually, for a specific travel planning request. The rest of the facts belong to a knowledge base which is independent from any specific travels. They can be stored in a local database, retrieved from a search engine, or collected from different sites.

A flight (more precisely, the combination of a flight and a return flight) is specified by the start and destination airport as well as the date of departure and the date of arrival at the starting airport. A flight can be a direct or a transfer flight. Airports can be close to one or more cities:

```
travel:City(#Dusseldorf).
travel:City(#Siena).
flight:Airport(#DUS).
flight:Airport(#FLR).
travel:close(#DUS,#Dusseldorf).
0.7 travel:close(#FLR,#Siena).

flight:TransferFlight(#Flight).
flight:from(#Flight,#DUS).
flight:to(#Flight,#FLR).
flight:start(#Flight,"2003-12-23").
flight:end(#Flight,"2003-12-25").
```

Finally, we model hotels by the days where at least one room is free, and the city in which a hotel is located:

```
hotel:Hotel(#Hotel).
hotel:free(#Hotel,"2004-12-23").
hotel:free(#Hotel,"2004-12-24").
hotel:in(#Hotel,#Siena).
```

5.2 Rules for planning travels

For describing potential airports for a trip, we introduce two new properties `travel:fromairport` and `travel:toairport`:

```
travel:fromairport(T,A) :-
    travel:from(T,C) &
    travel:close(C,A).
travel:toairport(T,A) :-
    travel:to(T,C) &
    travel:close(C,A).
```

In our case, we obtain facts:

```
travel:fromairport(#T,#DUS).
0.7 travel:toairport(#T,#FLR).
```

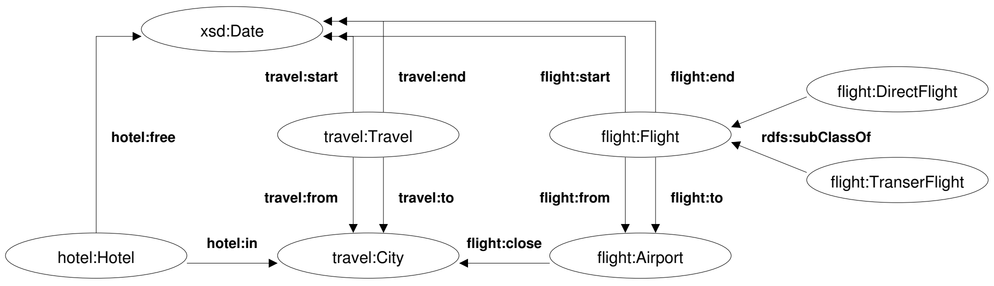

Figure 2: Example ontology for travel planning

We want to find potential flights w. r. t. the potential airports and our travel dates, where direct flights are preferred over transfer flights:

We are looking for flights whose departure date is specified by `travel:start` (or earlier), and which leave the destination city on the date specified by `travel:end` or later.

```
travel:candflight0(T,F) :-
    travel:fromairport(T,FA) &
    travel:traveltoairport(T,TA) &
    travel:start(T,S) &
    travel:end(T,E) &
    flight:from(F,FA) &
    flight:to(F,TA) &
    flight:start(F,FS) &
    flight:end(F,FE) &
    <=(FS,S) & >=(FE,E).

1.0 travel:candflight(T,F) :-
    travel:candflight0(T,F) &
    flight:DirectFlight(F).

0.6 travel:candflight(T,F) :-
    travel:candflight0(T,F) &
    flight:TransferFlight(F).
```

We only obtain one candidate flight:

```
0.42 travel:candflight(#T,#Flight).
```

Finally, we have to ensure that on every day (except the last one) we get a free room in the hotel. As we cannot specify the all-quantor directly, we have to a use a trick with negation:

```
travel:notfree(T,H) :-
    hotel:Hotel(H) &
    travel:candflight(T,F) &
    flight:start(F,FS) &
    flight:end(F,FE) &
    between(D,FS,FE) &
    !hotel:free(H,D).

travel:candhotel(T,H) :-
    travel:Travel(T) &
    hotel:Hotel(H) &
    !travel:notfree(T,H).
```

This returns our hotel in Siena:

```
0.42 travel:candhotel(#T,#Hotel).
```

6 Conclusion and outlook

Popular description logics like DAML+OIL are useful for defining types, objects and their properties. However, two important features, uncertainty and rules, are missing.

In this paper we extended the language DAML+OIL for coping with both of them. We defined the semantics of the resulting language pDAML+OIL by mapping the models onto four-valued probabilistic Datalog rules. Then, the semantics of the resulting pDatalog program is used as the semantics of the original pDAML+OIL model.

We also showed that this extension to DAML+OIL is useful in a practical environment. In our travel planning example, we can define uncertain knowledge like the closeness of an airport to a city, and we can describe rules for deciding whether a travel plan (flight and hotel) is useful.

DAML+OIL is equivalent to the $\mathcal{SHOIQ}(D)$ description logics [5], and all mappings from DAML+OIL onto pDatalog we presented in this paper could also be done from $\mathcal{SHOIQ}(D)$. Not all of its features, in particular those with a high computational complexity (e.g. `disjointWith`) can be mapped onto Datalog [4]. In this paper, we showed how this can be solved by switching to four-valued logics.

In the future, we will have a closer look on the remaining features of DAML+OIL, how they can be mapped onto pDatalog in practice, and investigate the computational complexity of this mapping. We will also develop guidelines when weighting is useful (e.g., it is not useful for specifying that something is a class).

We are currently developing a prototype for parsing pDAML+OIL models, mapping them onto four-valued pDatalog, and for evaluating the resulting programs.

We also consider to switch to OWL [13], the successor of DAML+OIL. OWL DL is similar to DAML+OIL, OWL Lite is derived by removing the features with raise complexity problems.

Acknowledgements

This work is supported in part by the EU commission (grant IST-2000-26061, MIND), and in part by the DFG (grant BIB47 DOuv 02-01, PEPPER).

We wish to thank the anonymous reviewers for their helpful comments, and Umberto Straccia for fruitful discussions.

References

[1] R. Fikes and D. L. McGuinness. An axiomatic semantics for RDF, RDF schema, and DAML+OIL. Technical report. KSL-01-01, `http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomati% c-semantics.html`.

[2] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[3] N. Fuhr and T. Rölleke. HySpirit – a probabilistic inference engine for hypermedia retrieval in large databases. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proceedings of the 6th International Conference on Extending Database Technology (EDBT), Valencia, Spain*, Lecture Notes in Computer Science, pages 24–38, Heidelberg et al., 1998. Springer.

[4] B. N. Grosof, I. Horrocks, S. Decker, and R. Volz. Description logic programs: combining logic programs with description logic. In *Proceedings of the twelfth international conference on World Wide Web*, 2003.

[5] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705, pages 161–180. Springer-Verlag, 1999.

[6] H. Nottelmann and N. Fuhr. Combining DAML+OIL, XSLT and probabilistic logics for uncertain schema mappings in MIND. In *European Conference on Digital Libraries (ECDL 2003)*, Heidelberg et al., 2003. Springer.

[7] K. Ross. Modular stratification and magic sets for Datalog programs with negation. *Journal of the ACM*, 41(6):1216–1266, Nov. 1994.

[8] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville (Md.), 1988.

[9] A. van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[10] W3C. Resource description framework (RDF) model and syntax specification. Technical report, World Wide Web Consortium, Feb. 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[11] W3C. DAML+OIL (march 2001). Technical report, World Wide Web Consortium, 2001. `http://www.w3.org/TR/daml+oil-reference`.

[12] W3C. RDF vocabulary description language 1.0: RDF Schema. Technical report, World Wide Web Consortium, Apr. 2002. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[13] W3C. OWL. Technical report, World Wide Web Consortium, 2004. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.