Bubo - Implementing OWL in rule-based systems

Raphael Volz
Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe, Germany
volz@aifb.uni-karlsruhe.de

Stefan Decker
ISI
University of Southern California (USC)
Marina-Del-Rey, CA
stefan@isi.edu

Daniel Oberle
Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe, Germany
oberle@aifb.uni-karlsruhe.de

ABSTRACT

The Semantic Web is build around a semi-structured data model - RDF - and an explicit conceptualization for such data - so-called ontologies. A standardized language for the specification of the latter has recently be proposed by the W3C. This paper explores the strategies for the implementation of this language in logic programming environments such as Prolog and relational databases. Along these lines we establish the subset of OWL primitives that is compatible for further rule-based extensions paving the way to the upper levels of the Semantic Web layer cake. We also capture a subset, for which query languages can easily be implemented by compiling expressions given in a declarative query language to the query language offered by the system on top of which the implementation is based.

1. INTRODUCTION

The Semantic Web is build around a semi-structured data model - RDF - and an explicit conceptualization for such data - so-called ontologies. A standardized language for the specification of the latter has recently be proposed by the W3C. This language, OWL [12], is based on a description logic SHIQ [8]. Description logics offer efficient support for questions about a conceptualization, namely whether classes are equivalent or disjoint to each other, whether a class subsumes another class, or whether a given class description is satisfiable at all.

The Query language perspective. However, the capabilites of description logics with respect to instances is rather low. It cannot even express the least-expressive query language usually taken into account by database research - conjunctive queries [3]. This area is a strong hold of logic programming, which offers highly expressive constructs for instance reasoning. Hence, it is very promising to combine description logics with this paradigm to obtain the ability to state expressive instance queries on terminological knowledge bases.

The rule language perspective. Another perspective for this combination is the extension of OWL knowledge bases with logic program rules, which offer further modelling capabilities. The layer of rule languages has already been envisioned by Tim-Berners Lee in his famous Semantic Web layer cake (cf. Figure 1) and a large group of people from the logic programming community are working on RuleML[1], a possible candidate for this layer. However, RuleML is not layered on top of the ontology layer, as envisioned, instead it operates on the data layer only. Hence, the ontology and rules worlds are split. Our approach establishes a link between the two worlds and opens the possibility to state rules on top of terminological knowledge bases.

The data integration perspective. The majority of today's data resides in relational databases. This will not change when the Semantic Web grows. Most likely people will start exporting their data as RDF instances according to some ontology they have chosen. This essential leads to data that is replicated to enable ontology-based processing of that data. Today, the latter is done by reading some files into a classifier, such as FaCT [8] or Racer [6]. However, logic programming systems such as XSB [15] allow to access database data directly through built-in predicates. Furthermore, stratified Datalog programs, a restricted variant of logic programs with limited expressivity, can directly be implemented on top of SQL99-compliant relational databases. Hence, a LP-based implementation of OWL allows a closer interaction with live data.

The implementation perspective. Currently, no full implementation of OWL is available. In order to become a success many implementations of OWL must be available. Many free and commercial implementations of logic programming systems are available. SQL99-compliant databases enjoy an even wider user community. This paper shows how these systems can be used as a basis for reasoning with OWL.

The paper is structured as follows. Section 2 gives a short overview about the relation between the aforementioned Semantic Web layers and gives a brief introduction to OWL. Section 3 shows how instance data may be represented in LP systems and discusses the chosen representation with respect to efficiency and ease-of integration with legacy data sources. Section 4 details the basic principles used to map of OWL ontologies to LP knowledge bases. Section 5 discusses the mappings in detail and establishes the OWL/LP fragment. The subset of LP, which can be implemented on top of relational databases is presented in Section 7. Section 6 introduces our prototypical implementation. We conclude discussing related work summarizing our contribution and giving an outlook to future work.

2. THE SEMANTIC WEB

2.1 The basic idea

The term *Semantic Web* encompasses efforts to build a new WWW architecture that enhances content with formal semantics. This will enable automated agents to reason about Web content, and carry out more intelligent tasks on behalf of the user. *Expressing mean-*

[1] http://www.dfki.uni-kl.de/ruleml/

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

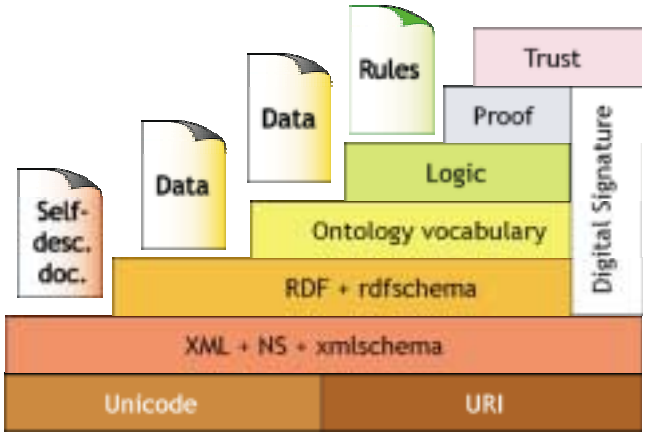


Figure 1: Semantic Web Layer Cake

ing is the main task of the Semantic Web. In order to achieve this objective several layers of representational structures are envisioned. Figure 1 presents the layers of the Semantic Web: *(I)* the XML layer is used as a syntax layer, *(II)* the RDF layer represents the data layer, *(III)* the ontology layer, based on a formal common agreement, specifies meaning and structure of the data, *(IV)* the logic layer provides rules that enable further intelligent reasoning, *(V)* the proof layer supports the exchange of *proofs* in inter-agent communication.

2.2 The XML syntax layer

XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean. Tag-names per se do not provide semantics. The Semantic Web utilizes XML for syntax purposes only.

2.3 The RDF data layer

The Resource Description Framework [10] is an infrastructure that enables encoding, exchange and reuse of structured metadata. Principally, information is stored in the form of RDF statements, which represent data in an uniform way (subject, predicate, object). This simple edge-labeled graph model facilitates machine understandability by resolving syntactic ambiguities. This abstract model is serialization independent, though the proposed standard serialization relies on XML. Unfortunally, the semantics specified for this layer already defines first entailments. Hence, systems operating on RDF data should provide some reasoning mechanisms to gain full compatibility with the standard. Due to the generality of the data model RDF offers modelling primitives that can be extended according to the needs at hand.

2.4 Specifying meaning - the ontology layer

The generality of RDF allows to build the third basic component of the Semantic Web - ontologies. In Artificial Intelligence and Web research the term ontology describes a formal, shared conceptualization of a particular domain of interest. By defining shared and common domain theories, ontologies help both people and machines to communicate concisely, supporting the exchange of semantics and not only of syntax.

Imagine a simple genealogy application. Apparently, the domain description, viz. the ontology, will include classes that talk about Persons and make a distinction between Males and Females . People are related with each other by several relations expressing parenthood and siblings. Ergo, properties like hasParent, childOf etc. will be in place. This domain description can be easily constructed with standard description logics (cf. Figure 2) .

Very recently a working group at W3C has continued the work of several research programs to come up with a recommendation for an ontology language. The language has two layers of primitives: a reduced set called OWL Lite and a full set of primitives OWL/DL. OWL/DL corresponds largely to an established description logic variant called $\mathcal{SHIQ}$ [8]. A Description logic is defined recursively by starting from a schema S of class names $\mathcal{CN}$, property names $\mathcal{PN}$ and names for individuals $\mathcal{IN}$. The semantics of terms is given denotationally, using the notion of an interpretation $\mathcal{I} =< \triangle^{\mathcal{I}}, (\cdot)^{\mathcal{I}} >$, which starts with a domain of values $\triangle^{\mathcal{I}}$ and a mapping $(\cdot)^{\mathcal{I}}$ from class descriptions to subset of the domain, and property descriptions to sets of 2-tuples over the domain. Each individual name is associate to some value in $\triangle^{\mathcal{I}}$. The Interpretaion function is extended recursively to composite descriptions as given in Table 2.4.

The meaning of a description D is the mapping from interpretations $\mathcal{I}$ to extents $D^{\mathcal{I}}$ and a variety of queries can now be defined on this basis, *(I)* whether a description E subsumes D, this is the case iff for every interpretation $\mathcal{I}$, $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. *(II)* whether a description D is coherent/satisfiable, this is the case if there is at least one $\mathcal{I}$ such that $D^{\mathcal{I}} \neq \emptyset$, and *(III)*whether descriptions E and D are disjoint, this is the case iff for every interpretation $\mathcal{I}$, $D^{\mathcal{I}} \cap E^{\mathcal{I}} = \emptyset$.

The reader may note, that OWL additionally features the primitives `FunctionalProperty`, `InverseFunctionalProperty` [2] and `SymmetricProperty`. All can straightforwardly be expressed via a combination of other primitives provided in $\mathcal{SHIQ}$. For example, a `SymmetricProperty` can be expressed by saying that it is inverse to itself (P `inverseOf` P).

A further layer of OWL with an extended semantics that is fully compatible to the RDF semantics is also defined in the specification, but not considered here since no efficient reasoning strategies are known for this variant. Also we do not consider the issue of datatypes, which is still under active discussion.

2.5 Further reasoning - The logic layer

While the ontology layer already provides means to deduce new information and provides restricted reasoning support, many applications require further means to combine and deduce information. If we return to the given example, the sisters and aunts of a person have to be stated explicitly. However, within a rule-based system, it is easy to build rules which capture those facts automatically, e.g.:

```
sisterOf(X,Y) :- childOf(X,Z), childOf(Y,Z), Woman(Y).
auntOf(X,Y) :- childOf(X,Z), sisterOf(Z,Y).
```

Logic programming systems, such as Prolog, HiLog[4] and Frame Logic [9], offer efficient environments to do so.

A large group of people from the logic programming community are working on a standard for exchange of rules in the Semantic Web called RuleML[3]. Although a possible candidate for this layer, RuleML is not layered on top of the ontology layer, as envisioned, instead it operates on the data layer only. Hence, the ontology and rules worlds are split. This paper therefore investigates how those two worlds can be related to each other. Not surprisingly, as we will see in section 4, the worlds are not disjoint.

Figure 3 sketches the semantic relation between RDF, OWL and the world of rules. We will establish two intersections of OWL and logic programming: OWL/LP (red) and OWL/Datalog (purple). The OWL/Datalog fragment can be safely implemented on

[2] not shown in the table

[3] http://www.dfki.uni-kl.de/ruleml/

OWL Primitives	Interpretation
Thing	$\triangle^{\mathcal{I}}$
Nothing	$\oslash$
C subClassOf D	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
C unionOf D	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
C intersectionOf D	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
complementOf C	$\triangle^{\mathcal{I}} \backslash C^{\mathcal{I}}$
C disjointWith D	$C^{\mathcal{I}} \cap D^{\mathcal{I}} = \oslash$
P subPropertyOf Q	$P^{\mathcal{I}} \subseteq Q^{\mathcal{I}}$
C sameClassAs D	$C^{\mathcal{I}} = D^{\mathcal{I}}$
P samePropertyAs Q	$P^{\mathcal{I}} = Q^{\mathcal{I}}$
x sameIndividualAs y	$x^{\mathcal{I}} = y^{\mathcal{I}}$
x differentIndividualFrom y	$x^{\mathcal{I}} \in \triangle^{\mathcal{I}} \backslash \{y\}$
TransitiveProperty P	$\forall p, q, r[(p, q) \in P^{\mathcal{I}} \wedge (q, r) \in P^{\mathcal{I}} \rightarrow (p, r) \in P^{\mathcal{I}}]$
SymmetricProperty P	$\forall p, q[(p, q) \in P^{\mathcal{I}} \leftrightarrow (q, p) \in P^{\mathcal{I}}]$
P inverseOf Q	$P^{\mathcal{I}} = Q^{\mathcal{I}^{-1}}]$
P minimumCardinality C *n*	$\{x \in \triangle^{\mathcal{I}} \mid \sharp\{y \mid (x, y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geqslant n\}$
P maximumCardinality C *n*	$\{x \in \triangle^{\mathcal{I}} \mid \sharp\{y \mid (x, y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leqslant n\}$
P allValuesFrom C	$\forall x, y[(x, y) \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}]$
P someValuesFrom C	$\forall x \exists y[(x, y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}]$
P hasValue b	$\forall x[(x, b) \in P^{\mathcal{I}}]$
oneOf($b_1, b_2, ..., b_i, b_n$)	$\{b_1^{\mathcal{I}}, ..., b_n^{\mathcal{I}}\}$

Table 1: OWL Primitives and their semantics

top of relational databases using known techniques such as magic templates [13], while OWL/LP can be easily implemented in Prolog variants such as XSB [15]. The later involves further steps such as the implementation of skolemization to substitute existential quantifiers by functions and the axiomatization of equivalence (cf. 4).

The established fragments can then be extended with additional rules operating on the ontology to obtain the intended reasoning support. If an appropriate exchange syntax is standardized for such rules, this extension can also be communicated across systems.

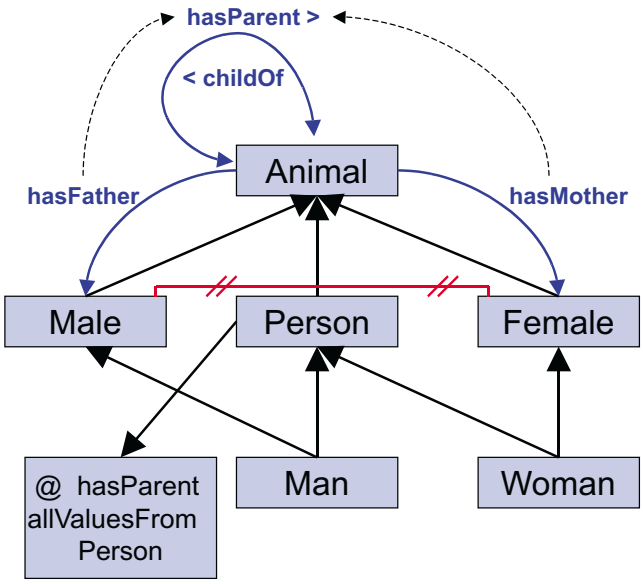
3. DATA REPRESENTATION

OWL ontologies are syntactically represented in RDF. RDF allows a very simple form of representation, which maps each RDF statement to one logical ternary predicate such as chosen in [14, 2]:

```
statement(subject,predicate,object)
```

Here, subject and predicate correspond to RDF resources and object is either a RDF resource or a RDF datatype/literal. This vertical form of representation is often chosen to representing sparse data with a large number of attributes such as found in many e-Commerce or digital library scenarios [1].

The usage of such a single ternary relation for storage of directed labelled-graphs such as RDF seems to be the most simple solution. On the other hand, each resource-value pair could be stored in a separate binary relation on a per property basis. The evaluation of [1] suggests that this representation also slightly outperforms the naïve ternary representation.



OWL Definitions

Class Definitions

Person subClassOf Animal
Male subClassOf Animal
Man subClassOf Person
Woman subClassOf Person
Person subClassOf (hasParent allValuesFrom Person)
Male disjointWith Female
Female subClassOf Animal
Man subClassOf Male
Woman subClassOf Female

Property Definitions

hasFather subPropertyOf hasParent
hasMother subPropertyOf hasParent
hasParent inverseof childOf

Figure 2: Genealogy Ontology example

3.1 Data integration perspective

From a perspective of integrating existing data the binary representation offers additional benefits (cf. Figure 4. Existing data can be transformed to the appropriate binary form by means of relational view definitions. Such a view definition may also integrate multiple source tables by means of using the union operator. Using a single ternary representation would yield a single view definition, which is very complex to write and maintain and will ultimately lead to very poor performance due to the complexity of the involved query.

For example in Figure 4 several properties defined in the ontology have to be associated with concrete data that is stored in several relational tables. The mapping between the ontology and this live data is done by transforming the source data in several binary representations using SQL views. During this process data can be transformed, e.g. table columns can be concatenated. Keys from each table are translated to globally unique URIS and used as instance idenfiers[4]. Each binary view is then used as data source for building the extension of the respective property in the ontology. The reader may note that additional indirections have to be used to cope with derived facts in ObjectProperties (cf. Section 7)

3.2 Translation to predicates

[4]In the example this is done by appending a URI prefix, which has to be unique for each table. In case of compound keys a more sophisticated approach has to be used, e.g. coding each key component as parameters.

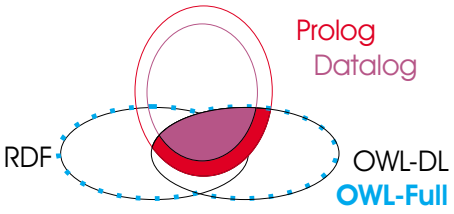


Figure 3: Relation between Horn Clause Programs, RDF and OWL

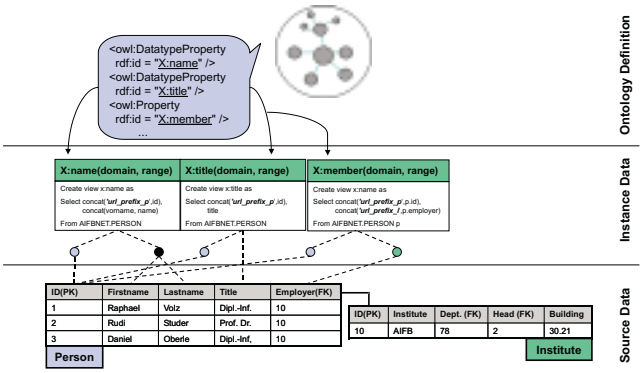


Figure 4: Integration of source data

RDF statements are written as binary predicates. Here the predicate of the statement is the name of the Datalog predicate. Hence, if (a, b) is instance of property P, we write the fact

$$P(a, b). \tag{1}$$

In our approach this representation is extended with additional unary relations which are used to store the class-individual relationship, where a separate predicate is created for each class. Hence, additionally to writing $type(a, C)$., facts stating that a is an instance of a class C, we write the fact

$$C(a). \tag{2}$$

This syntactic construction supports a more efficient processing of class extensions. Otherwise we would have to add rules, that distinguish between all three use cases of the RDF `type` predicate: class instantiation, class and property definition. As a side effect, a catalogue of classes and properties of their respective OWL type[5] is created.

4. OWL TRANSLATION PROCESS

This section now presents how and which parts of OWL can be implemented in logic programming environments. The translation is achieved through a mapping operator $\mathcal{T}$ that is applied recursively such as done in [3]. It takes DL-constructs as parameter and yields an unary predicate $C(x)$ for classes $\mathcal{T}^x\langle C\rangle$ and a binary predicate $P(x, y)$ for properties $\mathcal{T}^{x,y}\langle P\rangle$ and thereby fits into the aforementioned data representation scheme.

In order to translate n-ary class constructors such as union, enumeration and intersection, (n-1) intermediate anonymous class expressions are built, that provide a pairwise combination of descriptions in a nested manner. For example the DL expression like

$$C \equiv D \sqcap E \sqcap F \sqcap G$$

is translated to the following set of descriptions

$$C \equiv A_3, A_1 \equiv D \sqcap E, A_2 \equiv A_1 \sqcap F, A_3 \equiv A_2 \sqcap G$$

The reader may note, that this translation does not alter the specified semantics.

[5] such as ObjectProperty, SymmetricProperty etc.

OWL Class Expressions	$\mathcal{T}^x\langle C\rangle$
Thing	$x = x$
Nothing	$\neg(x = x)$
C subClassOf D	$\forall x[\mathcal{T}^x\langle D\rangle \leftarrow \mathcal{T}^x\langle C\rangle]$
C unionOf D	$\mathcal{T}^x\langle C\rangle \vee \mathcal{T}^x\langle D\rangle$
C intersectionOf D	$\mathcal{T}^x\langle C\rangle \wedge \mathcal{T}^x\langle D\rangle$
complementOf C	$\neg\mathcal{T}^x\langle C\rangle$
C disjointFrom D	$\neg\exists x(\mathcal{T}^x\langle C\rangle \leftrightarrow \mathcal{T}^x\langle D\rangle)$
P allValuesFrom C	$\forall y[\mathcal{T}^y\langle C\rangle \leftarrow \mathcal{T}^{x,y}\langle P\rangle]$
P someValuesFrom C	$\exists y[\mathcal{T}^y\langle C\rangle \wedge \mathcal{T}^{x,y}\langle P\rangle]$
C sameClassAs D	$\forall x[\mathcal{T}^x\langle C\rangle \leftrightarrow \mathcal{T}^x\langle D\rangle]$
P hasValue b	$\exists y(y = b) \wedge \mathcal{T}^{x,y}\langle p\rangle$

Table 2: Some OWL Class Constructors and their translation in FOL following

4.1 Translation to FOL

Borgida [3] showed how a prototypical description logic $\mathcal{DL}$, which is a superset of OWL, can be correctly translated into first order logic (FOL). Table 4.1 provides the appropriately adopted translation for OWL class expressions. In order to provide a translation for cardinality constraints [3] extends the syntax of FOL with counting quantifiers to map those DL constructors. The reader may note, that an alternative mapping to the unmodified FOL is possible using inequalities, but would lead to a more general subset of FOL than intended by [3].

The above mentioned translations are formula of FOL. Hence, the results cannot be directly transferred to Logic Programming environments, since only a subset of FOL is allowed there. The following sections discuss which fragment can be translated to LP. To proceed we first introduce our notion of LP.

4.2 Characterization of logic programming

Logic Programming is the language of first-order Horn clauses often extended with a closed-world negation and arithmetic predicates. The reader may note that the Horn clause form involves only universally quantified variables.

The basic elements of the language are predicates and terms. Terms can be either constant symbols such as names (so-called *atomic values*) and numbers (integer, float ...) or logical variable symbols. Usually variables and constants are distinguished by syntactic convention. In the following all variables will start with an upper-case letter. Each logical variable is a placeholder for another value, which can be instantied by substituting it by another term. However, it cannot be assigned directly.

Logic programs are composed of goals, queries and implications (also called *clauses*). Goals are syntactically represented by a predicate:

$$P(T_1, ..., T_n) \tag{3}$$

This states that predicate P is true of terms T_1 and all other T_i. Queries are syntactically represented as a set of goals:

$$E_1, E_2, ..., E_n \tag{4}$$

This retrieves all E_i. Clauses are syntactically written like implications in Prolog:

$$E_0 : -E_1, ..., E_n. \quad (5)$$

This means, that E_0 must be true, if all E_i are also true. E_0 is also called the head of the clause, while the remaining E_i are called the body of the clause. The body of the clause may be empty. Then the clause is called a *fact*. The clause is also referred to as a *rule* if the body is nonempty. We speak of *recursive rules* if a predicate appears in both the head and the body of a rule. Each rule corresponds to a FOL formula, where all occuring variables are universally quantified. This forumla contains a single implication between body and head. The body itself is translated to a conjunction of literals. For example, the rule $E_0(X,Y) : -E_1(X), E_2(Y)$ corresponds to the following FOL formula

$$\forall X, Y : (E_1(X) \wedge E_2(X)) \rightarrow E_0(X,Y)$$

4.3 Handling existentials

Many translations of OWL descriptions involve existential quantification. To implement existential quantification, a procedure called *Skolemization* must be applied.

Skolemization is a syntactic transformation routinely used in automatic inference systems in which existential variables are replaced by 'new' functions applied to universal variables appearing in front of the existential quantifier of the variable. If the original formula is satisfiable, then so is the skolemized formula.

4.4 Handling equivalence

Many translations of OWL descriptions require the use of equivalence. However, an *equivalence* predicate is usually not available in Logic Programming environments. We therefore have to provide such a predefined predicate and must establish the correct semantics. This behavior may be realized by capturing the five axioms of the equivalence. Reflexivity, Symmetry and Transitivity ensure that the equality predicate possess the algebraic properties of equivalence relation:

$$= (x,x) \leftarrow \quad (6)$$

$$= (x,y) \leftarrow = (y,x) \quad (7)$$

$$= (x,z) \leftarrow = (x,y), = (y,z) \quad (8)$$

The axioms of substitutivity ensures the correct semantics of equivalence when function and predicate symbols, e.g. when

$$= (c,d) \wedge Q(c)$$

holds, then also $Q(d)$ for all predicates Q.

$$= (f(x_1,..,x_n), f(y_1,..,y_n)) \leftarrow \{= (x_i,y_i) | 1 \leqslant i \leqslant n\} \quad (9)$$

$$Q(x_1,..,x_n) \leftarrow Q(y_1,..,y_n) \cup \{= (x_i,y_i) | 1 \leqslant i \leqslant n\} \quad (10)$$

[7] shows that such an axiomatization renders exactly the same semantics as the built-in predicate = in FOL. The reader may note that the axioms of substitutivity have to be instantiated for all predicates Q and functions f used in the rule base.

5. TRANSLATION TO LP

5.1 OWL Lite - RDF Schema Features

This section derives an horn clause axiomatization (suitable for logic programming systems) for the RDF Schema features of OWL Lite [11]. In correspondence to OWL, we focus only on the *domain modelling capabilities* of RDF Schema and, e.g., don't deal with the definitions that create new metaclasses by defining subclasses of `rdfs:Class`.

The two basic modelling primitives provided are *classes* and *properties*. As detailed in section 3, Classes are represented as unary predicates, properties are represented as binary predicates.

The domain modelling part of RDF Schema consists of four additional primitives, namely

- `rdfs:subClassOf`: For each (C `rdfs:subClassOf` D) relationship an implication is generated, making explicit that each instance of C is also an instance of D. The following rule pattern expresses the `rdfs:subClassOf` relationship for two classes C and D.

$$D(X) : -C(X). \quad (11)$$

- `rdfs:subPropertyOf`: For each (M `rdfs:subPropertyOf` N) statement the following rule pattern is instantiated, explicating the subset relationship between the relation N and M.

$$N(X,Y) : -M(X,Y). \quad (12)$$

- `rdfs:domain`: A (P `rdfs:domain` C) statement indicates that the domain of a property P is of a particular class C. Given an property $P(a,b)$, it follows that the a is an instance of class C. The following rules captures the semantics of `rdfs:domain`

$$C(X) : -P(X,Y). \quad (13)$$

- `rdfs:range`: A (P `rdfs:range` C) statement indicates that the range of a property P is of a particular class C. Analogous to `rdfs:domain` the following rules captures the semantics of `rdfs:range`

$$C(Y) : -P(X,Y). \quad (14)$$

5.2 OWL Lite equality and inequality

This section provides the translation for the different means in OWL to define equalities and inequalities for classes, properties and individuals.

5.2.1 Equality of classes and properties

Equality of classes and properties in OWL can be established by different means. One possibility is to establish equality by providing via cyclic `rdfs:subClassOf` and `rdfs:subPropertyOf` definitions. There is not further need to investigate this possibility, since the translation of the `rdfs:subClassOf` and `rdfs:subPropertyOf` definitions are already sufficient to reflect the semantics. The other in OWL and OWL Lite to define equality of classes and properties are the following:

- `owl:sameClassAs`: For each (C `owl:sameClassAs` D) the following rule pattern is instantiated:

$$D(X) : -C(X). \quad (15)$$

$$C(X) : -D(X). \quad (16)$$

The two rules realize equality by establishing a circular subclass definition.

- `owl:samePropertyAs`: For each (M `owl:samePropertyAs` N) the following rule pattern is instantiated:

$$N(X,Y) : -M(X,Y). \quad (17)$$

$$M(X,Y) :- N(X,Y). \quad (18)$$

Analogous to the `owl:sameClassAs` case, the two rules realize equality by establishing a circular subProperty definition.

5.2.2 Disjointness of classes

The pairwise disjointness of two classes can be expressed using the `owl:disjointWith` constructor. It guarantees that there exists no individual that is member of both classes. The following rule pattern captures this:

$$inconsistent(X,X) :- D(X), C(X). \quad (19)$$

5.2.3 Equality and inequality of individuals

Equivalence of individuals is established via the `owl:sameIndividualAs` primitive. For the purpose of readability we will use the symbol $=$ to refer to this primitive, which constitutes an equivalence relation over individuals, such as defined in section 4.4.

To be able to handle skolem functions correctly, additional rules have to be introduced for each skolem function symbol that occurs. For each skolem function symbol f of arity n, a rule as described in section 4.4 needs to be added.

Equivalent individuals are also part of the classes and properties. To handle the extensions correctly the following nodes are required for a classes and properties following the p-substitutivity axiom.

- Class membership:

$$C(X) :- C(Y), = (X,Y). \quad (20)$$

- Property membership:

$$P(X,Z) :- P(X,Y), = (Y,Z). \quad (21)$$

$$P(Y,Z) :- P(X,Z), = (X,Y). \quad (22)$$

The `owl:differentIndividualFrom` primitive is used to denote that two individuals are not the same, hence we use the symbol $\neq$ as a convenient notation. Since OWL may infer that some individuals may be the same, an may inconsistency arises, if `owl:sameIndividualAs` and `owl:differentIndividualFrom` can be derived for the same two individuals. This is captured by the following rule:

$$inconsistent(X,Y) :- \neq (X,Y), = (X,Y). \quad (23)$$

5.3 Property Characteristics

OWL Lite allows to state property characteristics. Syntactically the properties that are subject of those additional characteristics are represented as subclasses of `owl:ObjectProperties`. The class `owl:ObjectProperty` is in turn a subclass of `rdf:Property`. These statements are part of the metalanguage of OWL Lite and treated like `rdf:Property` via binary predicates in the database and unary predicates to capture the property membership.

5.3.1 Unary Property Characteristics

However, depending on if a property P is defined to be transitive or symmetric, further rule patterns are instantiated for p, formalizing the properties of a transitive or symmetric property.

- `owl:TransitiveProperty`:

$$P(X,Z) :- P(X,Y), P(Y,Z). \quad (24)$$

- `owl:SymmetricProperty`:

$$p(X,Y) :- P(Y,X). \quad (25)$$

5.3.2 Binary Property Characteristics

OWL Lite allows to specify that the values of two `owl:ObjectProperties` are inverse to each other, using the (`P owl:inverseOf R`) primitive. Hence the semantics of all (`p owl:inverseOf r`) statements has to be captured via the instantiation of the following rule patterns :

$$R(X,Y) :- P(Y,X) \quad (26)$$

$$P(X,Y) :- R(Y,X) \quad (27)$$

Functional properties are properties that are stated to have a unique value. If a property is a `owl:FunctionalProperty`, then it has no more than one value. It may have no values. Another way of saying this is that the property's minimum cardinality is zero and its maximum cardinality is 1. The consequence is that if two values of p exists they must be identical. This can be represented in Horn clauses as follows:

$$= (X,Y) :- p(A,X), p(A,Y). \quad (28)$$

Hence, it is entailed that X and Y must be equivalent instances. If there is a statement, which declares them to be different from each other, the knowledge base is in an inconsistent state.

A property of type `owl:InverseFunctionalProperty`[6] is a subclass of `owl:ObjectProperty`. If a property is of this type, then the inverse of the property is functional - that means the inverse of the property has at most one value.

This can be represented in Horn clauses by the following rule pattern:

$$= (X,Y) :- P(X,A), P(Y,A) \quad (29)$$

Hence, it is entailed that X and Y are equal. If there is a statement, which declares them to be different from each other, the knowledge base is inconsistent.

5.4 Representation of predefined OWL classes

Thing. The predefined class Thing can be represented by the following rule:

$$Thing(X) :- . \quad (30)$$

Nothing. The predefined class Nothing can be represented by the following rule:

$$Nothing(X) :- \neq (X,X). \quad (31)$$

Nothing per default has an empty extension. Every user defined class subsumes Nothing.

The following rule captures that a consistency follows, if an instance somehow shows up in nothing:

$$inconsistent(X,X) :- Nothing(X). \quad (32)$$

5.5 Class constructors

Class constructors may be nested, hence the translation must be carried out in a recursive manner. The are applied in a recursive manner as described in table 4.1. Whenever a OWL primitive appears in such a nesting, the right hand side of the translation is applied.

This approach is problematic with respect to LP systems, since we cannot control, where such a substitution happens and horn-clauses impose certain restrictions. The following paragraphs will

[6] This type of property was previously called unambiguous property and IsTheOnlyOne property.

explore the behaviour of each OWL class constructor with respect to the limitations of Horn-Clauses and show thereby which constructors can be supported.

5.5.1 owl:hasValue

The owl:hasValue primitive allows to define a class via a certain property value. It can be supported as follows. Let v the value of property P, which constitutes the class C, then the following rule pattern can capture this semantics:

$$C(X) :- P(X, v). \tag{33}$$

$$P(X, v) :- C(X). \tag{34}$$

hasValue does not impose any difficulties wrt. to Horn clauses, since the substitution P(X,v) which is inserted for any definition of C(X) may happen both in the head of a rule (as it occurs when subclassOf is the previously applied translation) and in the body (as it occurs when equivalence is applied).

5.5.2 Cardinality Constraints

OWL means to restrict the cardinality of properties when used on certain classes. The values for such cardinalities are restricted to the values 0 and 1 in the case of OWL Lite. The owl:cardinality construct is a convenience constructor for setting both owl:minCardinalty and owl:maxCardinality to the same value.

Defining a property to have a owl:maximumCardinality of 1 expresses that a property is functional. It is therefore translated to the instantiation of the rule pattern stated for owl:FunctionalProperty (see rule 28).

A owl:Cardinality value of 0 for a property p on class C means that a property may not be instantiated. Hence an inconsistency follows from having any property value on p:

$$inconsistent(X, Y) :- p(X, Y). \tag{35}$$

The class therefore also corresponds to the predefined class Nothing, whose extension is per default empty.

$$Nothing(X) :- C(X). \tag{36}$$

$$C(X) :- Nothing(X). \tag{37}$$

A owl:minimumCardinalty of 1 states that for all domain values of that property there is at least one range value. Expressing the owl:minimumCardinalty constraint in predicate logic results in the following formula:

$$\forall X \exists Y : P(X, Y) \tag{38}$$

The formula is using an existential quantifier, which is not directly expressible using Horn logic. However, using a *skolem function* produces the same effect.

$$p(X, f(X)). \tag{39}$$

Please note that for each constraint a "fresh" skolem function needs to be used, since otherwise unintentional equalities could follow. Hence, this equation may safely occur in both the body and head of rule (which occurs in case of equivalence), when the rule is expanded out during the recursive translation process.

Unrestricted Cardinality. The unrestricted use of cardinality constraints can not be supported efficiently in Logic Programming environments, since it involves counting using inequalities. The basic technique follows from the restricted min- and maxcardinality cases shown above. To support a minimal cardinality n, we must create a new skolem functions for the missing values but take care of all occuring values. To support maxcardinality, we must create new equality assignments if the specified boundary has been broken. this involves to take into consideration all present values. However, we could not yet prove the validity of our idea, since the procedure breaks out of the frame given by [3], who uses counting quantifiers which are not present in logic programming. Hence, we do not support unrestricted cardinality for the time being.

5.5.3 Local range restrictions

A property on a particular class may have a local range restriction associated with it. There are two kinds of local range restrictions:

- owl:allValuesFrom: This means that if an individual instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class.

  May C be the local range restriction on property P for class D. This can be captured via rules of the following form

  $$C(Y) :- P(X, Y), D(X). \tag{40}$$

- owl:someValuesFrom: This means that a particular class may have a restriction on a property that at least one value for that property is of a certain type. It can be captured in first-order logic as follows.

$$\forall X \exists Y : C(Y) \wedge P(X, Y) \leftarrow D(X) \tag{41}$$

$$= \quad \forall X \exists Y : (P(X, Y) \wedge C(Y) \vee \neg D(X)) \tag{42}$$

$$= \quad \forall X \exists Y : (C(Y) \vee \neg D(X)) \wedge (P(X, Y) \vee \neg D(X)) \tag{43}$$

$$= \quad \forall X \exists Y : (C(Y) \leftarrow D(X)) \wedge (P(X, Y) \leftarrow D(X)) \tag{44}$$

The last formula can be skolemized again - this leads to two clauses

$$C(f(X)) :- D(X). \tag{45}$$

$$P(X, f(X)) :- D(X). \tag{46}$$

Please note that again for every owl:someValuesFrom a new skolem function is required.

5.5.4 Set construction of classes

Conjunction. The conjunction of classes ($D \equiv C_1 \sqcap C_2$) can be easily supported easily via the following rule pattern:

$$D(X) :- C_1(X), C_2(X). \tag{47}$$

$$C_1(X) :- D(X). \tag{48}$$

$$C_2(X) :- D(X). \tag{49}$$

Disjunction. Disjunction of classes is problematic since disjunction disjunction in the consequent of the rule, which can not be provided by a Horn clauses, can occur in the case of equivalence ($D \equiv C_1 \sqcup C_2$).

For the $D \sqsupseteq C_1 \sqcup C_2$ direction the following rule pattern is instantiated:

$$D(X) :- C_1(X). \tag{50}$$

$$D(X) :- C_2(X). \tag{51}$$

However the other direction no Horn clause can be stated, since disjunction in the head would occur, such as captured by the following FOL formula:

$$\forall X : C_1(X) \vee C_2(X) \leftarrow D(X) \tag{52}$$

Class Complement. OWL features the complementOf primitive, which cannot be implemented in Horn Logics due to the fact, that there may be no negation in the head, since even the subClassOf substitution for the class construct can not be transformed during the recursive translation. The inability to support equivalence follows directly from this situation.

5.5.5 Construction of classes by enumeration

The `owl:oneOf` primitive can be partially supported. To state the class of the individuals listed in the argument of the `owl:oneOf` primitive, for each member a_i of the primitive a fact is generated:

$$C(a_i). \tag{53}$$

To support the other direction (which states that every instance of C is one of the listed a_i the following formula is required:

$$\forall X\ C(X) \rightarrow = (X, a_1) \vee \cdots \vee = (X, a_n). \tag{54}$$

Unfortunately axiom 54 requires a disjunction in the consequent of the rule, which can not be provided by a Horn clauses.

5.6 Supporting terminological queries

As mentioned in the introduction, DLs support the following set of queries: *(I)* whether a description E subsumes D, this is the case iff for every interpretation $\mathcal{I}$, $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. *(II)* whether a description D is coherent/satisfiable, this is the case if there is at least one $\mathcal{I}$ such that $D^{\mathcal{I}} \neq \emptyset$, and *(III)*whether descriptions E and D are disjoint, this is the case iff for every interpretation $\mathcal{I}$, $D^{\mathcal{I}} \cap E^{\mathcal{I}} = \emptyset$. As those queries are concerned with every possible interpretation $\mathcal{I}$, we may simulate those queries by inserting hyptothetical (unique) inidividuals. Checking if C subsumes D only requires to introduce a new individual i, assert C(i), and see wether D(i) follows. Checking the equivalence, corresponds to doing this simulation in both directions. The classes are disjoint if neither direction follows. Satisfiability of classes is guaranteed since we cannot generate any contradictions with the class constructors expressible in the above frame. The only situation where this could happen, would be if Nothing is involved in the conjunction of classes. However, once any instance of Nothing is created the inconsistency predicate would hold a value. If we apply the above strategy, and the newly inserted instance would show up in the extension of the inconsistent extension, then a class is not satisfable.

6. TRANSLATION TO DATABASES

After having established the OWL/LP fragment in the previous section, we will take a closer look to the fragment of OWL which can be implemented on top of standard relational databases. Luckily, Logic Programming is also an elegant language for data-oriented problems, for example it allows to obtain languages equivalent to known database languages by making various syntactic restrictions. One language that can be obtained by such restrictions is Datalog, which underlies deductive databases. Compared to logic programming Datalog makes the following restrictions.

1. range restriction: all variables in the head of a rule must occur in at least one of the body predicates. This guarantees that rules are strongly safe if the underlying body predicates are safe. A predicate is safe, if it is finite. The range restriction mainly guarantees that queries and rules can be computed in a bottom-up manner, as it is done in databases.
2. function symbols with arity > 0 are excluded.

6.1 Effects of datalog restrictions

Range restriction. With respect to range restrictions, two rules are effected: (Rule 6) the reflexivity of equivalence and (Rule 30) the representation of Thing, which is the top most class. However, we can find equivalent variants of these rules by adding an unary predicate resource(X), which contains all RDF resources and relying on this predicate for the definition of the above rules.

$$= (X, X) : -resource(X). \tag{55}$$

$$Thing(X) : -resource(X). \tag{56}$$

Please note, that this requires to change the RDF representation proposed in section 3 to populate the resource predicate accordingly.

Lack of function symbols. This obviously drops the rule for f-substitutivity. In theory we could simulate skolemization by generation of artificial and unique uris. This requires to implement such a facility, e.g. as a stored procedure in the database. However, this is not a logical characteristic of the Datalog model, hence it will be very difficult to proof the correctness of that approach wrt. to the logic. As a consequence, the existential local range restriction (Rules 45 and 45) , minimum cardinalities (Rule 39) are not supported in our database implementation.

6.2 Translation to relational databases

Datalog programs can be implemented on top of relational databases. To perform this implementation all explicit facts of a predicate p are stored in a dedicated table p_{ext}. All non-recursive rules are translated to relational views. Rule bodies are translated to appropriate SQL queries (usually operating on other views). To obtain all explicit and implicit information, a view is defined to represent each predicate p. The query of the view integrates the explicit information, found in p_{ext} with the queries that represent the bodies of those rules, where p is the head. The interested reader may refer to [17] for an in-depth description, algorithm and proof. Intuitively, this result follows from the following substitutions:

- Each Datalog-rule can be simulated using the select-from-where construct of SQL.
- Multiple rules defining the same predicate can be simulated using *union*.
- Negation in rule bodies can be simulated using *not in*

To compute the answer for user queries the translated views are used. This realizes a form of Bottom up processing, since the queries involved in view definitions are performed on the extensional data and intermediate results are propagated up to a final query, which is the user query. Notably, many irrelevant facts are computed in the intermediate steps, however more efficient procedures based on sideways information passing have been developed in the deductive database literature.

However, the above mentioned strategy is not possible for recursively defined rules. Here additional processing is required.

6.3 Handling recursion

Modern relational database systems, which support the SQL:99 standard, can process some limited form of recursion, namely linear recursion with a path length one. Hence, the predicate used as the rule head may occur only once in the rule body. Cycles other than such linear self-references can also not be implemented.

Usually, binary recursive rules such as transitivity can be rewritten into a linear form. E.g. a transitive predicate like TransitiveProperty (Rule 24) can be rewritten into

$$P(X,Y) :- P_Ext(X,Y). \tag{57}$$

$$P(X,Z) :- P_Ext(X,Y), P(Y,Z). \tag{58}$$

The equality issue. However the rewriting can not be done in our case due to the p-substitutivity axiom of equivalence, where all predicates are already linearly associated with themselves. As mentioned above multiple rules with the same head are simulated by union, hence only one possibility for linear recursion exists and is always taken by the p-substitutivity axiom of equivalence.

The usual strategy to compute the remaining forms of recursive rules in relational databases is in-memory processing using some iterative strategy, e.g. the magic template procedure [13]. However, the constitution of sameIndividualAs in OWL leads to the necessity to always use this strategy for the complete knowledge base. Apparently, this somehow nullifies the use of database query languages as a host language altogether and makes efficient processing of large volumes of instance data questionable.

We therefore decided to drop sameIndividualAs in our database implementation. As a consequence all OWL constructs, where new instance equality is deduced such as Functional- and InverseFunctionalProperties are not supported. Additionally primitives which are logically justified only by the existance of instance equality, such as differentIndividualFrom, are not supported.

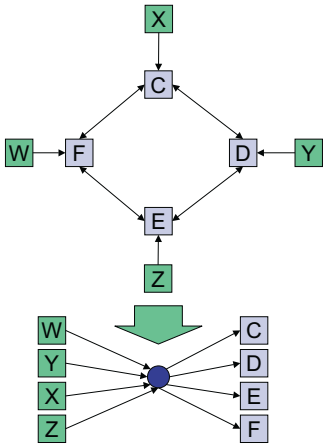


Figure 5: Cyclic Reference Removal

Indirect Recursion. The remaining cases of non-linear recursion that cannot be rewritten into the SQL:99 constructs are mainly represented by the possibility of having cyclic class and property hierarchies. However we can translate this case into the database by exploiting the observation that this form of recursion decomposes into unions, since no join processing of intermediate results such as involved in computing the transitive closure in TransitiveProperty are necessary. This is immediately clear for classes, since they are monadic predicates. A closer look at all axioms where binary predicates (properties) are in the head reveals the same. Hence, these cyclic references can be implemented via an algorithm that detects equivalence classes (each constituted by a cycle) in graphs. All incoming edges to an equivalence class must be duplicated to all members of the equivalence class, such as done in Figure 5. This may done by using a new intermediate predicate to collect the incoming edges and deriving the members of the equivalence class from this intermediate predicate. Afterwards all rules that constitute the cyclic references within the equivalence class may safely be removed. The reader may note that this can also be performed with appropriate adaptions on the cyclic references imposed by inverse properties.

Figure 6: Bubo Online web service

7. IMPLEMENTATION

We have implemented the discussed OWL subsets on top of relational databases and on top of XSB. The implementation is named after the Latin name of the biological genus of eagle owls: bubo. Bubo is a prototypical implementation of the discussed OWL subsets. It is freely available at http://km.aifb.uni-karlsruhe.de/owl/. We also host an online service (cf. Figure 6) that allows to reason over all online OWL ontologies that meet the restrictions discussed in section 4.

Bubo consists of four central components (cf. Figure 7) which are described in the following. Two alternative implementations are provided, one utilizing XSB [15] as a prolog engine, the other DB2 as a SQL:99 compliant database. In this way, implementations of the OWL/LP fragment and the OWL/Datalog fragment are offered.

The **Rule Compiler** generates the appropriate XSB logic program and the respective database view definitions from a given source ontology. All ground facts are stored in separate predicate to allow their combination with the ground facts retrieved from the integrated database content.

The **View Definitions** are specified by the user to map given

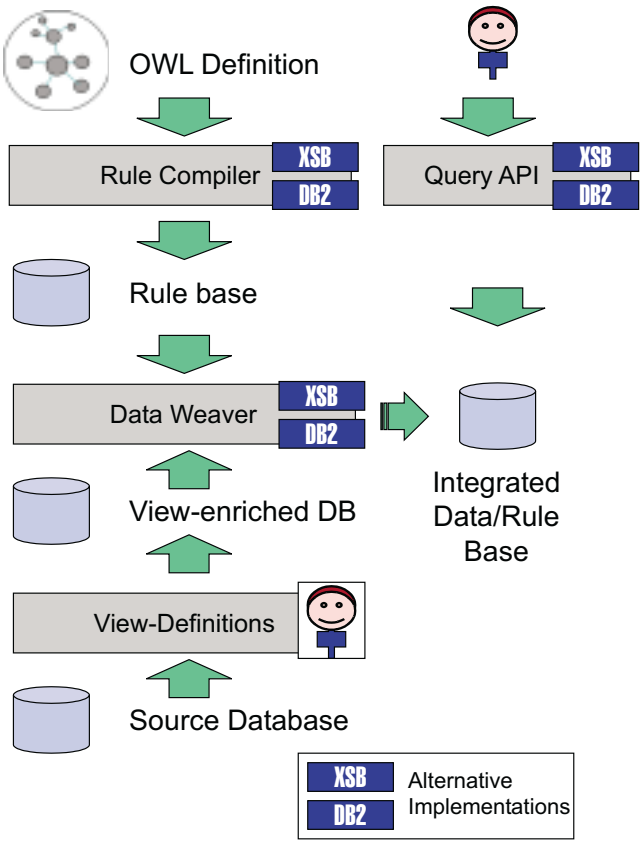


Figure 7: Bubo Architecture

source data to extensions of the ontology. At the moment, two types of views are allowed, binary views to populate properties (cf. Section 3) and unary views to populate classes. The users must follow a defined naming pattern to allow the Data Weaver to recognize the defined views.

The **Data Weaver** merges the instance data defined in the ontology with the data integrated from relational source databases. For XSB this is done via the built-in tuple interface that can be used to retrieve data from databases with the XSB-ODBC module. To do that, the appropriate connections to the source databases are opened (by appending the required statements to the translated rule base). All views defined by the user in the view definition component are bound as XSB predicates using the ODBC import statement:

```
?- odbc_import('PropertyView'('DOMAIN',
                'RANGE'),'PropertyView').
?- odbc_import('ClassView'('INSTANCE'),
                'PropertyView').
```

This data is then combined with the instance data provided in the ontology definition via simple additional rules:

```
P(X,Y) :- P_Onto(X,Y).
P(X,Y) :- P_Db(X,Y).
C(X) :- C_Onto(X).
C(X) :- C_Db(X).
```

The DB2 implementation conceptually follows this approach. However, due to the implementation on the native database, we do not need to import the defined views. The integration of data defined in the ontology and found in the database is done through another view definition:

```
CREATE VIEW P(X,Y) AS
(
(SELECT * FROM P_Onto)
UNION ALL
(SELECT * FROM P_DB)
)
```

The **Query API** provides predefined operations for the standard DL queries, lists available properties and classes and allows read access to data through the query languages offered by XSB and DB2 SQL respectively. All queries operate on the integrated data- and rule-base that is generated by the Data Weaver.

8. RELATED WORK

An axiomatization of DAML+OIL, the precursor of OWL, was given by McGuinness and Fikes [5]. There are a number of consequences of the axiomatization that are not obvious from the language, which is given in KIF and therefore not directly implementable in LP-based systems. It can rather be interpreted by theorem proofers. The axiomatization has not been updated to meet changes made to OWL, for example the definition of SymmetricProperty.

Three systems try to implement OWL using a LP-based approach: The Euler Proof Mechanism [14] by Jos De Roo of Agfa and Tim Berners-Lee's Closed World Machine (CWM) [2] correspond very closely and use the same syntactic format for rules. However both do not consider data integration. They try to axiomatize everything, whereas we try to rely on the features of logic itself, such as implication to operationalize the transitivity of subclassof. Their axiomatization is not proven to be correct or complete, e.g. they do not capture the substituivity of sameIndividualAs and capture only one direction of hasValue. There is no formaly proven characterization of the inference algorithms employed by Euler and CWM. So it is unclear what they are actually doing. On the opposite Horn logic has been well investigated and many optimization methods for evaluation exist in the literature. The DAML XSB compiler [18] derives its axiomatization from [5] and uses XSB to implement DAML+OIL reasoning. Obviously, some KIF rules can not be captured, such as Ax 105 and Ax 128 of [5] The current version is slightly outdated and incomplete, e.g. with respect to equivalence.

Our approach is very close to the work of Grosof and Horrocks, which produced a document called Description Rules in context of the DAML programme. However, the document is still in draft version and presents preliminary findings. They consider an extended logic programming language, with a built-in equivalence predicate and do not discuss how this correlates to existing implementations, which do not sport that feature. Their approach is less complete, but is conceptually similar, since they also follow the translation approach proposed by [3]. However, their translation is less complete, e.g. e.g. cardinality axioms are not suggested. Also, they do not discuss how the approach may be used in conjunction with available database technology.

Last but not least several papers presented a translation of description logics to first-order logic, such as [3] and [16]. Both approaches translate a description logics into FOL, which is not directly implementable in logic-programming environments.

9. CONCLUSION

We have presented the OWL subset which can safely and consistently be represented in logic programming environments. We

have additionally shown, how this translation can be used to derive a translation for relational databases by following the restrictions of the Datalog paradigm with respect to the full Logic Programming paradigm.

Our future work mainly resolves around a more detailed look at cardinality constraints > 1 and coming up with an alternative semantics for the aspects of OWL, which intuitively state constraints, such as cardinality or functionality of properties. We expect that many people coming from a database and object-oriented programming community will find those semantics more natural. We are currently working on trying to find the correspondences between such a constraint semantics and the DL semantics that entails additional equalities to meet specified constraints. Another future topic will be the evaluation and adaption of optimization techniques known for processing logic programs for this special subset of logic programming which centers around unary and binary predicates. We expect that a rule extension in the Semantic Web will largely stay in this framework, hence efficient optimizations for this special type of horn clauses may be achievable.

We also want to investigate how logic programming systems could interact more closely with DL reasoners realizing hybrid reasoning schemes by performing efficient T-Box reasoning on the DL reasoner side and efficient A-Box reasoning on the LP side. Another important topic not investigated yet will be the support for XML datatypes such as recently drafted by the W3C for both RDF and OWL. Obviously, we will again try to reuse as much machinery as possible from the underlying implementations.

Acknowledgements. We thank Ian Horrocks, Boris Motik, and Steffen Staab for their feedback on previous versions of the paper.

10. REFERENCES

[1] Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *The VLDB Journal*, pages 149–158, 2001.

[2] Tim Berners-Lee. Cwm - close world machine. Internet: http://www.w3.org/2000/10/swap/doc/cwm.html, 2002.

[3] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[4] Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[5] R. Fikes and D. McGuiness. An axiomatic semantics for rdf, rdf schema and daml+oil. Technical Report KSL-01-01, KSL, Stanford University, 2001.

[6] Volker Haarslev and Ralf Moller. Description of the RACER system and its applications. In *DL2001 Workshop on Description Logics, Stanford, CA*, 2001.

[7] Steffen Hoelldobler. *Foundations of Equational Logic Programming*, volume 353 of *LNAI*. Springer, 1987.

[8] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705, pages 161–180. Springer-Verlag, 1999.

[9] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. Technical Report TR-90-003, 1, 1990.

[10] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. Internet: http://www.w3.org/TR/REC-rdf-syntax/, 1999.

[11] D. McGuinness and F. van Harmelen. Feature synopsis for owl lite and owl. W3C Working Draft 29 July 2002, Internet: http://www.w3.org/TR/owl-features/, 1999.

[12] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language 1.0 reference. Internet: http://www.w3.org/TR/owl-ref/.

[13] R. RAMAKRISHNAN. Magic templates: A spellbinding approach to logic programs. *J. Logic Programming*, 11:189–216, 1991.

[14] Jos De Roo. Euler proof mechanism. Internet: http://www.agfa.com/w3c/euler/, 2002.

[15] K. Sagonas, T. Swift, and D. S. Warren. Xsb as an efficient deductive database engine. In R. T. Snodgrass and M. Winslett, editors, *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94)*, pages 442–453, 1994.

[16] J. Schmolze and D. Israel. KL-ONE: semantics and classification. Technical Report 5421, BBN Laboratories, 1983.

[17] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume 1. Computer Science Press, 1988.

[18] Youyong Zou. Daml xsb interpretation. Version 0.3, Internet: http://www.cs.umbc.edu/ yzou1/daml/damlxsb.P.txt, January 2001.

APPENDIX

A. OWL/LP REPRESENTATION OF THE GENEALOGY ONTOLOGY

```
Thing(X) :- .
=(X,X) :- .
Animal(X) :- Animal(Y),=(X,Y)
Animal(X) :- Person(X).
Animal(X) :- Male(X).
Animal(X) :- Female(X).
Person(X) :- Person(Y),=(X,Y)
Person(X) :- Man(X).
Person(X) :- Woman(X).
Person(X) :- hasParent(Y,X), Person(Y).
Male(X) :- Male(Y),=(X,Y).
Male(X) :- Man(X).
Female(X) :- Female(Y), =(X,Y).
Female(X) :- Woman(X).
Woman(X) :- Woman(Y), =(X,Y).
Man(X) :- Man(Y), =(X,Y).
inconsistent(X,X) :- Male(X),Female(X).
inconsistent(X,Z) :- inconsistent(X,Y), =(Y,Z).
inconsistent(X,Y) :- =(X,Y), !=(X,Y).
hasParent(X,Y) :- childOf(Y,X).
hasParent(X,Y) :- hasMother(X,Y).
hasParent(X,Y) :- hasFather(X,Y).
hasParent(X,Z) :- hasParent(X,Y), =(Y,Z).
childOf(X,Y) :- hasParent(Y,X).
childOf(X,Z) :- childOf(X,Y), =(Y,Z).
hasMother(X,Z) :- hasMother(X,Y), =(Y,Z).
hasFather(X,Z) :- hasFather(X,Y), =(Y,Z).
```

B. OWL/DB REPRESENTATION OF THE GENEALOGY ONTOLOGY

```
CREATE VIEW Thing(X) AS (SELECT * FROM Resource(X))
CREATE VIEW Man(X) AS (SELECT * FROM Man_Ext)
CREATE VIEW Woman(X) AS (SELECT * FROM Woman_Ext)
CREATE VIEW hasFather(X,Y) AS (SELECT * FROM hasFather_Ext)
```

```
CREATE VIEW hasMother(X,Y) AS (SELECT * FROM hasMother_Ext)
CREATE VIEW hasParent(X,Y) AS
((SELECT * FROM hasParent_Ext)
UNION ALL
(SELECT * FROM hasFather)
UNION ALL
(SELECT * FROM hasMother)
UNION ALL
(SELECT childOf_Ext.Y as X, childOf_Ext.X as Y
 FROM childOf_Ext))
CREATE VIEW childOf(X,Y) AS
((SELECT * FROM childOf_Ext)
UNION ALL
(SELECT hasFather.Y as X, hasFather_Ext.X as Y
 FROM hasFather)
UNION ALL
(SELECT hasMother.Y as X, hasMother.X as Y
FROM hasMother)
UNION ALL
(SELECT hasParent_Ext.Y as X,hasParent_Ext.X as Y
 FROM hasParent_Ext))
CREATE RECURSIVE VIEW Person(X) AS
( ((SELECT * FROM Person_Ext)
  UNION ALL
  (SELECT * FROM Man)
  UNION ALL
  (SELECT * FROM Woman))
  UNION ALL
  (SELECT hasParent.Y as X
   FROM hasParent, Person
   WHERE hasParent.X = Person.X))
CREATE VIEW Male(X) AS (
(SELECT * FROM Male_Ext) UNION ALL (SELECT * FROM Man))
CREATE VIEW Female(X) AS (
(SELECT * FROM Male_Ext) UNION ALL (SELECT * FROM Man))
CREATE VIEW Animal(X) AS
((SELECT * FROM Animal_Ext) UNION ALL (SELECT * FROM Male)
UNION ALL
(SELECT * FROM Female) UNION ALL (SELECT * FROM Person))
```