Dieter Fensel
Katia Sycara
John Mylopoulos (Eds.)

LNCS 2870

The Semantic Web – ISWC 2003

Second International Semantic Web Conference
Sanibel Island, FL, USA October 2003
Proceedings

Springer

Dieter Fensel Katia Sycara
John Mylopoulos (Eds.)

The Semantic Web - ISWC 2003

Second International Semantic Web Conference
Sanibel Island, FL, USA, October 20-23, 2003
Proceedings

Volume Editors

Dieter Fensel
DERI, located at the Universities of Galway and Innsbruck
E-mail: dieter.fensel@uibk.ac.at

Katia Sycara
Carnegie Mellon University, School of Computer Science
Pittsburgh, PA 15213, USA
E-mail: katia@cs.cmu.edu

John Mylopoulos
University of Toronto, Department of Computer Science
40 St. George Street, rm 7266
Toronto, Canada M5S 2E4
E-mail: jm@cs.toronto.edu

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): C.2, H.3, H.4, H.5, F.3, I.2, K.4

ISSN 0302-9743
ISBN 3-540-20362-1 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP Berlin GmbH
Printed on acid-free paper SPIN: 10964071 06/3142 5 4 3 2 1 0

Table of Contents

Foundations

Ontological Reasoning

Semantic Web Services

Security, Trust, and Privacy

Agents and the Semantic Web

Information Retrieval

Multi-media

Tools and Metodologies

Applications

Industrial Track

Representing the UMLS® Semantic Network Using OWL
(Or "What's in a Semantic Web Link?")

Vipul Kashyap[1] and Alex Borgida[2]

[1] LHCNBC, National Library of Medicine,
8600 Rockville Pike, Bethesda, MD 20894
[2] Department of Computer Science, Rutgers University,
New Brunswick, NJ 08903

Abstract. The Semantic Network, a component of the Unified Medical Language System® (UMLS), describes core biomedical knowledge consisting of semantic types and relationships. It is a well established, semi-formal ontology in widespread use for over a decade. We expected to "publish" this ontology on the Semantic Web, using OWL, with relatively little effort. However, we ran into a number of problems concerning alternative interpretations of the SN notation and the inability to express some of the interpretations in OWL. We detail these problems, as a cautionary tale to others planning to publish pre-existing ontologies on the Semantic Web, as a list of issues to consider when describing formally concepts in any ontology, and as a collection of criteria for evaluating alternative representations, which could form part of a methodology of ontology development.

1 Introduction

The Unified Medical Language System® (UMLS®) project was initiated in 1986 by the U.S. National Library of Medicine (NLM). Its goal is to help health professionals and researchers use biomedical information from *a variety of different sources* [1]. The UMLS consists of (i) biomedical concepts and associated strings, comprising the Metathesaurus (*MT*), (ii) a Semantic Network (*SN*) [2], and (iii) a collection of lexical tools (including SPECIALIST lexicon). Both data and tools are integrated in the UMLS Knowledge Source Server[1] and used in a large variety of applications (e.g. PubMed[2], ClinicalTrials.gov[3]). The *MT* provides a common structure for integrating more than 95 source biomedical vocabularies, organized by "concept" (cluster of terms representing the same meaning). The *SN* is a structured description of core biomedical knowledge consisting of semantic types and relationships, used to categorize *MT* concepts, with the *SN* being viewed by some as a semi-formal ontology. It (along with the *MT*) has been in use for more than a decade in the context of information retrieval applications. We expected to "publish" the *SN* on the Semantic Web by expressing it in OWL with relative ease, since there have been lots

---

[1] http://umlsks.nlm.nih.gov
[2] http://www.ncbi.nlm.nih.gov/entrez/query.fcgi
[3] http://www.clinicaltrials.gov

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 1–16, 2003.

- Determining the closest parents and children of a concept in the DAG (for concept translations e.g., [6]).
- Subsumption checking to tighten estimates of semantic distance between concepts, and to limit navigation of the DAG (for concept translations [6], and determination of relevant articles and result ranking for IR.)

We discuss next the "Vanilla" *SN* and its naïve OWL representation. Section 3 then presents various possible interpretations of "links" in the *SN* (prompting the sub-title of the paper as a twist on the famous paper by W. Woods [12]), and the resulting multiple representations. Section 4 discusses possible criteria that might be used to choose between these multiple representations. Conclusions and ongoing/future work are presented in Section 5.

2 The ("Vanilla") Semantic Network and OWL

We start by relating the simple, uncontroversial aspects of the *SN* to the OWL ontology language. The *SN* is a typical semantic network (see Figure 1) with nodes (called "semantic types") and links ("semantic relationships"). The types are organized into two high level hyponym/is-a hierarchies rooted at **`Entity`** and **`Event`**. Intuitively, but not formally, types are organized either by their inherent properties (e.g., a `Mammal` is a `Vertebrate` with constant body temperature) or by certain attributed characteristics (e.g., `ProfessionalGroup` is a set of individuals classified by their vocation). As illustrated in Figure 1, a `MentalBehavioralDysfunction` is a `DiseaseOrSyndrome`, which in turn is a `PathologicFunction`. The relationships used in the SN are also organized in a (shallow) is-a hierarchy, with 5 roots: (a) **`physically_related_to:`** e.g., `part_of`; (b) **`spatially_related_to:`** e.g., `surrounds`, (c) **`temporally_related_to:`** e.g., `precedes`, (d) **`functionally_related_to:`** e.g., `performs`, (e) **`conceptually_related_to:`** e.g., `measures`, `property_of`. For example, the relationship root **`functionally_related_to`** has several is-a children, including `affects` which in turn has many children, including `manages` and `treats`. As is, the relationships in the semantic network are binary.

In order to represent the above on the Semantic Web, RDF Schema (RDFS) would seem to be sufficient. However, RDFS cannot deal with some more advanced aspects of the *SN* to be presented below, and is not equipped to provide the kinds of inferences we had asked for earlier, thus leading us to consider OWL [22]. OWL, based on DAML+OIL [4], is intended to describe the terminology of a domain in terms of *classes/concepts* describing sets of individuals, and *properties/roles* relating these. An ontology consists of a set of *axioms* that assert characteristics of these classes and properties. OWL DL is a kind of *DL* – a logic with clear, formal semantics, (usually corresponding to a subset of First Order Predicate Calculus,) with desirable computational properties (e.g. decidable decision procedures). As in all *DL*, classes can be names (URIs) or *composite expressions*, and a variety of *constructors* are provided for building class expressions. The expressive power of the language is determined by the class (and property) constructors provided, and by the kinds of axioms allowed. Table 1 summarizes these for the underlying OWL. The connection between the *DL* notation and OWL's RDF syntax is shown by the translation of the disjunctive *DL* concept `Bacterium` $\cup$ `Virus`:

```
<owl:Class>
     <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#Bacterium"/>
         <owl:Class rdf:about="#Virus"/>
     </owl:unionOf>
</owl:Class>
```

In a *DL* representation of the UMLS Semantic Network, it is natural to associate *SN* semantic types with *DL* primitive concepts. So, the node `Organism` corresponds to *DL* concept `Organism`, which would be represented in OWL as the class `<owl:Class rdf:ID="Organism"/>`.

An *SN* relationship, such as `process_of`, corresponds to a *DL* primitive role, `process_of`, which would be translated to OWL object properties. In the simplest case, one could associate with a relationship the source and destination of the edge as the "domain" and "range" specification:

```
<owl:ObjectProperty rdf:ID="process_of">
                    <rdfs:domain rdf:resource="#BiologicFunction">
                    <rdfs:range rdf:resource="#Organism">
</owl:ObjectProperty>
```

However, as we shall see in the next section, this translation could be controversial.

Axioms originate from inheritance hierarchies of the various types and relationships. Thus the type/relationship hierarchy in the *SN* can be represented as a collection of subclass/subproperty axioms such as:

`Fungus` $\subseteq$ `Organism` *(sub-types using <owl:subClassOf>)*

`Virus` $\subseteq$ `Organism`

`part_of` $\subseteq$ `physically_related_to` *(sub-relationships using <owl:subPropertyOf>)*

`contains` $\subseteq$ `physically_related_to`

Some relationships in the *SN* have inverses, which is specified through axioms involving the inverseOf role constructor

`part_of` $\equiv$ `has_part`$^-$ *(asymmetric property)*

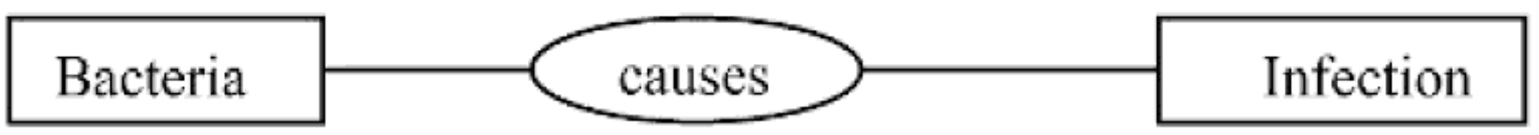
`adjacent_to` $\equiv$ `adjacent_to`$^-$ *(symmetric property)*

3 Semantics of a "Link" in the UMLS Semantic Network

SN types, relationships and their hierarchies, as well as inverses have clear corresponding OWL/DL constructs. However, there are serious difficulties in accurately capturing the semantics of the *SN*, due both to the under-specified meaning of the notion of "link" between two semantic types, and the somewhat unusual inferences/constraints that are associated with them in *SN*. We explore next various possible interpretations of "link", proposing OWL axioms for each, identifying when necessary new *DL* constructs needed. We then evaluate each in light of additional special "inferences" required in the *SN*, such as the notions "domain and range inheritance", "inheritance blocking" and "polymorphic relationships".

3.1 Multiple Interpretations of a "Link"

Consider the following simple diagram, with link labeled "causes" connecting two nodes, Bacteria and Infection:



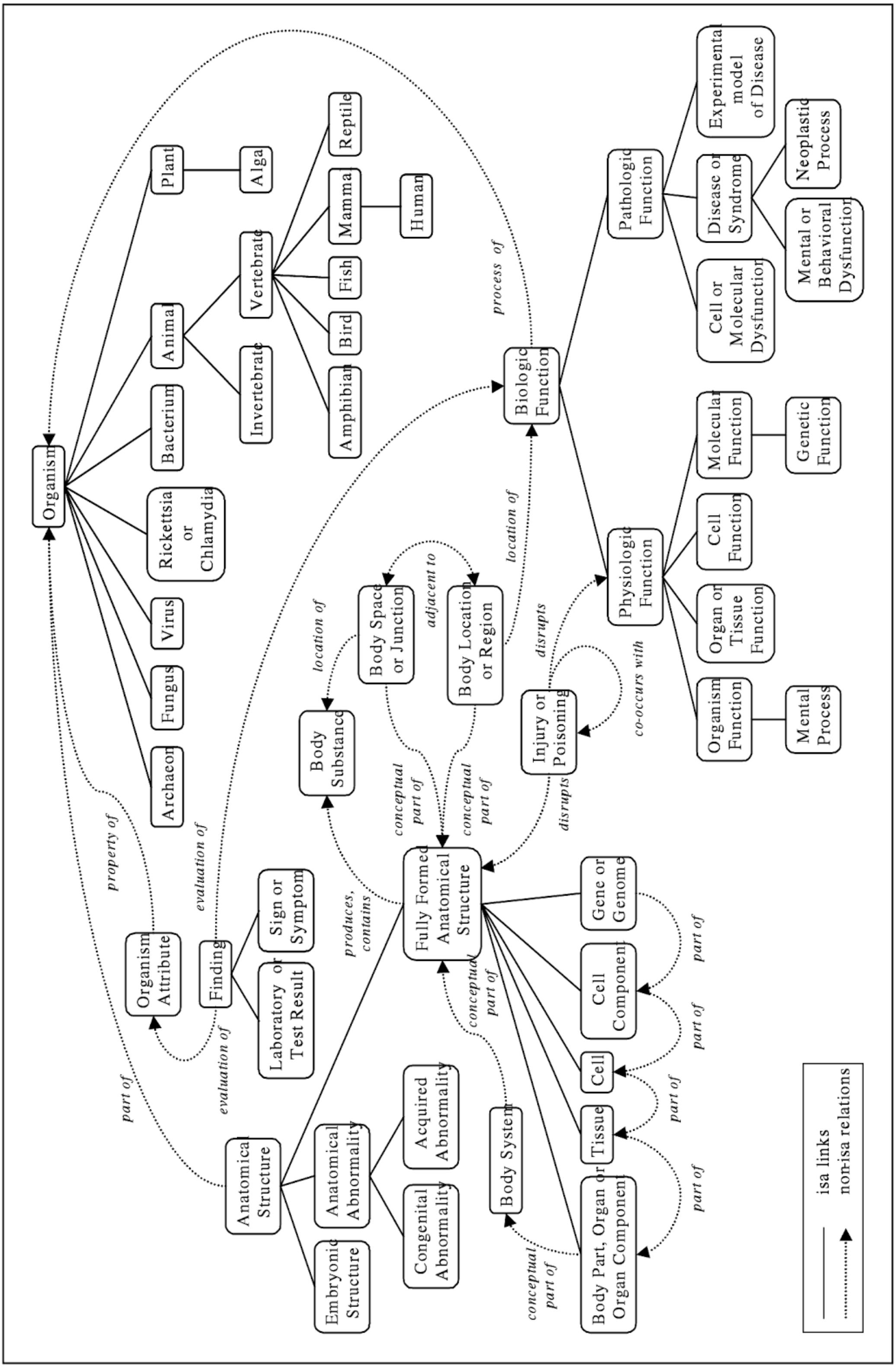


Fig. 1. A portion of the UMLS Semantic Network [23]

Table 1. OWL/RDF constructors and axioms

Constructor/Axiom	DL Syntax	Example
intersectionOf	$C_1 \cap \ldots \cap C_n$	Bacterium ∩ Animal
unionOf	$C_1 \cup \ldots \cup C_n$	Bacterium ∪ Virus
complementOf	$\neg C$	¬Plant
oneOf	$\{ x_1, \ldots, x_n \}$	{aspirin, tylenol}
allValuesFrom	$\forall P.C$	∀partOf.Cell
someValuesFrom	$\exists P.C$	∃processOf.Organism
hasValue	$\exists P.\{x\}$	∃treatedBy{aspirin}
top concept	$\top$	ENTITY
bottom concept	$\bot$	NOTHING
subClassOf	$C_1 \subseteq C_2$	Human ⊆ Animal ∩ Biped
sameClassAs	$C_1 \equiv C_2$	Man ≡ Human ∩ Male
subPropertyOf	$P_1 \subseteq P_2$	part_of ⊆ physically_related_to
samePropertyAs	$P_1 \equiv P_2$	has_temperature ≡ has_fever
disjointWith	$C_1 \subseteq \neg C_2$	Vertebrate ⊆ ¬Invertebrate
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{aspirin}≡{acetyl_salicylic_acid}
differentIndFrom	$\{x_1\} \subseteq \neg\{x_2\}$	{aspirin} ⊆ ¬{tylenol}
inverseOf	$P_1 \equiv P_2^{-}$	has_evaluation ≡ evaluation_of⁻
transitiveProperty	$P^{+} \subseteq P$	part_of⁺ ⊆ part_of
functionalProperty	$\top \subseteq \leq 1\ P$	⊤ ⊆ ≤ 1 has_genetic_profile
inverseFunctionalPropty	$\top \subseteq \leq 1\ P^{-}$	⊤ ⊆ ≤ 1 is_genetic_profile_of⁻
domain	$\exists P.\top \subseteq C$	∃evaluation_of.⊤ ⊆ Finding
range	$\top \subseteq \forall P.C$	⊤ ⊆ ∀evaluation_of.DiagnosticTest

Rector [9] identifies 5 possible interpretations of the above, corresponding to the English statements:

"**All** bacteria cause {**each/only/some**} infection(s)"

"**Some** bacteria cause {**all/some**} infections".

Since semantic web ontology languages emphasize describing relationships in terms of domains and ranges, let us also consider some statements using these notions. We start with defining two operators δ and ρ as follows:

$$\delta(\mathbf{P}) = \{ x \mid (\exists y) P(x, y) \} \text{ and } \rho(\mathbf{P}) = \{ y \mid (\exists x) P(x, y) \}$$

These operators define two sets for presentation purposes and under some interpretations they might correspond to the domain/range interpretations associated with RDFS, and DAML+OIL/OWL. These operators suggest three more interpretations.

"The set of Bacteria {**equals / is contained in/contains**} *δ*(causes)."

"The set of Infections {**equals/ is contained in /contains**} *ρ*(causes)."

Consider now representing each of the above 8 cases using *DLs*. *DL* descriptions can be used to represent **δ(P)** and **ρ(P)** as follows:

$$\delta(\mathbf{P}) \equiv \exists \text{causes}.\top, \quad \rho(\mathbf{P}) \equiv \exists \text{causes}^{-}.\top$$

we have the following axioms for the last three cases above:

- **"δ/ρ equals":**

  axioms: ∃causes.⊤ ≡ Bacteria, ∃causes⁻.⊤ ≡ Infection

- **"δ/ρ subsumed":**

  axioms: ∃causes.⊤ ⊆ Bacteria, ∃causes⁻.⊤ ⊆ Infection

  It may be noted that this corresponds to the domain/range interpretations specified in the RDFS, DAML+OIL/OWL context [24] [25].
- **"δ/ρ subsumes":**

  axioms: Bacteria ⊆ ∃causes.⊤, Infection ⊆ ∃causes⁻.⊤

For the 5 possible statements discussed earlier, we have:

- **"All/some"** (*"All bacteria cause some infection"*) :

  axiom: Bacteria ⊂ ∃causes.Infection
- **"All/only"** *("All bacteria cause only infections.")*:

  axiom: Bacteria ⊆ ∀causes.Infection
- "**All/each**" (*"All bacteria cause each infection.")* This interpretation corresponds to the FOL formula:

  (∀x)(Bacteria(x) ∧ (∀y)(Infection(y) → causes(x,y)))

  ≡ (∀x)(Bacteria(x) ∧ (∀y)(¬causes(x,y) → ¬Infection(y)))

  This can be represented as a subsumption axiom using the role complement operator in *DLs*:

  axiom: Bacteria ⊆ ∀¬causes.¬Infection

  or using the special concept constructor ∀**C.r** ("*objects related by r to all objects in C*"), which has been investigated by Lutz and Sattler [13]:

  Bacteria ⊆ ∀Infection.causes

  In either case, we go beyond the limits of OWL.
- **"Some/some"** *("Some bacteria cause some infections.")* This interpretation can be represented in a number of different ways, though none using axioms of the kinds described in **Table 1**. The alternatives include:

  (i) "There is *some* state of the world where a bacterium causes an infection"

  axiom: Bacteria ⊄ (≤ 0 causes Infection) or

  axiom: "Bacteria ∩ ∃causes.Infection **is consistent"**

  (ii) "A bacterium causes an infection in every possible state of the world"

  axiom: "the concept (Bacteria ∩ ∃causes.Infection) **is never empty"**

  It is the latter which corresponds to the desired logical formula (∃x)(∃y)(Bacteria(x) ∧ Infection(y) ∧ causes(x,y)); it can be expressed using a new kind of axiom, concerning the cardinality of *concepts*[6], which was introduced by Baader et al [8]:

  axiom: ≥ 1 (Bacteria ∩ ∃causes.Infection)
- **"Some/any"** *("Some bacteria cause all infections.")* This requires a combination of the two previous techniques

  axiom: ≥ 1 (Bacteria ∩ ∀¬causes.¬Infection)

A summary of the above interpretations and corresponding encodings may be viewed in Table 2, at the end of this section. We consider next three aspects of the *SN*, some corresponding to inferences and some to constraints, and evaluate the above listed encodings with them in mind.

---

[6] ≥ 1 C) is encoded in OWL by asserting the following axioms: ⊤ ⊆ ∃P.{b} and {b} ⊆ ∃P⁻.C, where b is a new atomic individual and P is a new role.

3.2 δ and ρ Inheritance

The "is-a" link gives rise to "inheritance" relationships, a hallmark of semantic networks. For example, the type `BiologicFunction` has a relationship `process_of` to the type `Organism` in the semantic network (Figure 1) – to be written henceforth as `process_of(BiologicFunction,Organism)`. By inheritance, the descendants of `BiologicFunction` such as `PhysiologicFunction`, `MentalProcess`, etc. are all understood to have the `process_of` relationship to `Organism`. Surprisingly, *SN* also assumes inheritance on the "range" of the relationship, i.e., `process_of(BiologicFunction,Animal)`, `process_of(PhysiologicFunction, Animal)` also hold. An encoding of *SN* will be said to support δ-inheritance if, given (an encoding of) `P(A,B)`, and a concept `C` such that `C` ⊆ `A`, (the encoding of) `P(C,B)` is entailed; and ρ-inheritance is supported if for a `D` such that `D` ⊆ `B`, `P(A,D)` is entailed. Consider now whether/how the encodings discussed in Section 3.1 support δ-inheritance and ρ-inheritance.

- **"δ/ρ equals":** This encoding doesn't support δ-inheritance or ρ-inheritance: from `P(A,B)` we have `δ(P)` ≡ `A`, and if `P(C,B)` were to be true, then `δ(P)` ≡ `C`, which means that `A` must have been equal `C`.
- **"δ/ρ subsumed":** This encoding also doesn't support δ-inheritance or ρ-inheritance, since {`C` ⊆ `A`, `δ(P)` ⊆ `A`} doesn't entail `δ(P)` ⊆ `C`.
- **"δ/ρ subsumes":** Interestingly , this encoding supports both δ-inheritance and ρ-inheritance, because `C` ⊆ `A` ⊆ `δ(P)` entails `C` ⊆ `δ(P)`, and `D` ⊆ `B` ⊆ `ρ(P)` entails `D` ⊆ `ρ(P)`, so that the representation of `P(C,D)` holds.
- **"All/some":** The encoding of `P(A,B)` as `A` ⊆ `∃P.B` supports δ-inheritance, but not ρ-inheritance since from `C` ⊆ `A` and `D` ⊆ `B` we get `C` ⊆ `∃P.B` but not `C` ⊆ `∃P.D`
- **"All/only":** The encoding `A` ⊆`∀P.B` behaves like the previous one, supporting δ-inheritance, but not ρ-inheritance.
- **"All/each":** The encoding `A` ⊆ `∀¬P.¬B` supports δ-inheritance as in the previous cases. It also supports ρ-inheritance since `D` ⊆ `B` entails `∀¬P.¬B` ⊆ `∀¬P.¬D`, so that `A` ⊆ `∀¬P.¬D` holds.
- **"Some/some":** The encoding `(≥ 1 (A ∩ ∃P.B))` doesn't support δ-inheritance or ρ-inheritance because the addition of `C` ⊆ `A` and `D` ⊆ `B` doesn't entail either `(≥ 1 (C ∩ ∃P.B))` or `(≥ 1 (A ∩ ∃P.D))`.
- **"Some/all":** The encoding `(≥ 1 (A ∩ ∀¬P.¬B))` doesn't support δ-inheritance, but supports ρ-inheritance: Since `D` ⊆ `B` entails `∀¬P.¬B` ⊆ `∀¬P.¬D`, we can infer `(A ∩ ∀¬P.¬B)` ⊆ `(A ∩ ∀¬P.¬D)`, and hence `(≥ 1 (A ∩ ∀¬P.¬D))` holds if `(≥ 1 (A ∩ ∀¬P.¬B))` holds.

In general, if an encoding does not support some form of inheritance, we will need to explicitly assert axioms corresponding to the missed inheritance inferences.

3.3 Inheritance Blocking

In some cases there will be a conflict between the placement of types in the *SN* and the links to be inherited. For example, `process_of(MentalProcess,Plant)` is

inheritance – it is called "subtype polymorphism" in Programming Languages. However, in UMLS SN, the same relationship can be stated to connect pairs of types that are not is-a related. For example, in Figure 1, we have, among others

```
location_of(BodySpaceorJunction,BodySubstance)
location_of(BodyLocationOrRegion,BiologicFunction)
contained_in(BodySubstance,EmbryonicStructure)
contained_in(BodySubstance,FullyFormedAnatomicalStructure)
```

Such examples, with edges $P(A_1, B_1)$ and $P(A_2, B_2)$, exhibit what might be called "ad-hoc polymorphism/overloading" in Object Oriented languages. One could interpret this to be equivalent to the introduction of two new relationships, P_1 and P_2, adding the axiom $P \equiv P_1 \cup P_2$, and modeling $P(A_1, B_1)$ and $P(A_2, B_2)$ with $P_1(A_1, B_1)$ and $P_2(A_2, B_2)$. Unfortunately, OWL does not support property union; and even if it did, the above encoding is ***non-incremental*** in the sense that we must detect cases of overloading, and *remove* earlier axioms, replacing them with new ones. Such non-locality seems unavoidable for blocking, which is inherently non-monotonic, but is otherwise undesirable since it makes it difficult to maintain sets of axioms.

Some intuitions related to polymorphism which may be useful to evaluate alternatives are: (i) When $A_1 \cap A_2$ and $B_1 \cap B_2$ are empty, as in the first pair above, the constraints should not affect each other. (ii) When $A_1 \equiv A_2 \equiv A$, as in the second example pair above, the encoding should not require R to be the empty relation if $B1 \cap B2 \equiv \emptyset$. Consider again the encodings from Section 3.1.

- **"δ/ρ subsumed":** From $\{\delta(P) \subseteq A_1, \delta(P) \subseteq A_2, \rho(P) \subseteq B_1, \rho(P) \subseteq B_2\}$ we get $\{\delta(P) \subseteq A_1 \cap A_2, \rho(P) \subseteq B_1 \cap B_2\}$, which means that case (i) above is miss-handled. It may be noted that this case corresponds to the RDFS, DAML+OIL/OWL interpretation of multiple ranges and domains [24][25].
- **"δ/ρ subsumes":** While the above intuitions are satisfied, $\{A_1 \subseteq \delta(P), A_2 \subseteq \delta(P), B_1 \subseteq \rho(P), B_2 \subseteq \rho(P)\}$ entails $\{A_1 \cup A_2 \subseteq \delta(P), B_1 \cup B_2 \subseteq \rho(P)\}$, so it seems we get $P(A_1 \cup A_2, B_1 \cup B_2)$ from $P(A_1,B_1)$ and $P(A_2,B_2)$. This is overly permissive, since it gives rise to *unintended* models, when $(x,y) \in P$, $x \in A_1$, $y \in B_2$ or $x \in A_2$, $y \in B_1$.

The previous two cases are ones where using sub-properties is appropriate.

- **"All/only":** The encoding $\{A_1 \subseteq \forall P.B_1, A_2 \subseteq \forall P. B_2\}$ supports polymorphism properly in the case when the A's are disjoint, but in case (ii) above we get $A \subseteq \forall P.(B_1 \cap B_2) \equiv \forall P.\bot \equiv (\leq 0\ P)$, which does not allow A to be related to anything via P, thus contradicting our intuitions for (ii). Therefore, we must replace the original axioms $\{A_1 \subseteq \forall P.B_1, A_2 \subseteq \forall P.B_2\}$ by a new set $\{(A_1 \cap \neg A_2) \subseteq \forall P.B_1, (\neg A_1 \cap A_2) \subseteq \forall P.B_2, (A_1 \cap A_2) \subseteq \forall P.(B_1 \cup B_2)\}$, another case of non-incrementality.
- **All/each, all/some, some/each, some/some**: All these encodings support polymorphism following an analysis similar to the previous case.

3.5 Summary

The various encoding schemes and their suitability for the three special aspects of *SN*, viz. domain and range inheritance, inheritance blocking and polymorphic relationships are summarized in Table 2.

Table 2. Interpretation, axioms and support for *SN* requirements

Interpretation	Encoding	δ/ρ Inheritance	Inheritance Blocking	Polymorphic Relations
δ/ρ equals	$\delta(P) \equiv A$ $\rho(P) \equiv B$	No/No	N/A	No
δ/ρ subsumed	$\delta(P) \subseteq A$ $\rho(P) \subseteq B$	No/No	N/A	Missed model
δ/ρ subsumes	$A \subseteq \delta(P)$ $B \subseteq \rho(P)$	Yes/Yes	Exceptions + compensation	Unintended model
all / some	$A \subseteq \exists P.B$	Yes/No	Exception in axiom	ok
all / only	$A \subseteq \forall P.B$	Yes/No	Exception in axiom	Modification
some / some	$\geq 1(A \cap \exists P.B)$	No/No	N/A	ok
some / all	$\geq 1(A \cap \forall \neg P.\neg B)$	No/Yes	Exception in axiom	ok
all / each	$A \subseteq \forall \neg P.\neg B$	Yes/Yes	Exceptions + compensation	ok

4 First Steps towards a Representation Choice Methodology

In the previous section we considered a list of alternative encodings of the *SN* into *DL*. We now propose an (incomplete) set of questions that could guide ontology developers in making choices among alternative representations in a formal ontology language such as OWL; these might form the basis of a methodological framework. The questions fall into several categories:

(a) Does the encoding support the "inferences" of the original notation?
(b) Does the encoding support inferences needed by expected applications?
(c) Does the encoding provide a reasonable intuitive model of the domain?
(d) Is the encoding supported by the formal ontology language and its reasoner?

Let us examine the alternatives from Section 3 in this regard as a way of illustrating and adding details to the list given below.

(a) Support for Inferences of the Notation

After identifying a number of possible representations for the node+link notation of *SN*, in Section 3, we looked to see which provided a logically consistent mechanism for performing inferences *explicitly, though informally, sanctioned by the notation*. Surprisingly, it appears that for *SN* the "**all/each"** encoding most closely captures these intuitions, with "**δ/ρ subsumes"** as the next best encoding. Several other aspects need to be considered, when dealing with graphical notations:

Does an encoding entail unintended inferences?

The "some/some" statement has the effect of "upwardly" inheriting a link to all the superclasses of the nodes associated with the link. This requires the ontology developer to identify such situations and assert axioms to prevent them.

Can/should something be inferred from the absence of a link?

It is not clear in *SN* whether links should be read as type constraints in programming languages: what is not explicitly permitted is forbidden. If so, the encoding $A \subseteq \forall P.B$

doesn't prohibit instances of ¬A being related to B. To prevent this, we would have to add the axiom (¬A ⊆ ≤ 0 P).

Should relationships be inferred to be asymmetric by default?
This is a special case of the previous general "default negation". According to [2], in the *SN*, links are "*usually asymmetric – MedicalDevice prevents PathologicFunction but not vice versa*"; moreover one can specify it: adjacent_to ≡ adjacent_to⁻. Axioms to this effect can be added automatically during translation, although asymmetricity again requires non-standard axioms: ¬(P ≡ P-).

Are the is-a children of nodes disjoint?
In the *SN*, there are some examples where this is not the case. Horrocks and Rector [17] give good arguments that the proper way to model ontologies is to start with trees of disjoint primitive concepts, and *define* everything else in terms of them.

(b) Support for Intended Application
If it is important to detect inconsistency in an ontology without overloading [11] e.g., {hasGender(FATHER,{male}), hasGender(FATHER,{female})}, where the relationship hasGender relates a concept FATHER to concepts represented as enumerations, e.g. {male}, {female}. An encoding of the form A ⊆ ∀P.B will not infer inconsistency, unless one also adds A ⊆ ∃P.T. Alternately, if the application uses only a limited set of inferences (e.g., because the form of the questions asked is limited), then one may not need to represent difficult kinds of axioms (e.g., properties being asymmetric). Such criteria can be criticized on the grounds that an ontology is supposed to be "application neutral", although there is always some arbitrary decision to be made about what is included and what is not.

(c) Reasonableness of the domain model
A strong case can be made that one should start first with an idea of how the world should be *modeled* – what are individuals, properties, etc, in the domain of discourse, and how concepts related to them, tied to the denotational semantics of the formal notation. The standard interpretation of a concept is a set of individual objects in the world, connected by properties (DL denotational semantics). Thus, causes (Bacteria, Infection) constrains the way in which any particular individual bacterium can be related by "causes" to a case of infection. So the questions are:

What are the intuitive encodings?
The "**all/some**" encoding Bacteria ⊆ ∃cause.Infection seems to be the representation of choice in several sophisticated medical ontologies developed with DL-knowledgeable collaborators [10][11], although in any state of the world some bacteria may not cause any infection. On the other hand, the "**all/only**" encoding Bacteria ⊆ ∀cause.Infection, is most frequently used in object-centered representations (e.g., [16]), though it runs into problems with polymorphism, since some bacteria might cause rashes and infections. Either way, interpretations such as "*all bacteria cause all infections*", or "*there is a bacterium that causes some infection*" seem quite odd, and were in fact rejected out of hand in [9].

Is an alternative interpretation possible?
The above seems rather discouraging for the **"all/each"** encoding, even though it satisfies all the *SN* requirements. However, suppose we prefix the relationship names by *"can/could_have/could_be"*: *"a bacterium could be the cause of any case of infection*". The resulting reading is much more reasonable, and may explain the inheritance and polymorphism properties of the *SN*, especially when noting that most relationships in the *SN* (unlike `fatherOf`, say) can be read this way.

The **"some/some"** reading – "some bacterium causes some infection", also seems to be unnatural, but interestingly McCray and Nelson explicitly endorse it: "**a relation is only established if at least some members of a set map onto at least one member of another set linked by the relationship**"[2]. The explanation for this may be that in the medical informatics community, concepts such as `INFECTION` are viewed by many as having kinds of infections, rather than specific cases of those infections, as instances. This is hinted at by the UMLS terminology of "labeling" concepts in the MT by the semantic types in the SN, rather than saying that the MT concepts are subclasses of semantic types, which form a high-level ontology. We note that this approach has been successful in the context of medical research literature search/retrieval, and propose this as an interesting topic of future research in the OWL/RDF context.

(d) Representation and inference in ontology language
Can the desired encoding be expressed in the ontology language of choice?
Given that the Semantic Web appears to be settling on a common ontology language (OWL) it is clearly important to encode the axioms in the constructs of this language. In this respect the representations based on role operators (negation, disjunction) and concept cardinality appear to be unsatisfactory. (But see below.)

Can the interpretation be represented using less "expensive" terms?
In addition to addressing the previous issue, reformulating an encoding using different constructors/axioms could provide significant computational benefits in view of the wealth of information about the computational complexity of various collections of DL concept and role constructors (e.g., see [7]) .

Is there some (better) way to capture the desired encoding knowing the technology used to implement the reasoner for the ontology language?
We have seen that one encoding of the some/some interpretation uses a language extension proposed by Baader et al. [8], who used esoteric techniques (based on "Hintikka trees" and automata), to reason about theses. There is an alternate encoding using nominals (individuals), and we saw that for the case of $(\geq 1\ C)$, there is even a solution in the basic ALC DL. Similarly, the axiom $\top \subseteq \forall P.C$ can be reasoned withmuch more efficiently by some tableaux reasoners than the logically equivalent $\exists P^{-}.\top \subseteq C$

Are there acceptable approximations to concepts/axioms?
Another approach to deal with the limited expressiveness of the ontology language, or the complexity of reasoning, is by representing concepts and/or axioms in an *approximate* way; this requires understanding what kinds of questions applications

need to have answered, in order to evaluate the price of information loss. Consider the axiom $P \equiv P_1 \cup P_2$, used in property polymorphisms. To avoid property union, not available in OWL, we can assert $P_1 \subseteq P$ and $P_2 \subseteq P$. Approximation of disjunction using hierarchies has been considered in [18], and there has been considerable research on approximation in *DL*s [19].

5 Conclusions and Future Work

We have reviewed in this paper some of our experiences with representing in OWL, a well established semi-formal ontology, the UMLS Semantic Network, which has been in use for over a decade in the context of medical informatics. Whereas the representation of the "vanilla" *SN* was straightforward, we encountered obstacles representing the semantics of "links" in the *SN*, especially in the context of requirements such as δ/ρ inheritance, inheritance blocking and polymorphism. This led us to investigate the possible interpretations and encodings of a "link" in the *SN*. We did not use role transitivity and number restrictions, but did use class disjunctions, role hierarchies, axioms, inverses, all and some property restrictions. The encodings were evaluated based on their support to *SN* requirements above. The various issues enumerated in this context should be considered by ontology and content developers while formally describing concepts in an ontology. Among the criteria we have identified are (i) support for inferences desired, (ii) intuitiveness of the resulting denotational semantic model, (iii) representation and effective reasoning in the ontology language. The latter involves formal worst case complexity results about the cost of reasoning, direct exploitation of the reasoner technology underlying the ontology language, and the possibility of approximate representation. We hope that the parts of a methodology provided above will be helpful to ontology developers that have embarked on the task of expressing their ontologies using OWL. We observe that although some of our problems could have been avoided if *SN* would have had a formal semantics. However, even starting with a language with equivalent translations to OWL, is not enough, since questions related to expressibility and intended modeling semantics, among others, still remain.

At the NLM, we are exploring various research issues related to the Semantic Web [3], both in the context of enhancing existing biomedical applications and for enabling new applications. Some ongoing investigations are: (a) The Semantic Vocabulary Interoperation project [21], which aims to provide tools and techniques to translate a term in a source biomedical vocabulary (e.g., ICD-9-CM) to a target biomedical vocabulary (e.g., SNOMED) by using the knowledge present in the *SN* and *MT*; (b) Potential improvement for searching and retrieving text and citation information by annotation of biomedical content using semantic web markup languages such as RDF and OWL; and (c) Enhancement and consistency maintenance of biomedical vocabularies based on reasoning with OWL descriptions as proposed in [5].

Acknowledgements. We would like to acknowledge helpful discussions with Alexa McCray, Olivier Bodenreider and Sergio Tessari. Support for this work was provided by the Oak Ridge Institute for Science and Education (ORISE).

References

[1] Lindberg D, Humphreys B, McCray A. "The Unified Medical Language System." Methods Inf Med 1993:32(4):281–91.

[2] McCray A, Nelson S. "The representation of meaning in the UMLS." Methods Inf Med 1995:34(1–2):193–201

[3] Berners-Lee T, Hendler J, Lassila O. "The Semantic Web." Scientific American, May 2001. http://www.sciam.com/2001/0501issue/0501berners-lee.html

[4] Horrocks I. "DAML+OIL: A Description Logic for the Semantic Web." IEEE Bulletin for the Technical Committee on Data Engineering, 2002.

[5] Stevens R., Goble R., Horrocks I. and Bechhofer S. "Building a Bioinformatics Ontology using OIL." IEEE Information Technology in Biomedicine, special issue on Bioinformatics, Vol 6(2):135–141, June 2002.

[6] Mena E, Kashyap V, Illarramendi A and Sheth A. "Imprecise answers in a Distributed Environment: Estimation of Information Loss for Multiple Ontology-based Query Processing." Int. J. of Cooperative Information Systems (IJCIS), 9(4), December 2000.

[7] "The Description Logic Handbook: Theory, Implementation and Applications." F. Baader, D. Calvanese, D. McGuiness , D. Nardi and P F Patel-Schneider (editors), Cambridge University Press, 2003.

[8] Baader F, Buchheit F, and Hollunder B. "Cardinality Restrictions on Concepts." Artificial Intelligence, 88(1–2):195–213, 1996.

[9] Rector A L, Rogers J and colleagues. "Introduction to Ontologies." Tutorial presented at the AMIA Annual Symposium 2002.

[10] Rector A L, Bechhofer S K, Goble C A, Horrocks I, Nowlan W A, Solomon W D. "The GRAIL Concept Modelling Language for Medical Terminology." Artificial Intelligence in Medicine, Volume 9, 1997

[11] Gangemi A, Pisanelli D M and Steve G. "An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies." Data and Knowledge Engineering, 31(2):183–220, 1999.

[12] Woods W A. "What's in a link: Foundations for Semantic Networks." In R. J. Brachman and H J Levesque (editors), Readings in Knowledge Representation, 218–241. Morgan Kaufman Publishers, 1985

[13] Lutz C and Sattler U. "Mary likes all Cats." Proc. 2000 Int. Workshop on Description Logics (DL 2000).

[14] http://phd1.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRSTR.daml

[15] Horrocks, I. "The FaCT system." In Proc. Int. Conf. Tableaux '98, Springer-Verlag LNCS 1397, pp. 307–312, May 1998.

[16] Calvanese D, Lenzerini M and Nardi D. "Unifying Class-Based Representation Formalisms." Journal of Artificial Intelligence Research (JAIR) 11: 199–240 (1999)

[17] Horrocks I., and Rector A. "Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions." Proc. Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97).

[18] Borgida, A., and Etherington, D.W. "Hierarchical Knowledge Bases and Efficient Disjunctive Reasoning." Proc. KR'89, pp. 33–43

[19] Brandt S., Küsters R., and Turhan A.-Y. "Approximating ALCN-Concept Descriptions." In Proc. of the 2002 Int. Workshop on Description Logics (DL 2002).

[20] Medical Subject Headings – Home Page, http://www.nlm.nih.gov/mesh/meshhome.html

[21] The Semantic Vocabulary Interoperation Project, http://cgsb2.nlm.nih.gov/~kashyap/projects/SVIP.html

[22] OWL Web Ontology Language Guide, http://www.w3.org/TR/2003/WD-owl-guide-20030331/

Reducing OWL Entailment to Description Logic Satisfiability

Ian Horrocks[1] and Peter F. Patel-Schneider[2]

[1] Department of Computer Science
University of Manchester
horrocks@cs.man.ac.uk
[2] Bell Labs Research
Lucent Technologies
pfps@research.bell-labs.com

Abstract. We show how to reduce ontology entailment for the OWL DL and OWL Lite ontology languages to knowledge base satisfiability in (respectively) the $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$ description logics. This is done by first establishing a correspondence between OWL ontologies and description logic knowledge bases and then by showing how knowledge base entailment can be reduced to knowledge base satisfiability.

1 Introduction

The aim of the Semantic Web is to make web resources (not just HTML pages, but a wide range of web accessible data and services) more readily accessible to automated processes. This is to be done by augmenting existing *presentation* markup with *semantic* markup, i.e., meta-data annotations that describe their content [2]. According to widely known proposals for a Semantic Web architecture, ontologies will play a key role as they will be used as a source of shared and precisely defined terms that can be used in such metadata [15].

The importance of ontologies in semantic markup has prompted the development of several ontology languages specifically designed for this purpose. These include OIL [7], DAML+OIL [10] and OWL [4]. OWL is of particular significance as it has been developed by the W3C Web Ontology working group, and is set to become a W3C recommendation.

The proposed OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. Like OWL's predecessor DAML+OIL, OWL Lite and OWL DL are basically very expressive description logics with an RDF syntax. They can therefore exploit the considerable existing body of description logic research, e.g., to define the semantics of the language and to understand its formal properties, in particular the decidability and complexity of key inference problems [6]. OWL Full provides a more complete integration with RDF, but its formal properties are less well understood, and key inference problems would certainly be

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 17–29, 2003.

much harder to compute.[1] In this paper we will, therefore, concentrate on the provision of reasoning services for OWL Lite and OWL DL.

1.1 OWL Reasoning

Reasoning with ontology languages will be important in the Semantic Web if applications are to exploit the semantics of ontology based metadata annotations, e.g., if semantic search engines are to find pages based on the semantics of their annotations rather than their syntax. As well as providing insights into OWL's formal properties, OWL's relationship to expressive description logics provides a source of algorithms for solving key inference problems, in particular satisfiability. Moreover, in spite of the high worst case complexity of reasoning in such description logics, highly optimised implementations of these algorithms are available and have been shown to work well with realistic problems. Two difficulties arise, however, when attempting to use such implementations to provide reasoning services for OWL:

1. OWL's RDF syntax uses frame-like constructs that do not correspond directly to description logic axioms; and
2. as in RDF, OWL inference is defined in terms of ontology entailment rather than ontology satisfiability.

The obvious solution to the first problem is to define a mapping that decomposes OWL frames into one or more description logic axioms. It turns out, however, that the RDF syntax used in OWL cannot be directly translated into any "standard" description logic because it allows the use of anonymous individuals in axioms asserting the types of and relationships between individuals. The obvious solution to the second problem is to reduce entailment to satisfiability. Doing this naively would, however, require role negation, and this is not supported in any implemented description logic reasoner.

In this paper we will show that, in spite of these difficulties, ontology entailment in OWL DL and OWL Lite can be reduced to knowledge base satisfiability in the $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$ description logics respectively. This is achieved by mapping OWL to an intermediate description logic that includes a novel axiom asserting the non-emptiness of a class, and by using a more sophisticated reduction to satisfiability that both eliminates this constructor and avoids the use of role negation.

This is a significant result from both a theoretical and a practical perspective: it demonstrates that computing ontology entailment in OWL DL (resp. OWL Lite) has the same complexity as computing knowledge base satisfiability in $\mathcal{SHOIN}(\mathbf{D})$ ($\mathcal{SHIF}(\mathbf{D})$), and that description logic algorithms and implementations (such as RACER [8]) can be used to provide reasoning services for OWL Lite. Unfortunately, the design of "practical" algorithms for $\mathcal{SHOIN}(\mathbf{D})$ is still an open problem – the search for such algorithms must obviously be a high priority within the Semantic Web research community.

---

[1] Inference in OWL Full is clearly undecidable as OWL Full does not include restrictions on the use of transitive properties which are required in order to maintain decidability [13].

2 The OWL Web Ontology Language

As mentioned in Section 1, OWL [4] is an ontology language that has recently been developed by the W3C Web Ontology Working Group. OWL is defined as an extension to RDF in the form of a vocabulary entailment [9], i.e., the syntax of OWL is the syntax of RDF and the semantics of OWL are an extension of the semantics of RDF.

OWL has many features in common with description logics, but also has some significant differences. The first difference between OWL and description logics is that the syntax of OWL is the syntax of RDF. OWL information is thus encoded in RDF/XML documents [1] and parsed into RDF Graphs [14] composed of triples. Because RDF Graphs are such an impoverished syntax, many description logic constructs in OWL are encoded into several triples. Because RDF Graphs are graphs, however, it is possible to create circular syntactic structures in OWL, which are not possible in description logics. Subtle interactions between OWL and RDF cause problems with some of these circular syntactic structures.

The second difference between OWL and description logics is that OWL contains features that do not fit within the description logic framework. For example, OWL classes are objects in the domain of discourse and can be made instances of other concepts, including themselves. These two features, also present in RDF, make a semantic treatment of OWL quite different from the semantic treatment of description logics.

2.1 OWL DL and OWL Lite

Fortunately for our purpose, there are officially-defined subsets of OWL that are much closer to description logics. The larger of these subsets, called OWL DL, restricts OWL in two ways. First, unusual syntactic constructs, such as descriptions with syntactic cycles in them, are not allowed in OWL DL. Second, classes, properties, and individuals (usually called concepts, roles and individuals in description logics) must be disjoint in the semantics for OWL DL.

Because of the syntactic restrictions in OWL DL, it is possible to develop an abstract syntax for OWL DL [16] that looks much like an abstract syntax for a powerful frame language, and is not very different from description logic syntaxes. This is very similar to the approach taken in the OIL language [7]. The abstract syntax for OWL DL has classes and data ranges, which are analogues of concepts and concrete datatypes in description logics, and axioms and facts, which are analogues of axioms in description logics. Axioms and facts are grouped into ontologies, the analogue of description logic knowledge bases, which are the highest level of OWL DL syntax.

The constructors used to form OWL DL descriptions and data ranges are summarised in Figure 1, where A is a class name, C (possibly subscripted) is a description, o (possibly subscripted) is an individual name, R is an object (or abstract) property, T is a datatype property,[2] B is a datatype, D is a data range, v (possibly subscripted) is a data value and ℓ, m, n are non-negative integers; elements {enclosed in braces} can be repeated zero or more times and elements [enclosed in square brackets] are optional. The details of these constructors can be found in the OWL documentation [4].

[2] An object property is one that associates pairs of individuals; a datatype property associates an individual with a data value.

Classes
A
intersectionOf(C_1 ... C_n)
unionOf(C_1 ... C_n)
complementOf(C)
oneOf(o_1 ... o_n)
restriction(R {allValuesFrom(C)} {someValuesFrom(C)} {value(o)} [minCardinality(n)] [maxCardinality(m)] [cardinality(ℓ)])
restriction(T {allValuesFrom(D)} {someValuesFrom(D)} {value(v)} [minCardinality(n)] [maxCardinality(m)] [cardinality(ℓ)])
Data Ranges
B
oneOf(v_1 ... v_n)

Fig. 1. OWL DL Constructors

Descriptions and data ranges can be used in OWL DL axioms and facts to provide information about classes, properties, and individuals. Figure 2 provides a summary of these axioms and facts. The details of these constructors can also be found in the OWL documentation [4]. In particular, Figure 2 ignores annotations and deprecation, which allow uninterpreted information to be associated with classes and properties, but which are not interesting from a logical point of view.

Because of the semantic restrictions in OWL DL, metaclasses and other notions that do not fit into the description logic semantic framework can be ignored. In fact, OWL DL has a semantics that is very much in the description logic style, and that has been shown to be equivalent to the RDF-style semantics for all of OWL [16]. Again, we will not present all of this semantics, instead concentrating on its differences from the usual description logics semantics.

There is a subset of OWL DL, called OWL Lite, the motivation for which is increased ease of implementation. This is achieved by supporting fewer constructors than OWL DL, and by limiting the use of some of these constructors. In particular, OWL Lite does not support the `oneOf` constuctor (equivalent to description logic *nominals*), as this constructor is known to increase theoretical complexity and to lead to difficulties in the design of practical algorithms [11]. In Section 5 we will examine these differences in more detail, and explore their impact on the reduction from OWL entailment to description logic satisfiability.

2.2 Semantics for OWL DL

The semantics for OWL DL is fairly standard by description logic standards. The OWL semantic domain is a set whose elements can be divided into abstract objects (the abstract domain), and datatype values (the datatype or concrete domain, written $\Delta^{\mathcal{I}}_{\mathbf{D}}$). Datatypes

Class Axioms
Class(A partial $d_1 \ldots d_n$)
Class(A complete $d_1 \ldots d_n$)
EnumeratedClass(A $o_1 \ldots o_n$)
DisjointClasses($d_1 \ldots d_n$)
EquivalentClasses($d_1 \ldots d_n$)
SubClassOf(d_1 d_2)
Property Axioms
DatatypeProperty(U super(U_1) $\ldots$ super(U_n) [Functional]
domain(d_1) $\ldots$ domain(d_m) range(r_1) $\ldots$ domain(r_l))
ObjectProperty(P super(P_1) $\ldots$ super(P_n) [inverseOf(P_0)]
[Functional] [InverseFunctional] [Symmetric] [Transitive]
domain(d_1) $\ldots$ domain(d_m) range(e_1) $\ldots$ domain(e_l))
EquivalentProperties($U_1 \ldots U_n$)
SubPropertyOf(U_1 U_2)
EquivalentProperties($P_1 \ldots P_n$)
SubPropertyOf(P_1 P_2)
Facts
Individual([o] type(d_1) $\ldots$ type(d_1)
value(p_1 v_1) $\ldots$ value(p_1 v_1))
SameIndividual($o_1 \ldots o_n$)
DifferentIndividuals($o_1 \ldots o_n$)

Fig. 2. OWL DL Axioms and Facts (simplified)

in OWL are derived from the built-in XML Schema datatypes [3]. Datatype values are denoted by special literal constructs in the syntax, the details of which need not concern us here.

An interpretation in this semantics is officially a four-tuple consisting of the abstract domain and separate mappings for concept names, property names, and individual names (in description logics, the mappings are usually combined to give a two-tuple, but the two forms are obviously equivalent). OWL DL classes are interpreted as subsets of the abstract domain, and for each constructor the semantics of the resulting class is defined in terms of its components. For example, given two classes C and D, the interpretation of the intersection of C and D is defined to be the intersection of the interpretations of C and D. Datatypes are handled by means of a mapping $.^{\mathbf{D}}$ that interprets datatype names as subsets of the concrete domain and data names (i.e., lexical representations of data values) as elements of the concrete domain.

OWL DL axioms and facts result in semantic conditions on interpretations. For example, an axiom asserting that C is a subclass of D results in the semantic condition that the interpretation of C must be a subset of the interpretation of D, while a fact asserting that a has type C results in the semantic condition that the interpretation of a must be an element of the set that is the interpretation of C. An OWL DL ontology O is satisfied by an interpretation $\mathcal{I}$ just when all of the semantic conditions resulting from the axioms and facts in $\mathcal{O}$ are satisfied by $\mathcal{I}$.

The main semantic relationship in OWL DL is entailment – a relationship between pairs of OWL ontologies. An ontology O_1 entails an ontology O_2, written $O_1 \models O_2$,

Constructor Name	Syntax	Semantics
atomic concept **A**	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
datatypes **D**	D	$D^{\mathbf{D}} \subseteq \Delta^{\mathcal{I}}_{\mathbf{D}}$
abstract role $\mathbf{R}_A$	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
datatype role $\mathbf{R}_{\mathbf{D}}$	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{\mathbf{D}}$
individuals **I**	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
data values	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$
inverse role	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
conjunction	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
negation	$\neg C_1$	$(\neg C_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$
oneOf	$\{o_1, \ldots\}$	$\{o_1, \ldots\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots\}$
exists restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y. \langle x \rangle y \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
atleast restriction	$\geqslant n\, R$	$(\geqslant n\, R)^{\mathcal{I}} = \{x \mid \sharp(\{y. \langle x, y \rangle \in R^{\mathcal{I}}\}) \geqslant n\}$
atmost restriction	$\leqslant n\, R$	$(\leqslant n\, R)^{\mathcal{I}} = \{x \mid \sharp(\{y. \langle x, y \rangle \in R^{\mathcal{I}}\}) \leqslant n\}$
datatype exists	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y. \langle x, y \rangle \in U^{\mathcal{I}}$ and $y \in D^{\mathbf{D}}\}$
datatype value	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y. \langle x, y \rangle \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$
datatype atleast	$\geqslant n\, U$	$(\geqslant n\, U)^{\mathcal{I}} = \{x \mid \sharp(\{y. \langle x, y \rangle \in U^{\mathcal{I}}\}) \geqslant n\}$
datatype atmost	$\leqslant n\, U$	$(\leqslant n\, U)^{\mathcal{I}} = \{x \mid \sharp(\{y. \langle x, y \rangle \in U^{\mathcal{I}}\}) \leqslant n\}$
datatype oneOf	$\{v_1, \ldots\}$	$\{v_1, \ldots\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \ldots\}$
Axiom Name	**Syntax**	**Semantics**
concept inclusion	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
object role inclusion	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
object role transitivity	$\mathsf{Trans}(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
datatype role inclusion	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
individual inclusion	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
individual equality	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
concept existence	$\exists C$	$\sharp(C^{\mathcal{I}}) \geqslant 1$

Fig. 3. Syntax and semantics of $\mathcal{SHOIN}^+(\mathbf{D})$

exactly when all interpretations that satisfy O_1 also satisfy O_2. This semantic relationship is different from the standard description logic relationships, such as knowledge base and concept satisfiability. The main goal of this paper is to show how OWL DL entailment can be transformed into DL knowledge base (un)satisfiability.

3 $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$

The main description logic that we will be using in this paper is $\mathcal{SHOIN}(\mathbf{D})$, which is similar to the well known $\mathcal{SHOQ}(\mathbf{D})$ description logic [11], but is extended with inverse roles ($\mathcal{I}$) and restricted to unqualified number restrictions ($\mathcal{N}$). We will assume throughout the paper that datatypes and data values are as in OWL.

Let $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R_D}$, and $\mathbf{I}$ be pairwise disjoint sets of *concept names*, *abstract role names*, *datatype (or concrete) role names*, and *individual names*. The set of $\mathcal{SHOIN}(\mathbf{D})$-roles is $\mathbf{R}_A \cup \{R^- \mid R \in \mathbf{R}_A\} \cup \mathbf{R_D}$. In order to avoid considering roles such as R^{--} we will define $\mathsf{Inv}(R)$ s.t. $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$. The set of $\mathcal{SHOIN}(\mathbf{D})$-concepts is the smallest set that can be built using the constructors in Figure 3.

The $\mathcal{SHOIN}(\mathbf{D})$ axiom syntax is also given in Figure 3. (The last axiom in Figure 3 forms an extension of $\mathcal{SHOIN}(\mathbf{D})$, which we call $\mathcal{SHOIN}^+(\mathbf{D})$, which is used internally in our translation.) A *knowledge base* $\mathcal{K}$ is a finite set of axioms. We will use $\sqsubseteq^*$ to denote the transitive reflexive closure of $\sqsubseteq$ on roles, i.e., for two roles S, R in $\mathcal{K}$, $S \sqsubseteq^* R$ in $\mathcal{K}$ if $S = R$, $S \sqsubseteq R \in \mathcal{K}$, $\mathsf{Inv}(S) \sqsubseteq \mathsf{Inv}(R) \in \mathcal{K}$, or there exists some role Q such that $S \sqsubseteq^* Q$ in $\mathcal{K}$ and $Q \sqsubseteq^* R$ in $\mathcal{K}$. A role R is called *simple* in $\mathcal{K}$ if for each role S s.t. $S \sqsubseteq^* R$ in $\mathcal{K}$, $\mathsf{Trans}(S) \notin \mathcal{K}$ and $\mathsf{Trans}(\mathsf{Inv}(S)) \notin \mathcal{K}$. To maintain decidability, a knowledge base must have no number restrictions on non-simple roles [13].

The semantics of $\mathcal{SHOIN}^+(\mathbf{D})$ is given by means of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain $\Delta^{\mathcal{I}}$, disjoint from the datatype (or concrete) domain $\Delta^{\mathcal{I}}_{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$, which interprets atomic and complex concepts, roles, and nominals according to Figure 3. (In Figure 3, $\sharp$ is set cardinality.)

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* a $\mathcal{SHOIN}^+(\mathbf{D})$-axiom under the conditions given in Figure 3. An interpretation satisfies a knowledge base $\mathcal{K}$ iff it satisfies each axiom in $\mathcal{K}$; $\mathcal{K}$ is *satisfiable* (*unsatisfiable*) iff there exists (does not exist) such an interpretation. A $\mathcal{SHOIN}(\mathbf{D})$-concept C is satisfiable w.r.t. a knowledge base $\mathcal{K}$ iff there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$ that satisfies $\mathcal{K}$. A concept C is *subsumed* by a concept D w.r.t. $\mathcal{K}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each interpretation $\mathcal{I}$ satisfying $\mathcal{K}$. Two concepts are said to be equivalent w.r.t. $\mathcal{K}$ iff they subsume each other w.r.t. $\mathcal{K}$. A knowledge base $\mathcal{K}_1$ entails a knowledge base $\mathcal{K}_2$ iff every interpretation of $\mathcal{K}_1$ is also an interpretation of $\mathcal{K}_2$.

We define a notion of entailment in $\mathcal{SHOIN}^+(\mathbf{D})$ in the same way as it was defined for OWL DL. It is easy to show that $\mathcal{K} \models \mathcal{K}'$ iff $\mathcal{K} \models A$ for every axiom A in $\mathcal{K}'$.

The description logic $\mathcal{SHIF}(\mathbf{D})$ is just $\mathcal{SHOIN}(\mathbf{D})$ without the oneOf constructor and with the atleast and atmost constructors limited to 0 and 1. $\mathcal{SHIF}^+(\mathbf{D})$ is related to $\mathcal{SHOIN}^+(\mathbf{D})$ in the same way.

4 From OWL DL Entailment to $\mathcal{SHOIN}(\mathbf{D})$ Unsatisfiability

We will now show how to translate OWL DL entailment into $\mathcal{SHOIN}(\mathbf{D})$ unsatisfiability. The first step of our process is to translate an entailment between OWL DL ontologies into an entailment between knowledge bases in $\mathcal{SHOIN}^+(\mathbf{D})$. Then $\mathcal{SHOIN}^+(\mathbf{D})$ entailment is transformed into unsatisfiability of $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases. (Note that concept existence axioms are eliminated in this last step, leaving a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base.)

OWL fragment F	Translation $\mathcal{F}(F)$
Individual($x_1 \ldots x_n$)	$\exists(\mathcal{F}(x_1) \sqcap \ldots \sqcap \mathcal{F}(x_n))$
type(C)	$\mathcal{V}(C)$
value($R\ x$)	$\exists R.\mathcal{F}(x)$
value($U\ v$)	$\exists U.\{v\}$
o	$\{o\}$

Fig. 4. Translation from OWL facts to $\mathcal{SHOIN}^+(\mathbf{D})$

4.1 From OWL DL to $\mathcal{SHOIN}(\mathbf{D})$

An OWL DL ontology is translated into a $\mathcal{SHOIN}^+(\mathbf{D})$ knowledge base by taking each axiom and fact in the ontology and translating it into one or more axioms in the knowledge base. For OWL DL axioms, this translation is very natural, and is almost identical to the translation of OIL described by Decker et al. [5]. For example, the OWL DL axiom Class(A complete $C_1 \ldots C_n$) is translated into the pair of $\mathcal{SHOIN}^+(\mathbf{D})$ axioms $A \sqsubseteq \mathcal{V}(C_1) \sqcap \ldots \sqcap \mathcal{V}(C_n)$ and $\mathcal{V}(C_1) \sqcap \ldots \sqcap \mathcal{V}(C_n) \sqsubseteq A$, where $\mathcal{V}$ is the obvious translation from OWL classes to description logic concepts, again very similar to the transformation described by Decker et al. [5]. Similarly, an OWL DL axiom `DisjointClasses`($C_1 \ldots C_n$) is translated into the $\mathcal{SHOIN}^+(\mathbf{D})$ axioms $\mathcal{V}(C_i) \sqsubseteq \neg\mathcal{V}(C_j)$ for $1 \leq i < j \leq n$.

The translation of OWL DL facts to $\mathcal{SHOIN}^+(\mathbf{D})$ axioms is more complex. This is because facts can be stated with respect to anonymous individuals, and can include relationships to other (possibly anonymous) individuals. For example, the fact Individual(type(C) value(R Individual(type(D)))) states that there exists an individual that is an instance of class C and is related via the property R to an individual that is an instance of the class D, without naming either of the individuals.

The need to translate this kind of fact is the reason for introducing the $\mathcal{SHOIN}^+(\mathbf{D})$ existence axiom. For example, the above fact can be translated into the axiom $\exists(C \sqcap \exists R.D)$, which states that there exists some instance of the concept $C \sqcap \exists R.D$, i.e., an individual that is an instance of C and is related via the role R to an instance of the concept D. Figure 4 describes a translation $\mathcal{F}$ that transforms OWL facts into a $\mathcal{SHOIN}^+(\mathbf{D})$ existence axioms, where C is an OWL class, R is an OWL abstract property or $\mathcal{SHOIN}^+(\mathbf{D})$ abstract role, U is an OWL datatype property or $\mathcal{SHOIN}^+(\mathbf{D})$ datatype role, o is an individual name, v is a data value, and $\mathcal{V}$ is the above mentioned translation from OWL classes to $\mathcal{SHOIN}^+(\mathbf{D})$ concepts.

Theorem 1. *The translation from OWL DL to $\mathcal{SHOIN}^+(\mathbf{D})$ preserves equivalence. That is, an OWL DL axiom or fact is satisfied by an interpretation $\mathcal{I}$ if and only if the translation is satisfied by $\mathcal{I}$.*[3]

The above translation increases the size of an ontology to at most the square of its size. It can easily be performed in time linear in the size of the resultant knowledge base.

[3] The statement of the theorem here ignores the minor differences between OWL DL interpretations and $\mathcal{SHOIN}^+(\mathbf{D})$ interpretations. A stricter account would have to worry about these stylistic differences.

Axiom A	Transformation $\mathcal{G}(A)$
$c \sqsubseteq d$	$x : c \sqcap \neg d$
$\exists c$	$\top \sqsubseteq \neg c$
$\mathsf{Trans}(r)$	$x : \exists r.\exists r.\{y\} \sqcap \neg \exists r.\{y\}$
$r \sqsubseteq s$	$x : \exists r.\{y\} \sqcap \neg \exists s.\{y\}$
$f \sqsubseteq g$	$x : \bigsqcup_{z \in \mathbf{V}} \exists f.\{z\} \sqcap \neg \exists g.\{z\}$ for $\mathbf{V}$ = the set of data values in $\mathcal{K}$, plus one fresh data value for each datatype in $\mathcal{K}$
$a = b$	$a \neq b$
$a \neq b$	$a = b$

Fig. 5. Translation from Entailment to Unsatisfiability

4.2 From Entailment to Unsatisfiability

The next step of our process is to transform $\mathcal{SHOIN}^+(\mathbf{D})$ knowledge base entailment to $\mathcal{SHOIN}(\mathbf{D})$ knowledge base unsatisfiability. We do this to relate our new notion of description logic entailment to the well-known operation of description logic knowledge base unsatisfiability.

We recall from Section 3 that $\mathcal{K} \models \mathcal{K}'$ iff $\mathcal{K} \models A$ for every axiom A in $\mathcal{K}'$. We therefore define (in Figure 5) a translation, $\mathcal{G}$, such that $\mathcal{K} \models A$ iff $\mathcal{K} \cup \{\mathcal{G}(A)\}$ is unsatisfiable, for $\mathcal{K}$ a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base and A a $\mathcal{SHOIN}(\mathbf{D})$ axiom. In this transformation we have need of names of various sorts that do not occur in the knowledge base or axiom; following standard practice we will call these fresh names. Throughout the translation, x and y are fresh individual names.

Most of the translations in $\mathcal{G}$ are quite standard and simple. For example, an object role inclusion axiom $r \sqsubseteq s$ is translated into an axiom that requires the existence of an individual that is related to some other individual by r but not by s, thus violating the axiom. The only unusual translation is for datatype role inclusions $f \sqsubseteq g$. Because data values have a known "identity" (rather like individuals under the unique name assumption), a fresh value cannot be used to simulate an existentially quantified variable that could be interpreted as any element in the datatype domain (in the way the fresh nominal is used in the case of an object role inclusion axiom). Instead, it is necessary to show that the relevant inclusion holds for every data value that occurs in the knowledge base, plus one fresh data value (i.e., one that does not occur in the knowledge base) for each datatype in $\mathcal{K}$. Because there are no operations on data values, it suffices to consider only these fresh data values in addition to those that occur in the knowledge base.

The translation $\mathcal{G}$ increases the size of an axiom to at most the larger of its size and the size of the knowledge base. It can easily be performed in time linear in the larger of the size of the axiom and the size of the knowledge base. (If datatype role inclusions are not used, then $\mathcal{G}$ increases the size of an axiom by atmost a constant amount.)

The translation $\mathcal{G}$ eliminates concept existence axioms from the knowledge base $\mathcal{K}'$ on the right-hand side of the entailment. Our last step is to eliminate concept existance axioms from the knowledge base $\mathcal{K}$ on the left-hand side of the entailment. We do this by applying a translation $\mathcal{E}(\mathcal{K})$ that replaces each axiom of the form $\exists C \in \mathcal{K}$ with an axiom $a : C$, for a a fresh individual name. It is obvious that this translation preserves

OWL fragment F	Translation $\mathcal{F}'(F)$
Individual($x_1 \ldots x_n$)	$\mathcal{F}'(a : x_1), \ldots, \mathcal{F}'(a : x_n)$ for a a fresh individual name
a : type(C)	$a : \mathcal{V}(C)$
a : value($R\ x$)	$\langle a, b \rangle : R, \mathcal{F}'(b : x)$ for b a fresh individual name
a : value($U\ v$)	$\langle a, v \rangle : U$
$a : o$	$a = o$

Fig. 6. Translation from OWL Lite facts to $\mathcal{SHIF}^+(\mathbf{D})$

satisfiability, can be easily performed, and only increases the size of a knowledge base by a linear amount.

Theorem 2. *Let $\mathcal{K}$ and $\mathcal{K}'$ be $\mathcal{SHOIN}^+(\mathbf{D})$ knowledge bases. Then $\mathcal{K} \models \mathcal{K}'$ iff the $\mathcal{SHOIN}(\mathbf{D})$ knowledge base $\mathcal{E}(\mathcal{K}) \cup \{\mathcal{G}(\mathcal{A})\}$ is unsatisfiable for every axiom A in $\mathcal{K}'$.*

4.3 Consequences

The overall translation from OWL DL entailment to $\mathcal{SHOIN}(\mathbf{D})$ can be performed in polynomial time and results in a polynomial number of knowledge base satisfiability problems each of which is polynomial in the size of the initial OWL DL entailment. Therefore we have shown that OWL DL entailment is in the same complexity class as knowledge base satisfiability in $\mathcal{SHOIN}(\mathbf{D})$.

Unfortunately, $\mathcal{SHOIN}(\mathbf{D})$ is a difficult description logic. Most problems in $\mathcal{SHOIN}(\mathbf{D})$, including knowledge base satisfiability, are in NExpTime [17]. Further, there are as yet no known optimized inference algorithms or implemented systems for $\mathcal{SHOIN}(\mathbf{D})$. The situation is not, however, completely bleak. There is an inexact translation from $\mathcal{SHOIN}(\mathbf{D})$ to $\mathcal{SHIN}(\mathbf{D})$ that turns nominals into atomic concept names. This translation could be used to produce a partial, but still very capable, reasoner for OWL DL. Moreover, as is shown in the next section, the situation for OWL Lite is significantly different.

5 Transforming OWL Lite

OWL Lite is the subset of OWL DL that

1. eliminates the `intersectionOf`, `unionOf`, `complementOf`, and `oneOf` constructors;
2. removes the `value` construct from the `restriction` constructors;
3. limits cardinalities to 0 and 1;
4. eliminates the `enumeratedClass` axiom; and
5. requires that description-forming constructors not occur in other description-forming constructors.

Axiom A	Transformation $\mathcal{G}(A)$
$a : C$	$a : \neg C$
$\langle a, b\rangle : R$	$b : B, a : \forall R.\neg B$ for B a fresh concept name
$\langle a, v\rangle : U$	$a : \forall U.\overline{v}$

Fig. 7. Extended Transformation from Entailment to Unsatisfiability

The reason for defining the OWL Lite subset of OWL DL was to have an easier target for implementation. This was thought to be mostly easier parsing and other syntactic manipulations.

As OWL Lite does not have the analogue of nominals it is possible that inference is easier in OWL Lite than in OWL DL. However, the transformation above from OWL DL entailment into $\mathcal{SHOIN}(\mathbf{D})$ unsatisfiability uses nominals even for OWL Lite constructs. It is thus worthwhile to devise an alternative translation that avoids nominals.

There are three places that nominals show up in our transformation:

1. translations into $\mathcal{SHOIN}^+(\mathbf{D})$ of OWL DL constructs that are not in OWL Lite, in particular the `oneOf` constructor;
2. translations into $\mathcal{SHOIN}^+(\mathbf{D})$ axioms of OWL DL `Individual` facts; and
3. the transformation to $\mathcal{SHOIN}(\mathbf{D})$ unsatisfiability of $\mathcal{SHOIN}^+(\mathbf{D})$ entailments whose consequents are role inclusion axioms or role transitivity axioms.

The first of these, of course, is not a concern when considering OWL Lite.

The second place where nominals show up is in the translation of OWL `Individual` facts into $\mathcal{SHOIN}(\mathbf{D})$ axioms (Figure 4). In order to avoid introducing nominals, we can use the alternative transformation $\mathcal{F}'$ given in Figure 6. Note that, in this case, the translation $\mathcal{V}(C)$ does not introduce any nominals as we are translating OWL Lite classes.

The new transformation does, however, introduce axioms of the form $a : C$, $\langle a, b\rangle : R$ and $\langle a, v\rangle : U$ that we will need to deal with when transforming from entailment to satisfiability. We can do this by extending the transformation $\mathcal{G}$ given in Figure 5 as shown in Figure 7. The extension deals with axioms of the form $\langle a, b\rangle : R$ using a simple transformation, described in more detail by [12], and with axioms of the form $\langle a, v\rangle : U$ using a datatype derived from the negation of a data value (written $\overline{v}$).

The third and final place where nominals show up is in the transformation of entailments whose consequents are object role inclusion axioms or role transitivity axioms.

Object role inclusion axioms can be dealt with using a transformation similar to those given in Figure 7 (and described in more detail in [12]), which does not introduce any nominals. This is shown in the following lemma:

Lemma 1. *Let K be an OWL Lite ontology and let A be an OWL Lite role inclusion axiom stating that r is a subrole of s. Then $K \models A$ iff $\mathcal{E}(K) \cup \{x : B \sqcap \exists r(\forall s^{-}.\neg B)\}$ is unsatisfiable for x a fresh individual name, and B a fresh concept name.*

Transitivity axioms can be dealt with by exploiting the more limited expressive power of OWL Lite, in particular its inability to describe classes, datatypes or properties whose interpretations must be non-empty but finite (e.g., classes described using the `oneOf` constructor). As a result of this more limited expressive power, the only way to deduce the transitivity of a property r is to show that the interpretation of r cannot form any chains (i.e., consists only of isolated tuples, or is empty). This observation leads to the following lemma:

Lemma 2. *Let K be an OWL Lite ontology and let A be an OWL Lite role transitivity axiom stating that r is transitive. Then $K \models A$ iff $\mathcal{E}(K) \cup \{x : \exists r(\exists r \top)\}$ is unsatisfiable for x a fresh individual name (i.e., r forms no chains).*

The above lemmas, taken together, show that OWL Lite entailment can be transformed into knowledge base unsatisfiability in $\mathcal{SHIF}(\mathbf{D})$, plus some simple (and easy) tests on the syntactic form of a knowledge base. A simple examination shows that the transformations can be computed in polynomial time and result in only a linear increase in size.

As knowledge base satisfiability in $\mathcal{SHIF}(\mathbf{D})$ is in ExpTime [17] this means that entailment in OWL Lite can be computed in exponential time. Further, OWL Lite entailment can be computed by the RACER description logic system [8], a heavily-optimised description logic reasoner, resulting in an effective reasoner for OWL Lite entailment.

6 Conclusion

Reasoning with ontology languages will be important in the Semantic Web if applications are to exploit the semantics of ontology based metadata annotations. We have shown that ontology entailment in the OWL DL and OWL Lite ontology languages can be reduced to knowledge base satisfiability in, respectively, the $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$ description logics. This is so even though some constructs in these languages go beyond the standard description logic constructs.

From these mappings, we have determined that the complexity of ontology entailment in OWL DL and OWL Lite is in NExpTime and ExpTime respectively (the same as for knowledge base satisfiability in $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$ respectively). The mapping of OWL Lite to $\mathcal{SHIF}(\mathbf{D})$ also means that already-known practical reasoning algorithms for $\mathcal{SHIF}(\mathbf{D})$ can be used to determine ontology entailment in OWL Lite; in particular, the highly optimised RACER system [8], which can determine knowledge base satisfaction in $\mathcal{SHIF}(\mathbf{D})$, can be used to provide efficient reasoning services for OWL Lite.

The mapping from OWL DL to $\mathcal{SHOIN}(\mathbf{D})$ can also be used to provide complete reasoning services for a large part of OWL DL, or partial reasoning services for all of OWL DL. In spite of its known decidability, however, the design of "practical" decision procedures for $\mathcal{SHOIN}(\mathbf{D})$ is still an open problem. The search for such algorithms must obviously be a high priority within the Semantic Web research community.

References

[1] D. Beckett. RDF/XML syntax specification (revised). W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-rdf-syntax-grammar-20030123`.

[2] T. Berners-Lee. *Weaving the Web*. Harpur, San Francisco, 1999.

[3] P. V. Biron and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, 2001. Available at `http://www.w3.org/TR/2003/WD-xmlschema-2-20010502/`.

[4] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Web ontology language (OWL) reference version 1.0. W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-owl-ref-20030331`.

[5] S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, S. Melnik, M. Klein, and J. Broekstra. Knowledge representation on the web. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, pages 89–98, 2000.

[6] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997.

[7] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[8] V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[9] P. Hayes. RDF semantics. W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-rdf-mt-20030123`.

[10] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, 2002.

[11] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$ description logic. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[12] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2000.

[13] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[14] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-rdf-concepts-20030123`.

[15] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3C Recommendation, 1999. Available at `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`.

[16] P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. Web ontology language (OWL) abstract syntax and semantics. W3C Working Draft, 2003. Available at `http://www.w3.org/TR/2003/WD-owl-ref-20030331`.

[17] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

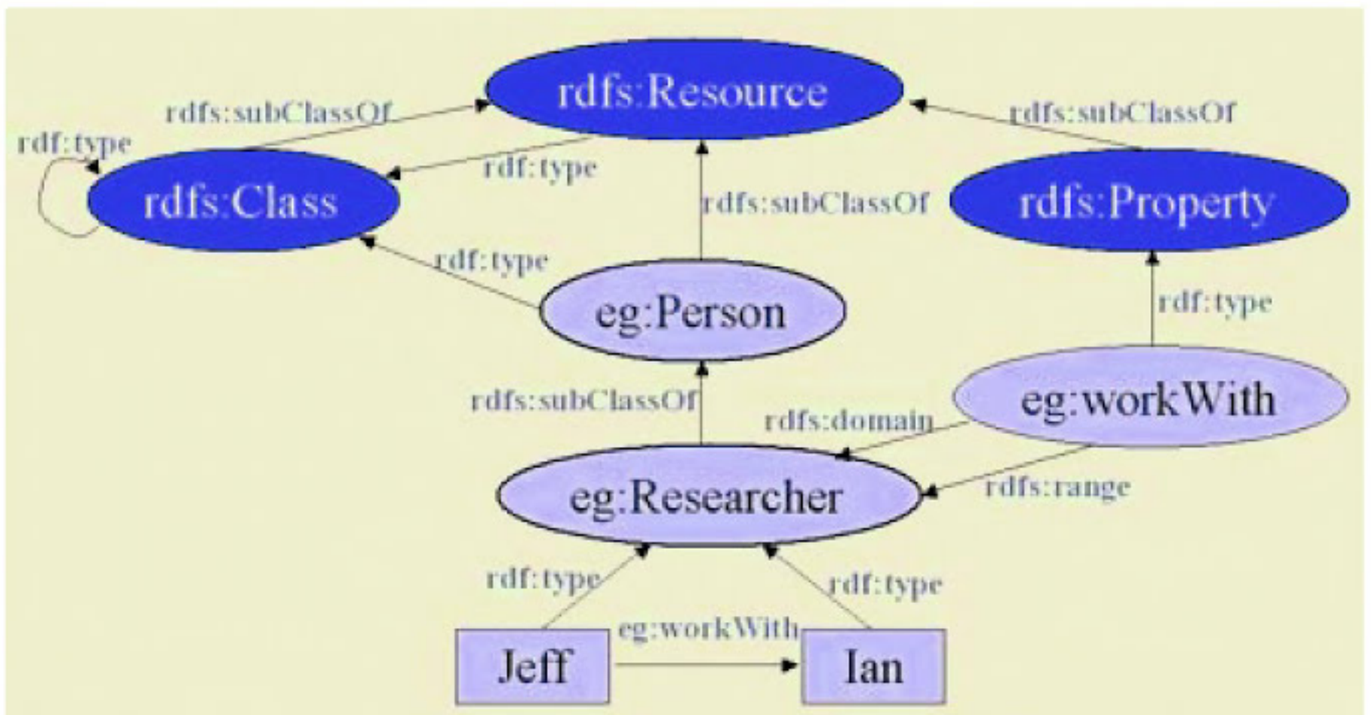


Fig. 1. An Example of Dual Roles in RDFS

rdfs:subClassOf eg:Person. eg:workWith is a property, whose rdfs:domain and rdfs:range are both eg:researcher. There are two instances of eg:Researcher, they are objects Ian and Jeff.

In this example, there seem to be more than one role for rdf:type and rdfs:subClassOf. For instance, rdf:type is used between objects and ontology classes (i.e. Jeff and eg:Researcher) and between ontology classes and built-in classes (i.e. eg:Person and rdfs: Class) etc. Similarly, rdfs:subClassOf is used between two ontology classes (i.e. eg:Researcher and eg:Person) and between two built-in classes (i.e. rdfs:Class and rdfs:Resource) etc.

Furthermore, there is a strange situation for rdfs:Class and rdfs:Resource as discussed in [12]. On the one hand, rdfs:Resource is an instance of rdfs:Class. On the other hand, rdfs:Class is a sub-class of rdfs:Resource. Thus rdfs:Resource is an instance of its sub-class?!

While RDF is mainly used as standard *syntax*, RDFS is expected to be the lowest layer to provide *semantics* for the Semantic Web. However, the existence of dual roles in RDFS makes it difficult to give clear semantics to RDFS. E.g. it is unclear whether rdfs:Resource should be interpreted as an instance or a super-class of rdfs:Class. This might partially explain why Brickley and Guha [3] didn't define the semantics for RDFS. Up to now, there are at least two ways to clear up any confusion and give a clear semantics to the schema language: RDFS(FA) and RDF MT. We will present them individually in the following two sections.

3 RDFS(FA)

In [12] we proposed a sub-language of RDFS - RDFS(FA), which provides a Fixed layer metamodeling Architecture for RDFS. RDFS(FA) eliminates dual roles by defining the modelling primitives *explicitly*, instead of implicitly.

We call the solution *stratification*. The universe of discourse is divided up into different strata (layers). Built-in modelling primitives of RDFS are stratified into different strata of RDFS(FA), so that certain modelling primitives belong to a certain stratum (layer). Different prefixes, e.g. o-, l- or m-, are used to label which stratum modelling primitives belong to. The semantics of modelling primitives

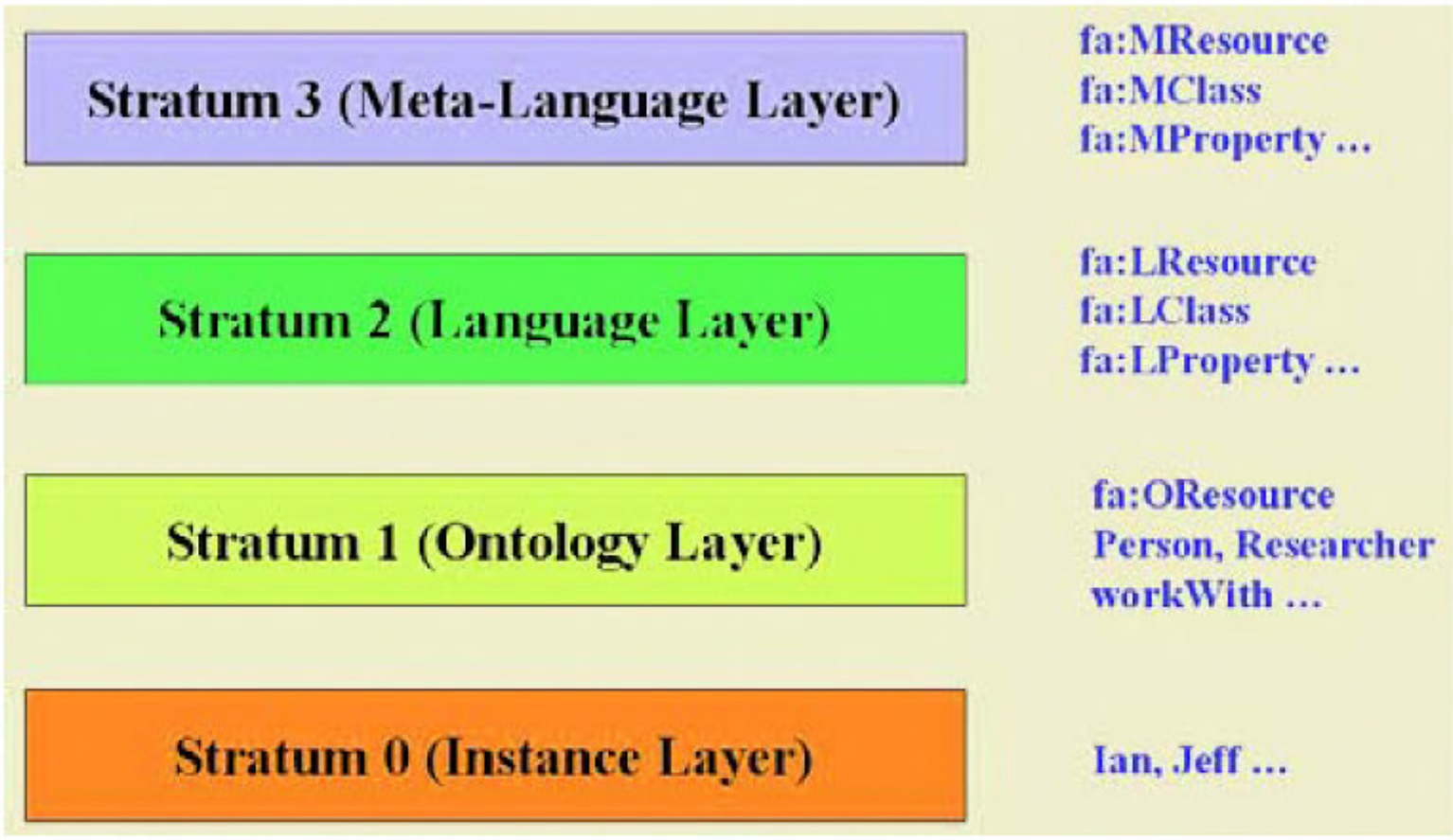


Fig. 2. Metamodeling Architecture (Four Strata) of RDFS(FA)

depend on the stratum they belong to. All these strata form the metamodeling architecture of RDFS(FA). Theoretically there can be infinite number of strata, while in practice, four strata are usually enough [12].

Figure 2 shows the metamodeling architecture (four strata) of RDFS(FA). Here are stratum 0,1,2 and 3. Some people like to call them layers, then they are called the Instance Layer, the Ontology Layer, the Language Layer and the Meta-Language Layer respectively.

Resources in the Instance Layer are objects, e.g. Ian and Jeff. Resources in the Ontology Layer are ontology classes, e.g. Person and Researcher, and ontology properties, e.g. workWith. Resources in the Language Layer are used to define and describe resources in the Ontology Layer, e.g. fa:LClass and fa:LProperty, and resources in the Meta-Language Layer are used to define and describe resources in the Language Layer.

As seen in Figure 2, rdfs:Resource is stratified into three layers, i.e. fa:OResource in the Ontology Layer, fa:LResource in the Language Layer and fa:MResource in the Meta-Language Layer. The same thing happens to rdfs:Class and rdfs:Property. They are stratified into the Language Layer and the Meta-Language Layer.

3.1 No Dual Roles in RDFS(FA)

There are no dual roles in RDFS(FA). Let's visit the same example again, but this time in RDFS(FA) (see Figure 3). As we mentioned earlier, rdfs:Resource and rdfs:Class are stratified into different layers in RDFS(FA), such that fa:OResource is an instance of fa:LClass, and fa:LClass is a sub-class of fa:LResource, while fa:LResource is an instance of fa:MClass.

As far as rdf:type and rdfs:subClassOf in RDFS(FA), rdf:type is stratified into fa:o-type, fa:l-type and fa:m-type[2] where

[2] In order to make it more readable, we change the syntax a bit and use fa:o-type, fa:l-type and fa:m-type, instead of fa:otype, fa:ltype and fa:mtype in [12].

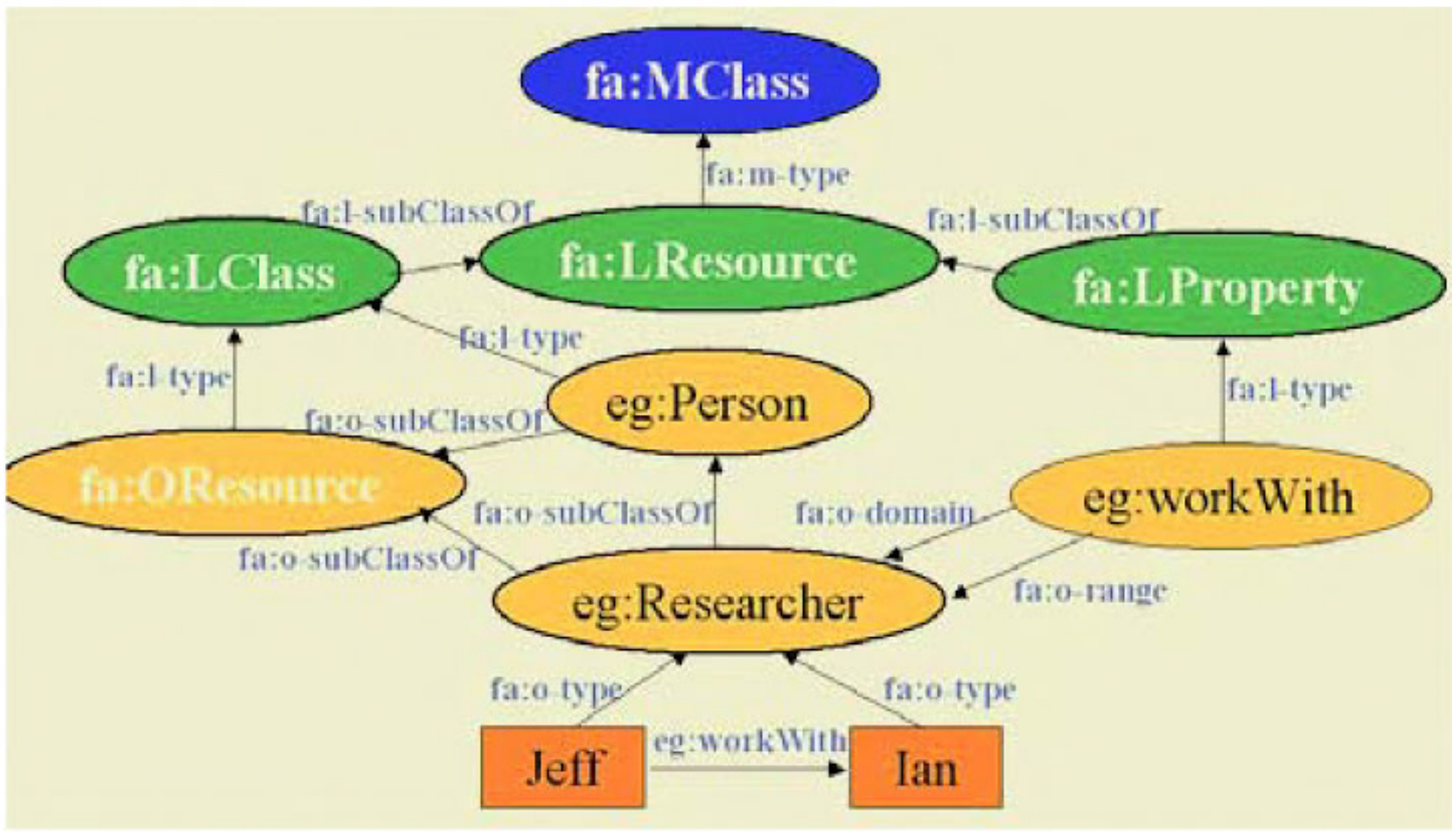


Fig. 3. No Dual Roles in RDFS(FA)

- fa:o-type is used between objects and ontology classes, e.g. Jeff and eg:Researcher;
- fa:l-type is used between resources in the Ontology Layer and resources in the Language Layer, such as eg:Person and fa:LClass, as well as eg:workWith and fa:LProperty;
- fa:m-type is used between resources in the Language Layer and resources in the Meta-Language Layer, e.g. fa:LResource and fa:MClass.

Similarly, rdfs:subClassOf is stratified into fa:o-subClassOf, fa:l-subClassOf and fa:m-subClassOf:

- fa:o-subClassOf is used between two ontology classes, such as eg:Researcher and eg:Person;
- fa:l-subClassOf is used between two classes in the Language Layer, e.g. fa:LClass and fa:LResource;
- fa:m-subClassOf is used between two classes in the Meta-Language Layer.

3.2 Design Philosophy

We discuss the design philosophy of RDFS(FA) in this section. The principle is to build the fixed layer metamodeling architecture on the basis of semantics. There are two groups of fundamental modelling primitives in RDFS(FA), which are classes primitives and property primitives.

What is the semantics of a class primitive in RDFS(FA)? A Class primitive is interpreted as a set of objects or a set of sets. E.g. in Figure 3, eg:Researcher is a class in the Ontology Layer, since it is mapped to a set of objects (e.g. Ian and Jeff). For the same reason, eg:person is also a class. In the Language Layer, fa:LClass is a class since it is mapped to a set of sets (such as eg:Person and eg:Research). fa:Property is also class primitive, since it is mapped to a set of sets (such as eg:workWith).

What is the semantics of a property primitive in RDFS(FA)? A property primitive is interpreted as a set of binary relationships (or pairs) between two

instance of class primitive(s) in the same stratum. E.g. in Figure 3, eg:workWith is a property primitive, since it is mapped to a set of binary relationships between two instances of eg:Researcher in the same stratum (the Ontology Layer). fa:o-subClassOf is also a property primitive, since it is mapped to a set of binary relationships between two instances of fa:LClass in the same stratum (the Language Layer).

Once a property primitive is defined in a certain stratum, it can be used in the adjacent lower stratum. E.g. in Figure 3, once eg:workWith is defined in stratum 1 (the Ontology Layer), it can be used in stratum 0 (the Instance Layer), e.g. Jeff eg:workWith Ian. Once fa:o-subClassOf is defined in stratum 2 (the Language Layer), it can be used in stratum 1 (the Ontology Layer), such as eg:Researcher fa:o-subClassOf eg:Person.

The only exceptions are the type properties, because they are just the instance-of relationships, and always cross adjacent strata (layers). Please note that the type properties are very special, because they are just the *connections* between the two groups of fundamental primitives.[3]

3.3 Formal Description

Based on the design philosophy, we will give a formal description of the stratification of RDFS(FA) in this section. To clarify the presentation, we will not consider datatypes and values in this section; they will be discussed in Section 3.4.

Let V be a vocabulary, which is a set of urirefs. V is divided into disjoint sets $V_0, V_1, V_2, \ldots$, the vocabularies used in strata 0,1,2 ... respectively, where all the individual names are in V_0, and the names of class/property primitives are in $V_{i+1} (i \geq 0)$.

Let $\mathbf{R}_i$, $\mathbf{C}_i$, $\mathbf{P}_i$ be the modelling primitives which are interpreted as the sets of all resources, all classes and all properties respectively in stratum i. According to the design philosophy, since their *instances* are in stratum i, $\mathbf{R}_i$, $\mathbf{C}_i$, $\mathbf{P}_i$ are classes in stratum $i+1$. For example, $\mathbf{R}_1$, $\mathbf{C}_1$ and $\mathbf{P}_1$ are fa:LResource, fa:LClass and fa:LProperty respectively; fa:LResource is mapped to the *set* of all resources in stratum 1 (the Ontology Layer), fa:LClass is mapped to the *set* of all ontology classes (such as eg:Person) in stratum 1, and fa:LProperty is mapped to the *set* of all ontology properties (such as eg:workWith) in stratum 1; since their instances are in stratum 1, fa:LResource, fa:LClass and fa:LProperty exist in stratum 2.

Let $\mathbf{D}_i$ be the domain in stratum i and IE be an interpretation function.

We start from stratum 0. Every individual name $x \in V_0$ is mapped to an object in the domain $\mathbf{D}_0$:

$$IE(x) \in \mathbf{D}_0,$$

the set of all the resources in stratum 0 (the interpretation of $\mathbf{R}_0$) is $\mathbf{D}_0$:

$$IE(\mathbf{R}_0) = \mathbf{D}_0.$$

[3] They are properties *and* are used to join classes and their instances.

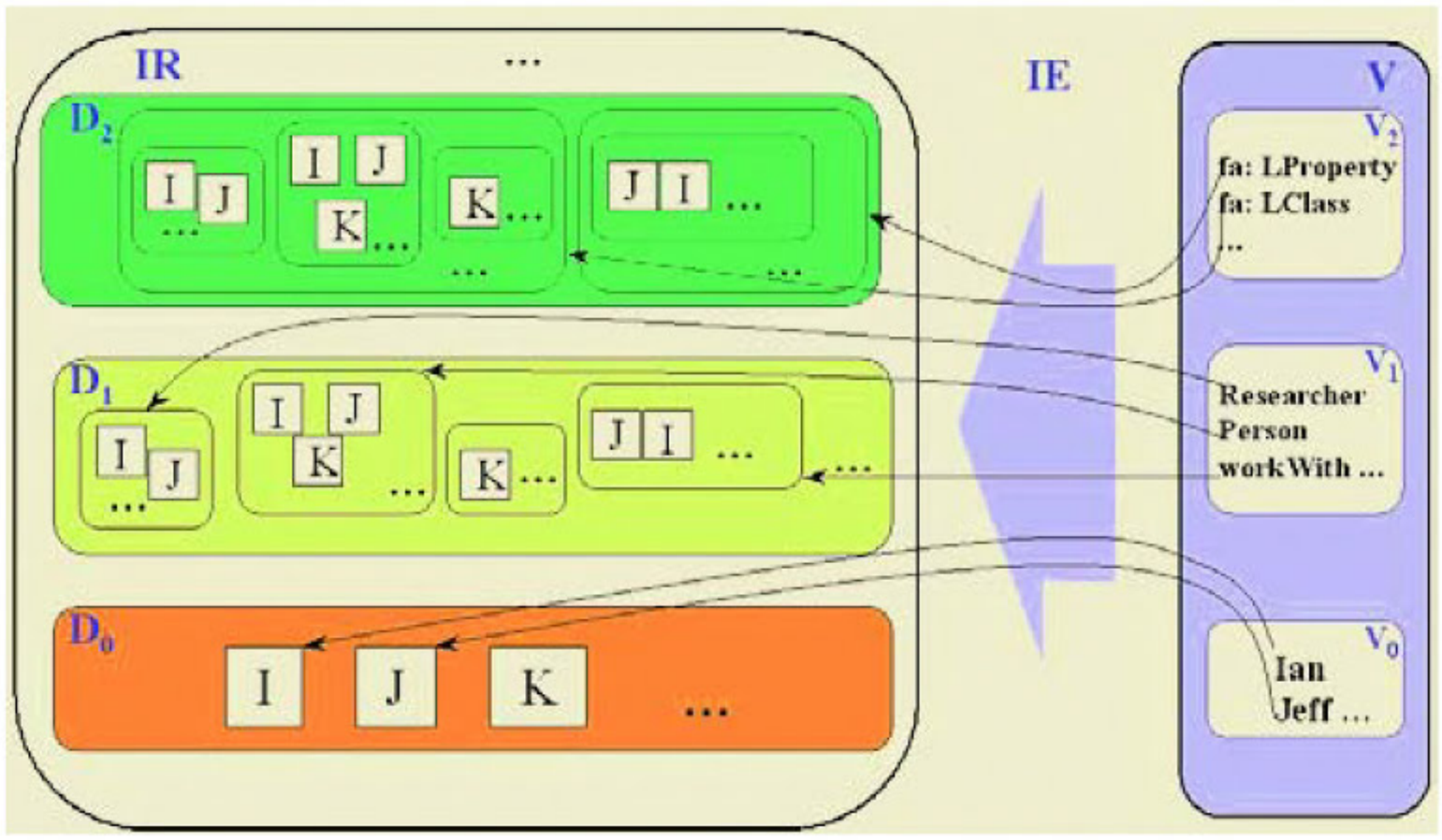


Fig. 4. Interpretation of RDFS(FA)

In stratum $i+1$ (where $i = 0, 1, 2, \ldots$), the set of all resources is equal to the domain of stratum i+1:

$$IE(\mathbf{R}_{i+1}) = \mathbf{D}_{i+1},$$

the domain $\mathbf{D}_{i+1}$ is equal to the union of the set of all classes and the set of all properties in stratum $i+1$:

$$IE(\mathbf{R}_{i+1}) = IE(\mathbf{C}_{i+1}) \cup IE(\mathbf{P}_{i+1})$$

Each class primitive $c_{i+1} \in V_{i+1}$ is interpreted as a set of resources in stratum i:

$$IE(c_{i+1}) \subseteq IE(\mathbf{R}_i),$$

and each property primitive $p_{i+1} \in V_{i+1}$ is interpreted as a set of pairs of resources in stratum i:

$$IE(p_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{R}_i).$$

The $type_{i+1}$ property is interpreted as a set of pairs, where the first resource is in stratum i, and the second resource is a class in stratum $i+1$:

$$IE(type_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{C}_{i+1}).$$

Since $IE(c_{i+1}) \subseteq IE(\mathbf{R}_i)$, we have $IE(c_{i+1}) \in 2^{IE(\mathbf{R}_i)}$, i.e. $IE(c_{i+1}) \in 2^{\mathbf{D}_i}$. According to the intended interpretation of $\mathbf{C}_{i+1}$, we have

$$IE(\mathbf{C}_{i+1}) = 2^{\mathbf{D}_i}.$$

Similarly, since $IE(p_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{R}_i)$, we have $IE(p_{i+1}) \in 2^{IE(\mathbf{R}_i) \times IE(\mathbf{R}_i)}$, i.e. $IE(p_{i+1}) \in 2^{\mathbf{D}_i \times \mathbf{D}_i}$. According to the intended interpretation of $\mathbf{P}_{i+1}$, we have

$$IE(\mathbf{P}_{i+1}) = 2^{\mathbf{D}_i \times \mathbf{D}_i}.$$

every literal $t \in V_0$ is interpreted as a data value in the domain **DD**

$$IE(t) \in \mathbf{DD},$$

and the set of all the resources in stratum 0 (the interpretation of $\mathbf{R}_0$) is $\mathbf{D_0}$

$$IE(\mathbf{R}_0) = \mathbf{D}_0 = \mathbf{OD} \cup \mathbf{DD}.$$

In stratum 1, each class primitive $c_1 \in V_1$ is interpreted as a set of objects:

$$IE(c_1) \subseteq \mathbf{OD},$$

each datatype name $d_1 \in V_1$ is interpreted as a set of data values:

$$IE(d_1) \subseteq \mathbf{DD},$$

each object property $p_1^o \in V_1$ is interpreted as a set of pairs of objects:

$$IE(p_1^o) \subseteq \mathbf{OD} \times \mathbf{OD},$$

each datatype property $p_1^d \in V_1$ is interpreted as a set of pairs, where the first resource is an object and the second resource is a data value:

$$IE(p_1^d) \subseteq \mathbf{OD} \times \mathbf{DD}.$$

In general, DAML+OIL, and other logical layer Semantic Web languages, can be built on top of both the syntax and semantics of RDFS(FA). Furthermore, the stratification of RDFS(FA) can benefit such logical layer Semantic Web languages by offering possibilities of extending them in stratum 3 (the Meta-Language Layer). It can also help avoiding "layer mistakes" [12] in DAML+OIL.

4 RDF Model Theory

Another way to clear up the kinds of confusion of RDFS is RDF Model Theory (RDF MT) [12], which gives a precise semantic theory for RDF and RDFS. It is a W3C working draft when this paper is being written.

An interpretation in the RDF model theory is a triple $\langle \mathbf{IR}, IEXT, IS \rangle$, where **IR** is the domain (of resources); IS is a function that maps URI references to elements of **IR**; and $IEXT$ is a partial function (called extension function) from **IR** to the powerset of $\mathbf{IR} \times \mathbf{IR}$. In RDF MT, meaning is given to properties by first mapping the property URI references to an object of the domain of discourse via IS. The domain object is then mapped into a set of pairs via $IEXT$.

In RDF MT, all resources (including all classes and properties) are objects (see Figure 6). IS maps the URI references of resources to objects in the domain **IR**, e.g. IS maps rdfs:subClassOf to object S, or IS(rdfs:subClassOf) , rdf:type to object T, or IS(rdf:type), rdfs:Class to object C, or IS(rdfs:Class) etc. **IP**, which is a set of all property objects, is a special sub-set of **IR**.

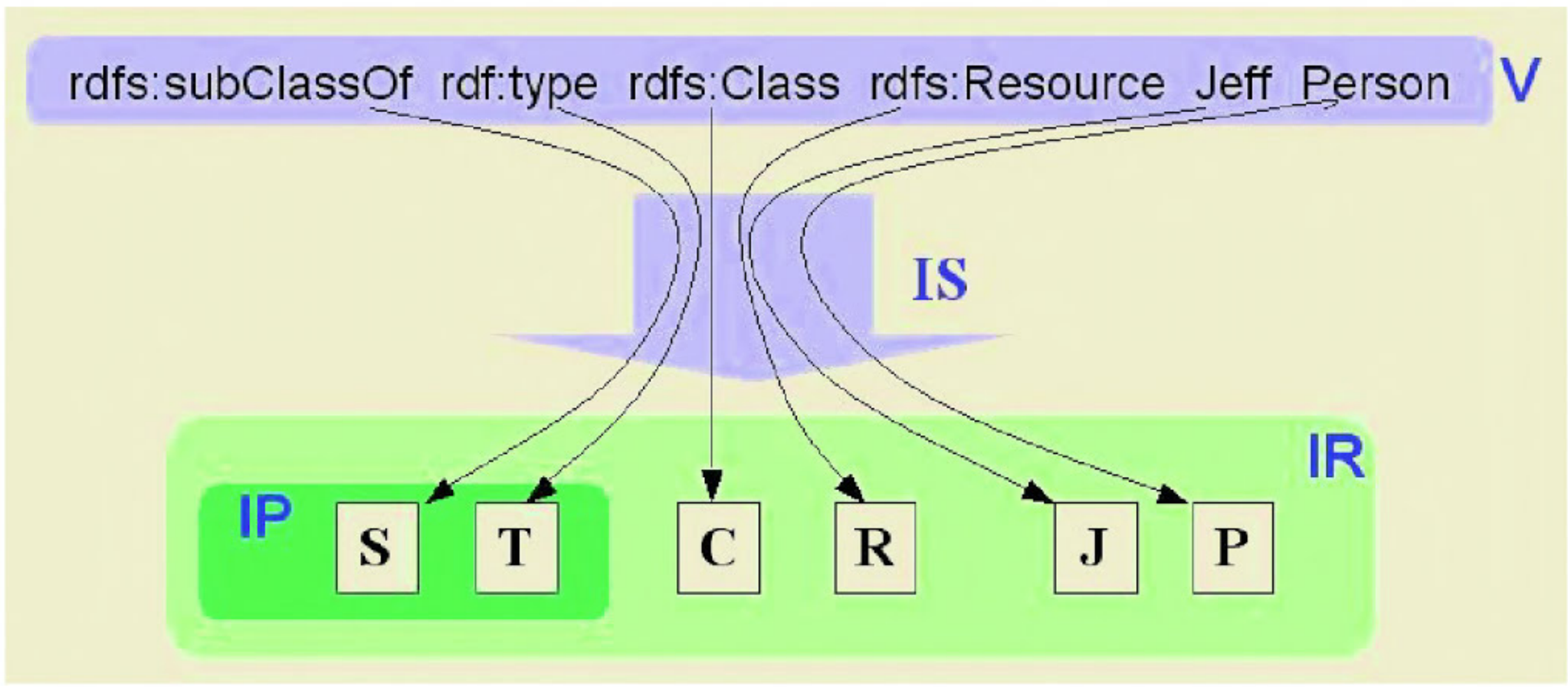


Fig. 6. Resources in RDF MT

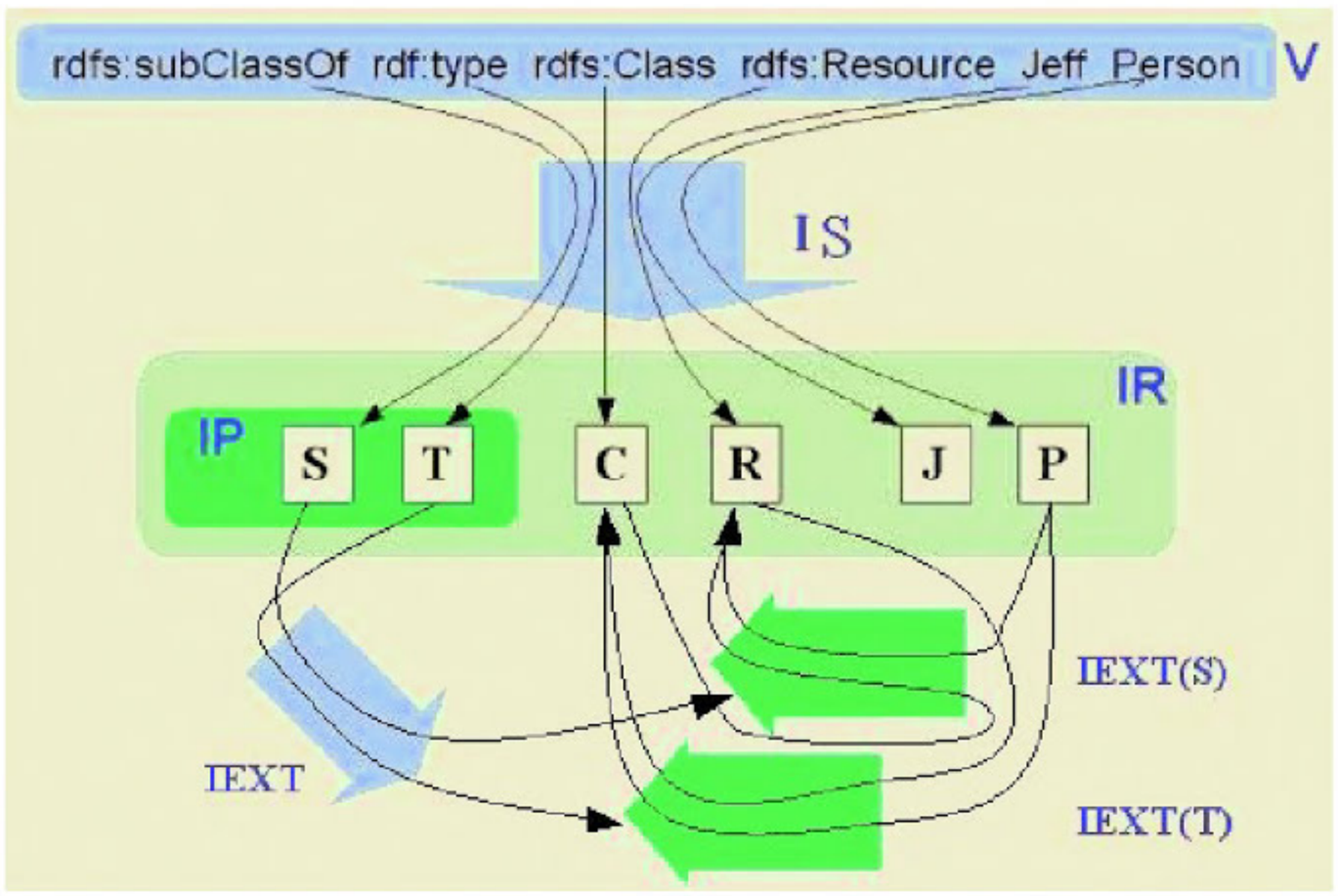


Fig. 7. Interpretation of RDF MT

Extension function $IEXT$ maps property objects to their extensions. E.g. in Figure 7, $IEXT$ maps S to $IEXT$(S), which is a set of pairs $\{\langle$P,R$\rangle$, $\langle$C,R$\rangle\}$. $IEXT$ maps T to $IEXT$(T), which is a set of pairs $\{\langle$P,C$\rangle$, $\langle$R,C$\rangle\}$. Class primitives are not fundamental primitives in RDF MT. Class extension $ICEXT$ is defined through the extension of IS(rdf:type):

$$ICEXT(x) = \{y \mid \langle y, x\rangle \text{ is in } IEXT(IS(\text{rdf:type}))\}$$

In Figure 7, $IEXT$(T)=$\{\langle$P,C$\rangle$, $\langle$R,C$\rangle\}$, so P and R are in ICEXT(C).

4.1 No Confusion in RDF MT

RDF MT justifies dual roles in RDFS by treating classes and properties as objects. In other words, class primitives in RDF MT are interpreted as objects that can have non-empty class extensions; property primitives in RDF MT are

interpreted as objects that can have non-empty extensions. Even though it is a bit strange to some people, there is no confusion in RDF MT.

Let's revisit the same example in RDF MT. rdfs:Class and rdfs:Resource are mapped to objects C and R in the domain of resource by IS, therefore rdfs:Class is rdfs:subClassOf rdfs:Resource means the pair of R and C is in the extension of the rdfs:subClassOf object S

$$\langle \text{C,R} \rangle \in IEXT(\text{S})$$

while rdfs:Resource is instance of rdfs:Class means the pair of R and C is in the extension of the rdf:type object T

$$\langle \text{R,C} \rangle \in IEXT(\text{T}).$$

According to the definition of $ICEXT$, we have

$$\text{R} \in ICEXT(\text{C}).$$

In this way, the situation between rdfs:Class and rdfs:Resource is given a well defined meaning.

5 Comparing the Two Approaches

In Section 3 and 4, we described two approaches to clear up any confusion of RDFS. In this section, we will first compare these two approaches, and then discuss their advantages and disadvantages.

5.1 Main Differences

On how to clear up the confusion of RDFS, RDFS(FA) stratifies dual roles into different strata and defines modelling primitives explicitly; while RDF MT justifies dual roles by treating classes and properties as objects, making dual roles a feature, instead of a problem, of the language.

Differences in Syntax. RDFS(FA) provides extra syntax to add restriction of stratification, so as to enable the layering style of RDFS, and to avoid dual roles. Since elements of RDFS(FA) exist in different strata of the domain of discourse, (prefixes of) symbols of elements should indicate this fact. E.g. fa:OResource is in stratum 1 (the Ontology Layer), fa:LProperty is in stratum 2 (the Language Layer), and fa:MClass is in stratum 3 (the Meta-Language Layer). [5]

Secondly, valid RDFS(FA) statements should be consistent with the design philosophy (see Section 3.2) of RDFS(FA). If one defines a class in stratum $i+1$, then the instances of that class should be in stratum i, e.g.

[5] Note that properties, except the type properties, are always used one stratum lower than the one where they are defined, e.g. fa:o-subClassOf is defined in stratum 2 (the Language Layer) and used in stratum 1 (the Ontology Layer).

```
<fa:LClass rdf:about=‘‘\#Person’’>
</fa:LClass>
```

Since fa:LClass is in stratum 2 (the Language Layer), the Class “Person” should be in stratum 1 (the Ontology Layer).

Similarly, if one defines a property in stratum $i+1$, then the classes as the domain and range of the property should be in stratum $i+1$ as well.

```
<fa:LProperty rdf:about=‘‘\#hasFriend’’>
  <fa:o-domain rdf:resource=‘‘\#Person’’/>
  <fa:o-range rdf:resource=‘‘\#Person’’/>
</fa:LProperty>
```

Furthermore, RDFS(FA) distinguishes its predefined property primitives (such as fa:o-type and fa:o-domain etc) from user-defined properties (such as eg:hasFriend). For instance, users are not allowed to define sub-properties of fa:o-type. In RDFS, however, there is no such restriction.

Differences in Semantics. Besides differences in syntax, there are also differences in semantics between RDFS(FA) and RDF MT.[6]

First of all, RDFS is more expressive than RDFS(FA). The reason for being less expressive is that the stratification of RDFS(FA) disallows most cross-stratum binary relationships (except the type properties). However, it could be argued that these kinds of relationship are too confusing for most users.

Secondly, there are different *fundamental* primitives in RDFS(FA) and RDF MT. In RDFS(FA), both class and property primitives are fundamental primitives, i.e., both are directly interpreted by the interpretation function IE. As seen in Section 3.3, RDFS(FA) class primitives in stratum $i+1$ are interpreted as sets of elements in stratum i, while property primitives in stratum $i+1$ are interpreted as sets of pairs of elements in stratum i. The type properties are *special* properties: their interpretations are just the *instance-of* relationships.

In RDF MT, although both class and property primitives are objects, *only* property primitives are fundamental primitives, i.e., only property primitives can be given non empty extensions by $IEXT$. The class extension $ICEXT$ is simply derived from the $IEXT$ extension of the rdf:type object. Note, however that although the rdf:type property is used to define membership of classes, in all other respects it is treated in the same way as any other property.

Thirdly, RDFS(FA) and RDF MT interpret property (and class) primitives in different ways. In RDFS(FA) (and conventional FOL), property (class) symbols are directly mapped to a set of pairs of elements (elements) in the domain. While in RDF MT, on the other hand, meaning is given to property and class symbols by first mapping them (via IS) to objects in the domain. A property object is then mapped (via $IEXT$) to a set of pairs of objects in the domain.

Based on the differences between RDFS(FA) and RDF MT, we will discuss their advantages and disadvantages in next section.

[6] Readers are advised to refer to Figure 4 and 7 for better understanding of these differences.

5.2 Advantages of RDF MT (Disadvantages of RDFS(FA))

Since there is no restriction of stratification, RDF MT (RDFS) is more expressive than RDFS(FA). This advantage of RDF MT is believed to be consistent with the following philosophy: anyone can say anything about anything. In RDF MT, this means

- properties can be defined between any two resources;
- any resource can be an instance of any resource (including itself).

However, this *"unlimited"* expressive power can lead to problems, as we will see in the following section.

5.3 Disadvantages of RDF MT (Advantages of RDFS(FA))

The syntax rules in RDFS are very weak, and there are not many restrictions on writing RDFS triples. As a result, this can be confusing and difficult to understand and, more importantly, the specification of its semantics requires a non-standard model theory, i.e. RDF MT.

This leads to semantic problems when trying to layer[7] conventional FOL, like DAML+OIL and OWL. E.g., as DAML+OIL is more expressive than RDFS, a large and more complex set of semantic conditions[8] is required to capture the meaning and characteristic of its additional constructs. It is very difficult to get such semantic conditions correct, not to mention that one should also prove that they are right.

There are at least three known problems if we extend RDFS with more expressive FOL constructs, e.g. conjunctions and qualified number restrictions, and extend RDF MT to so called "RDF+ MT" to give meaning to this extended language. These known problems are: (i) too few entailments; (ii) contradiction classes; (iii) size of the universe.

Too Few Entailments. Entailment is the key idea which connects model-theoretic semantics to real-world applications. What is entailment? In RDF MT, entailment means "If A entails B, then any interpretation that makes A true also makes B true," so that an assertion of A already contains the same "meaning" as an assertion of B [12].

[13] first addressed the problem of too few entailments and gave the following example: if John is an instance of the class Student ∩Employee∩European, is John an instance of the class Employee ∩Student?

In RDFS(FA) and FOL, the answer is simply "yes", since Student∩Employee∩ European is a sub-class of Employee ∩Student, so every instance of the former class is also an instance of the later one.

[7] RDFS, in some sense, is a very limited language, and serves as the bottom semantic layer of Semantic Web languages. So it is both necessary and desirable to layer more expressive ontology languages on top of it.

[8] Since the constructs of RDFS are simple, the set of semantic conditions for RDFS is relatively small.

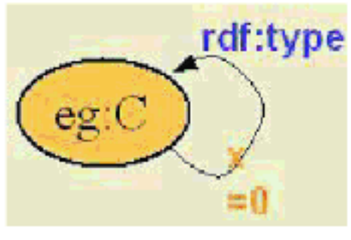


Fig. 8. Contradiction Classes

However, in "RDF+ MT", since every concept is also an object, "John is an instance of the concept Student∩Employee ∩European" can't guarantee there exists an object for Employee ∩Student in all the interpretations that make "John is an instance of the concept Student∩ Employee∩European" true. So the answer in RDF+ MT is "no".

In this case, the "RDF+ MT" semantics seems to be broken, because the semantics of an ontology language should give meaning to any possible class expressions. In order to fix the problem, one can/should introduce comprehension axioms to add all possible missing objects into the domain, e.g. the Employee ∩Student in this example. But that is surely a very difficult task. Theoretically, it is yet to be proved that proper objects are all added into the universe, no more and no less. Practically, there will be infinite numbers of possible class expressions.[9] It is still unknown whether there exists a practical approach to solve the problem.

Contradiction Classes. [13,14] also addressed the problem of contradiction classes. In RDFS, resources can be defined as instances of themselves, and rdf:type is treated as any other property. So, if the extended language supports qualified number restrictions, one can define a class eg:C as an instance of itself, and add a cardinality constraint "=0" on the rdf:type property (see Figure 8). It is impossible for one to determine the membership of this class. If an object is an instance of this class, then it isn't, because instances should have no rdf:type property pointing to itself. But if it isn't then it is. This is a contradiction class.

One might argue that we can simply avoid defining such classes. However, with the comprehension axioms (see Section 5.3.1), we must add all possible class objects into the domain, and the above contradiction class is one of them. In this way, all the interpretations will have such contradiction classes, and thus have ill-defined class memberships. Again, the "RDF+ MT" semantics seems to be broken[10]. RDFS(FA) does't have this problem, because the type properties are not treated as ordinary properties.

Size of the Universe. Like RDF MT, in "RDF+ MT" there is a pre-defined vocabulary (e.g. rdf:type, rdfs:Property etc), terms from which are mapped to

[9] Think about all the possible conjunctions, disjunctions, exist restrictions, value restrictions and qualified number restrictions ...

[10] One might solve the problem by making the comprehension axioms more complex. It is yet to be proved that we keep the objects of all possible contradiction classes outside the universe.

anyone can say anything about anything. Properties can be defined between any two resources, and a resource can be an instance of any resource (including itself). Some people, however, worry about this "unlimited" expressive power, in particular when layering more expressive languages on top of RDFS.

The advantage of RDFS(FA) is that FOLs, e.g. DAML+OIL, can be built on top of both the syntax and semantics of RDFS(FA). Furthermore, the stratification of RDFS(FA) can benefit such logical layer Semantic Web languages by offering possibilities of extending them in stratum 3 (the Meta-Language Layer).

The disadvantage of RDF MT is that there are at least three known problems if we extend RDFS with more expressive FOL constructs, and extend RDF MT to the so called "RDF+ MT" to give meaning to this extended language (see Section 5.3). Moreover, layering FOL on top of RDFS doesn't lead directly to any "computational pathway", i.e. it is not clear whether/how applications would be able to reason with languages layered on top of RDFS.

Generally speaking, on the one hand, RDF MT allows for a larger number of models of the universe, and can represent more heterogeneous states of affairs. On the other hand, RDFS(FA) allows more expressive ontology languages, e.g. DAML+OIL, to be layered on top of it, so that people can say more things about a smaller number of (more homogeneous) models of the universe.

It has yet to be proved that RDF MT can be extended to give a coherent meaning to more expressive ontology languages[11]. Moreover, it is not clear if the more heterogeneous models supported by RDF MT would be needed in many realistic applications. Given that the set of RDFS(FA) statements is a subset of the set of RDFS statements, one possible solution would be to support both semantics, with users able to choose if they are willing to use the layering style of RDFS to facilitate the extension of the language with more expressive modelling primitives. This solution could provide a good guideline for more expressive logical ontology languages designed on top of RDFS, and for users to be aware of the above problems when they choose to use the non-layering style of RDFS.

Acknowledgments. We would like to thank Peter Patel-Schneider for discussion on the stratification of RDFS(FA), and Peter Aczel for discussion on non-well founded sets.

References

1. T. Berners-lee. Semantic Web Road Map. W3C Design Issues. URL `http://www.w3.org/DesignIssues/Semantic.html`, Oct. 1998.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition) – W3C Recommendation 6 October 2000. Technical report, World Wide Web Consortium, 2000. Available at `http://www.w3.org/TR/REC-xml`.

[11] Note that in OWL the RDFS-compatible model-theoretic semantics has the same consequence as the direct semantics on OWL ontologies, *only* when the separate vocabulary restriction is satisfied [15].

3. D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommentdation, Mar. 2000.
4. J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling Knowledge Representation on the Web by Extending RDF Schema. In *Proc. of the 10th WWW conf. (WWW10)*, Hong Kong, May 2001.
5. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. S. eds. OWL Web Ontology Language 1.0 Reference. URL `http://www.w3.org/TR/owl-ref/`, Nov 2002.
6. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proc. of the 12th Eur. Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'00)*, pages 1–16, 2000.
7. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
8. P. Hayes. RDF Model Theory. Apr 2002. W3C Working Draft, URL `http://www.w3.org/TR/rdf-mt/`.
9. J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.
10. O. Lassila and R. R.Swick. Resource Description Framework (RDF) Model and Syntax Specification – W3C Recommendation 22 February 1999. Technical report, World Wide Web Consortium, 1999.
11. W. Nejdl, M. Wolpers, and C. Capella. The RDF Schema Specification Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000*, Apr. 2000.
12. J. Z. Pan and I. Horrocks. Metamodeling Architecture of Web Ontology Languages. In *Proc. of the 2001 Int. Semantic Web Working Symposium (SWWS 2001)*, July 2001. URL `http://www.cs.man.ac.uk/ panz/Zhilin/download/Paper/ Pan-Horrocks-rdfsfa-2001.pdf`.
13. P. F. Patel-Schneider. Layering the Semantic Web: Problems and Directions. In *Proc. of the 2002 Int. Semantic Web Conf. (ISWC 2002)*, Jun 2002.
14. P. F. Patel-Schneider. Two Proposals for a Semantic Web Ontology Language. In *2002 International Description Logic Workshop*, Apr 2002.
15. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, Mar. 2003. W3C Working Draft, URL `http://www.w3.org/TR/2003/WD-owl-semantics-20030331/`.
16. M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.
17. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference Description of the DAML+OIL(March 2001) Ontology Markuk Language. DAML+OIL Document. Available at `http://www.daml.org/2000/12/reference.html`, Mar. 2001.
18. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. A Model-Theoretis Semantics of DAML+OIL(March 2001). DAML+OIL Document, URL `http://www.daml.org/2001/03/model-theoretic-semantics.html`, Mar. 2001.

Web Ontology Reasoning with Datatype Groups

Jeff Z. Pan and Ian Horrocks

Department of Computer Science,
University of Manchester, UK M13 9PL
{pan,horrocks}@cs.man.ac.uk

Abstract. When providing reasoning services for ontology languages such as DAML+OIL and OWL, it is necessary for description logics to deal with "concrete" datatypes (strings, integers, etc.) as well as "abstract" concepts and relationships. In this paper, we present a new approach, the datatype group approach, to integrating DLs with multiple datatypes. We discuss the advantages of such approach over the existing ones and show how a tableaux algorithm for the description logic $\mathcal{SHOQ}(\mathbf{D_n})$ can be modified in order to reason with datatype groups.

1 Introduction

Datatypes are important in the Semantic Web ontologies and applications, because most of which need to represent, in some way, various "real world" properties such as size, weight and duration, and some other complex user defined datatypes. Reasoning and querying over datatype properties are important and necessary if these properties are to be understood by machines.

For instance, e-shops may need to classify items according to their sizes, and to reason that an item which has height less than 5cm and the sum of length and width less than 10cm belongs to a class, called "small-items", for which no shipping costs are charged. Accordingly the billing system will charge no shipping fees for all the instances of the "small-items"class.

Various Web ontology languages, such as RDF(S) [4], OIL [5], DAML+OIL[1] [7] and OWL[2], have witnessed the importance of datatypes in the Semantic Web. All of them support datatypes. For instance, the DAML+OIL language supports unary datatype predicates and qualified number restrictions on unary datatype predicates, e.g. a "less than 21" predicate could be used with the datatype property *age* to describe objects having age less than 21.[3]

Description Logics (DLs)[1], a family of logical formalisms for the representation of and reasoning about conceptual knowledge, are of crucial importance to

[1] http://www.daml.org/
[2] http://www.w3.org/2001/sw/WebOnt/
[3] It is important to distinguish between a unary predicate such as "less than 21", which is true of any number x that is less than 21, and a binary predicate such as "less than", which is true of any pair of numbers x, y where x is less than y.

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 47–63, 2003.

the development of the Semantic Web. Their role is to provide formal underpinnings and automated reasoning services for Semantic Web ontology languages such as OIL, DAML+OIL and OWL.

Using datatypes within Semantic Web ontology languages presents new requirements for DL reasoning services. Firstly, such reasoning services should be compatible with the XML Schema type system[3], and may need to support many different datatypes. Furthermore, they should be easy to extend when new datatypes are required.

DL researchers have been working on combining DLs and datatypes for quite a long time. Bader and Hanschke [2] first presented the concrete domain approach, Lutz [10] studied the effect on complexity of adding concrete domains to a range of DLs, Horrocks and Sattler [8] proposed the $\mathcal{SHOQ}(\mathbf{D})$ DL which combines DLs and type systems (e.g. the XML Schema type system), and more recently Pan and Horrocks [12] presented the $\mathcal{SHOQ}(\mathbf{D_n})$ DL, which extends $\mathcal{SHOQ}(\mathbf{D})$ with n-ary datatype predicates and qualified number restrictions.

To reason with $\mathcal{SHOQ}(\mathbf{D})$ and $\mathcal{SHOQ}(\mathbf{D_n})$, type checkers are introduced to work with DL "concept reasoners". By using a separate type checker, we can deal with an arbitrary conforming set of datatypes and predicates without compromising the compactness of the concept language or the soundness and completeness of the decision procedure [8]. Whenever new datatypes are required, only the type checkers need to be updated and the DL concept reasoner can be reused. The result is a framework that is both robust and flexible.

To support type systems and type checkers, in this paper, we present the datatype group approach, which extends existing approaches in order to overcome problems and limitations such as counter-intuitive negation, disjointness of different datatypes and mixed datatype predicates. We then describe an algorithm for reasoning with $\mathcal{SHOQ}(\mathbf{D_n})$, which improves on the one presented in [12] by allowing simpler "deterministic" type checkers to be used.

Next section, we will show the expressive power of the $\mathcal{SHOQ}(\mathbf{D_n})$ DL by an example of using n-ary datatype predicates.

2 An Example: Using Datatypes

Maybe you still remember the "small-items" example presented in last section, in which `the sum of ... less than 10cm` is an n-ary datatype predicate. In this section, we give another example of using n-ary (this time n=2) datatype predicates to support unit mapping.

Example 1 Miles and Kilometers.[4] Unit mapping is important because of the variety of units. For instance, you can find more than one hundred and sixty

[4] This example is inspired by a discussion about datatypes in the www-rdf-logic mailing list:
`http://lists.w3.org/Archives/Public/www-rdf-logic/2003Mar/0048.html`.

length units in `http://www.chemie.de/`[5]. This example concerns the mapping between the units of mile and kilometer.

Firstly, we define two datatypes to represent the units of mile and kilometer. Since the positive float is not a built-in XML Schema datatype, we define two derived XML Schema datatypes[6] lengthInMile and lengthInKMtr, as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema}"
                  xmlns="http://www.example.org/length-units.xsd}">
  <xsd:simpleType name="lengthInMile">
    <xsd:restriction base="xsd:float">
      <xsd:minInclusive value="0/>
    </xsd:restriction base="xsd:float">
  </xsd:simpleType>
  <xsd:simpleType name="lengthInKMtr">
    <xsd:restriction base="xsd:float">
      <xsd:minInclusive value="0/>
    </xsd:restriction base="xsd:float">
  </xsd:simpleType>
</xsd:schema>
```

We can then make use of these datatypes in DAML+OIL ontologies.[7] E.g., if we have datatype properties "length-mile" and "length-kmtr" defined in a river ontology:

```
<rdf:RDF
        xmlns="http://www.example.org/river#"
           ...
   xmlns:unit="http://www.example.org/length-units.xsd">
   <owl:DataTypeProperty rdf:ID="length">
     <rdfs:range rdf:resource="xsd:float"/>
   </owl:DataTypeProperty>
   <owl:Class rdf:ID="River"/>
   <owl:DataTypeProperty rdf:ID="length-mile">
     <rdfs:subPropertyOf rdf:resource="\#length"/>
     <rdfs:domain rdf:resource="\#River"/>
     <rdfs:range rdf:resource="unit:lengthInMile"/>
   </owl:DataTypeProperty>
   <owl:DataTypeProperty rdf:ID="length-kmtr">
     <rdfs:subPropertyOf rdf:resource="\#length"/>
     <rdfs:domain rdf:resource="\#River"/>
     <rdfs:range rdf:resource="unit:lengthInKMtr"/>
   </owl:DataTypeProperty>
```

[5] `http://www.chemie.de/tools/units.php3?language=e\&property=m`

[6] When we talk about XML Schema datatype in this paper, we mean XML Schema simple types.

[7] OWL currently does not support the use of derived datatypes.

we can describe the length of the Yangtze river as 3937.5 miles *and* 6300 kilometers:

```
<River rdf:ID="Yangtze">
  <length-mile rdf:datatype="unit:lengthInMile">3937.5</length>
</River>
<River rdf:ID="Yangtze">
  <length-kmtr rdf:datatype="unit:lengthInKMtr">6300</length>
</River>
```

We can specify the mapping between miles and kilometers using a binary datatype predicate. Sadly, XML Schema does not support n-ary datatype predicates. We have used an XML style syntax to present this predicate in `http://www.example.org/length-units.xsd` as follows:[8]

```
<predicate name="kmtrsPerMile" arity="2">
  <par var="i" base="lengthInKMtr">
  <par var="j" base="lengthInMile">
  <constraint val="i=1.6*j">
</predicate>
```

The binary (with $arity = 2$) datatype predicate "kmtrsPerMile" is defined over the datatypes "lengthInKMtr" and "lengthInMile".

Now we can add a restriction to the "River" class and require that the value of the "length-kmtr" property be 1.6 times that of the "length-mile" property. Again, we could imagine an extension of a language such as OWL to support the use of n-ary datatype predicates:

```
<owl:Class rdf:about="River">
  <rdfs:subClassOf>
    <owl:NaryRestriction>
      <owl:onProperties rdf:parseType="Collection">
        <owl:DatatypeProperty rdf:ID="length-kmtr">
        <owl:DatatypeProperty rdf:ID="length-mile">
      </owl:onProperties>
      <owl:allTuplesSatisfy rdf:resource="unit:kmtrsPerMile"/>
    </owl:NaryRestriction>
  <rdfs:subClassOf>
</owl:Class>
```

Note that, in the above restriction, the order of the properties in the `onProperties` list is significant, and "kmtrsPerMile" is a datatype predicate whose arity must match the number of properties in the `onProperties` list. Such restriction is expressible in the $\mathcal{SHOQ}(\mathbf{D_n})$ DL (see section 4).

[8] Currently we are also working on extending the common DL Interface DIG/1.1 to support n-ary datatype predicates in a similar manner.

One example of reasoning with the above ontology and datatypes is to check whether a large set of instances of the ontology class River are consistent with the above restriction of the River class. ♢

As we hope the above example shows, n-ary datatype predicates would be very useful in Semantic Web ontologies and applications. In next section, we will start to investigate how to provide DL reasoning services for ontologies and datatypes.

3 Concrete Domains and Datatype Groups

As mentioned in Section 1, Description Logic researchers have been working on combining DLs and datatypes for quite a long time, although they might not always have used the term "datatype". It was Baader and Hanschke [2] who first presented a rigorous treatment of datatypes, which they called "concrete domains". Lutz [9] presents a survey of DLs with concrete domains, concentrating on the effect on complexity and decidability of adding concrete domains to various DLs. More recently, Horrocks and Sattler [8] proposed a new approach to cope with datatypes structured by a type system. In the rest of this section, we will briefly describe the above two approaches, explaining their advantages and disadvantages in coping with multiple datatypes, then extend these approaches and present the datatype group approach.

3.1 The Concrete Domain Approach

A "concrete domain" is formally defined as followed:

Definition 1 *(**Concrete Domain.**)* A concrete domain $\mathcal{D}$ consists of a pair $(\Delta_\mathcal{D},\Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is the domain of $\mathcal{D}$ and $\Phi_\mathcal{D}$ is a set of predicate names. Each predicate name P is associated with an arity n, and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. Let $\mathbf{V}$ be a set of variables. A predicate conjunction of the form

$$c = \bigwedge_{j=1}^{k} P_j(v_1^{(j)}, \ldots, v_{n_j}^{(j)}), \tag{1}$$

where P_j is an n_j-ary predicate and the $v_i^{(j)}$ are variables from $\mathbf{V}$, is called *satisfiable* iff there exists a function δ mapping the variables in c to data values in $\Delta_\mathcal{D}$ s.t. $(\delta(v_1^{(j)}), \ldots, \delta(v_{n_j}^{(j)})) \in P_j^\mathcal{D}$ for $1 \leq j \leq k$. Such a function δ is called a *solution* for c. A concrete domain $\mathcal{D}$ is called *admissible* iff

1. $\Phi_\mathcal{D}$ is closed under negation[9] and contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$ and
2. the satisfiability problem for finite conjunctions of predicates is decidable.

[9] Closed under negation requires that if $P \in \Phi_\mathcal{D}$, then $\overline{P} \in \Phi_\mathcal{D}$.

will be modified from $\overline{P_j}^{\mathcal{D}_1} = \Delta_{\mathcal{D}_1}^{n_j} \setminus P_j^{\mathcal{D}_1}$ to $\overline{P_j}^{\mathcal{D}_1 \oplus \mathcal{D}_2} = (\Delta_{\mathcal{D}_1} \cup \Delta_{\mathcal{D}_2})^{n_j} \setminus P_j^{\mathcal{D}_1}$. We can see that $\overline{P_j}^{\mathcal{D}_1 \oplus \mathcal{D}_2}$ equals to $\overline{P_j}^{\mathcal{D}_1}$ union a set of n_j-ary tuples involving type errors, where at least one of the variables of P_j is mapped to a data value in $\mathcal{D}_2$ instead of $\mathcal{D}_1$. Counting type errors in the negations of datatype predicates may be counter-intuition. Let's see an example.

Example 3 The $>_5$ and $\leq_5$ Datatype Predicates. We first only consider the concrete domains INT, where datatype predicates $>_5, \leq_5 \in \Phi_{INT}$. We have $\overline{>_5}^{INT} = \leq_5^{INT}$ (see figure 1). When we consider two concrete domain INT and $STRING$, we might still expect that $\overline{>_5}^{INT \oplus STRING}$ only include the *integers* that are not in $>_5^{INT}$, e.g., the integer 3. However, since $\overline{>_5}^{INT \oplus STRING} = (\Delta_{INT} \cup \Delta_{STRING}) \setminus >_5^{INT}$, the string "Fred" is also in $\overline{>_5}^{INT \oplus STRING}$. ♢

As well as being counter-intuitive, the change in the interpretation of $\overline{P_j}$ does not fit well with the idea of using type checkers to work with DL reasoner. We will come back to this in Section 3.3.

Disjoint Domains. The concrete domain approach requires that two concrete domains be disjoint with each other if they are to be combined to form a new concrete domain. This does not accord with XML Schema datatypes, where some datatypes can be sub-types of other datatypes.

Mixed Datatype Predicates. The "kmtrsPerMile" example illustrates another limitation of the concrete domain approach: all of the arguments to a predicate must be from the same concrete domain. The example shows that in some cases it may be useful to have predicates taking arguments of different datatypes.

3.2 The Type System Approach

To solve the "disjoint domains" problem mentioned in the previous section, Horrocks and Sattler [8] proposed a new approach to combine DLs and *type systems* (e.g. XML Schema type system). A type system typically defines a set of "base datatypes", such as *integer* or *string*, and provides a mechanism for deriving new datatypes from existing ones. In this approach, multiple datatypes may be defined over a *universal concrete domain*.

Definition 2 *(***Universal Concrete Domain.***)* The universal concrete domain $\mathbf{D}$ consists of a pair $(\Delta_{\mathbf{D}}, \Phi^1)$, where $\Delta_{\mathbf{D}}$ is the domain of all datatypes and Φ^1 is a set of datatype (unary datatype predicate) names. Each datatype name d is associated with a unary datatype predicate $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$. Let $\mathbf{V}$ be a set of variables, a datatype conjunction of the form

$$c_1 = \bigwedge_{j=1}^{k} d_j(v^{(j)}), \tag{2}$$

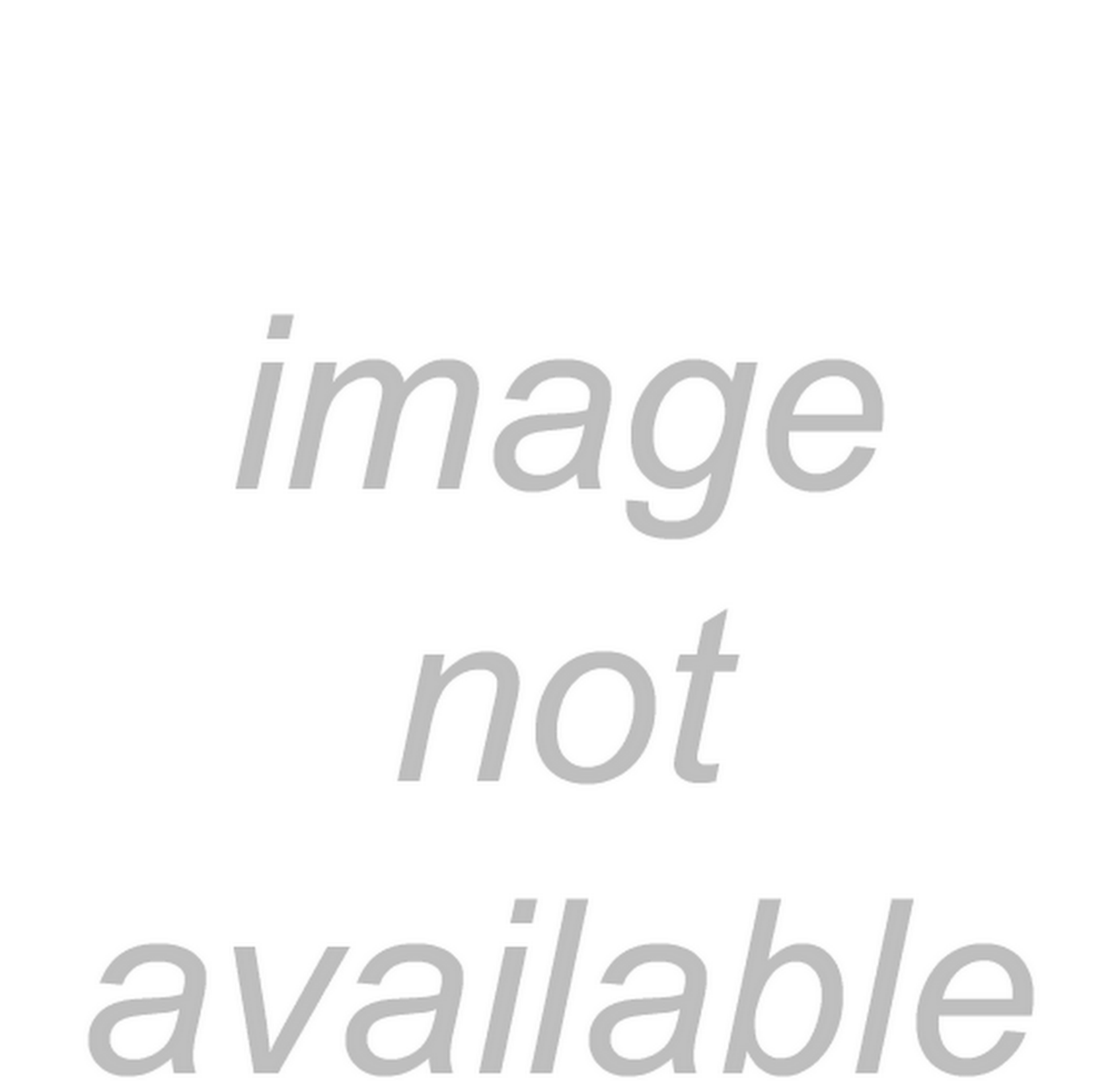


image
not
available

3.3 The Datatype Group Approach

In this section we describe an extension of the type system approach which we call the datatype group approach. Our motivation is to provide an easy and intuitive way to cope with datatypes structured by type systems, and to support n-ary datatype predicates such that (i) the interpretations of negations of datatype predicates does not change when new datatypes are introduced into a datatype group, and (ii) it is possible to reuse existing concrete domain algorithms for the satisfiability problem of predicate conjunctions (1). A "datatype group" is formally defined as follows.

Definition 3 *(***Datatype Group.***)* *A datatype group* $\mathcal{G}$ is a tuple $(\Delta_{\mathbf{D}}, \mathbf{D}_{\mathcal{G}}, \Phi^1_{\mathcal{G}}, \Phi_{\mathcal{G}})$, where $\Delta_{\mathbf{D}}$, which is disjoint with the abstract domain $\Delta^{\mathcal{I}}$, is the datatype domain covering all datatypes, $\mathbf{D}_{\mathcal{G}}$ is a set of base datatype names, $\Phi^1_{\mathcal{G}}$ is a set of derived datatype names and $\Phi_{\mathcal{G}}$ is a set of predicate names. Each base datatype name $d \in \mathbf{D}_{\mathcal{G}}$ is associated with a base datatype $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, each derived datatype name $d' \in \Phi^1_{\mathcal{G}}$ is associated with a derived datatype ${d'}^{\mathbf{D}} \subseteq d^{\mathbf{D}}$, where $d \in \mathbf{D}_{\mathcal{G}}$, and each predicate name $P \in \Phi_{\mathcal{G}}$ is associated with an arity n $(n > 1)$ and a n-ary predicate $P^{\mathbf{D}} \subseteq d_1^{\mathbf{D}} \times \ldots \times d_n^{\mathbf{D}} \subseteq \Delta^n_{\mathbf{D}}$, where $d_1 \ldots d_n \in \mathbf{D}_{\mathcal{G}} \cup \Phi^1_{\mathcal{G}}$.

The domain function $\mathsf{dom}(p, i)$ returns the domain of the i-th argument of the (possibly unary) predicate p, where datatypes can be regarded as unary predicates. According to the above definition, $\mathsf{dom}(p, i)$ is defined as

1. for each $d \in \mathbf{D}_{\mathcal{G}}$, $\mathsf{dom}(d, 1) = \Delta_{\mathbf{D}}$;
2. for each $d' \in \Phi^1_{\mathcal{G}}$, $\mathsf{dom}(d', 1) = d^{\mathbf{D}}$;
3. for each $P \in \Phi_{\mathcal{G}}$, $\mathsf{dom}(P, i) = d_i^{\mathbf{D}}$ $(1 < i \leq n)$ if the arity of P is n.

Let $\mathbf{V}$ be a set of variables. We will consider predicate conjunctions over $\mathcal{G}$ of the form

$$\mathrm{C} = \bigwedge_{j=1}^{k} p_j(v_1^{(j)}, \ldots, v_{n_j}^{(j)}), \tag{3}$$

where p_j is an n_j-ary predicate in $\mathbf{D}_{\mathcal{G}} \cup \Phi^1_{\mathcal{G}} \cup \Phi_{\mathcal{G}}$, and the $v_i^{(j)}$ are variables from $\mathbf{V}$. A predicate conjunction C is called *satisfiable* iff there exists a function δ mapping the variables in C to data values in $\Delta_{\mathbf{D}}$ s.t. $\langle \delta(v_1^{(j)}), \ldots, \delta(v_{n_j}^{(j)}) \rangle \in p_j^{\mathbf{D}}$ for $1 \leq j \leq k$. Such a function δ is called a *solution* for C. A datatype group $\mathcal{G}$ is *conforming* iff

1. $\mathbf{D}_{\mathcal{G}}$, $\Phi^1_{\mathcal{G}}$ and $\Phi_{\mathcal{G}}$ are closed under negation,
2. a binary inequality predicate $\neq_i \in \Phi_{\mathcal{G}}$ is defined for each datatype $d_i \in \mathbf{D}_{\mathcal{G}}$, and
3. the satisfiability problem for finite predicate conjunctions over $\mathcal{G}$ is decidable.

By $\overline{p}$, we denote the negation of the (possibly unary) predicate p, if the arity of p is n $(n \geq 1)$, then $\mathsf{dom}(\overline{p}, 1) = \mathsf{dom}(p, 1), \ldots, \mathsf{dom}(\overline{p}, n) = \mathsf{dom}(p, n)$ and $\overline{p}^{\mathbf{D}} = \mathsf{dom}(p, 1) \times \ldots \times \mathsf{dom}(p, n) \setminus p^{\mathbf{D}}$.

For convenience, we use $\top_{\mathbf{D}}$ as the name of $\Delta_{\mathbf{D}}$ and $\bot_{\mathbf{D}}$ as the name of the negation of $\top_{\mathbf{D}}$. ◇

A datatype group $\mathcal{G}$ is a natural n-ary predicate extension (introducing $\Phi_{\mathcal{G}}$) of the universal concrete domain $\mathbf{D}$, where the set Φ^1 of datatype names is divided into the set $\mathbf{D}_{\mathcal{G}}$ of base datatype names and the set $\Phi^1_{\mathcal{G}}$ of derived datatype names. The division is motivated by having different interpretation settings for their negations. The domain function $\mathsf{dom}(p, i)$ is defined for this purpose, so that the interpretation of the negation of datatypes and datatype predicates can be more intuitive, and do not change when new datatypes are introduced. Here is an example.

Example 6 The $>_5$ and $\leq_5$ Datatype Predicates. *(cont.)* A datatype group $\mathcal{G}$ can be defined as

$$\begin{array}{l}(\Delta_{\mathbf{D}},\\ \mathbf{D}_{\mathcal{G}} := \{INT, \overline{INT}, STRING, \overline{STRING}\},\\ \Phi^1_{\mathcal{G}} := \{>_5, \leq_5\},\\ \Phi_{\mathcal{G}} := \{\neq_{INT}, =_{INT}, \neq_{STRING}, =_{STRING}\}),\end{array}$$

where $\overline{INT}^{\mathbf{D}} = \mathsf{dom}(INT, 1) \setminus INT^{\mathbf{D}} = \Delta_{\mathbf{D}} \setminus INT^{\mathbf{D}}$, and $\overline{>_5}^{\mathbf{D}} = \mathsf{dom}(>_5, 1) \setminus >_5^{\mathbf{D}} = INT^{\mathbf{D}} \setminus >_5^{\mathbf{D}} = \leq_5^{\mathbf{D}}$. Therefore the integer 3 is in $\overline{>_5}^{\mathbf{D}}$ but not in $\overline{INT}^{\mathbf{D}}$, while the string "Fred" is in $\overline{INT}^{\mathbf{D}}$ but not in $\overline{>_5}^{\mathbf{D}}$. ◇

Since the datatype group approach supports n-ary datatype predicates, we can now present the binary predicate kilosPerMile in the miles and kilometers example.

Example 7 Miles and Kilometers. *(cont.)* A datatype group $\mathcal{G}_2$ can be defined as

$$\begin{array}{l}(\Delta_{\mathbf{D}},\\ \mathbf{D}_{\mathcal{G}_2} := \{FLOAT, \overline{FLOAT}\},\\ \Phi^1_{\mathcal{G}_2} := \{lengthInMile, \overline{lengthInMile}, lengthInKMtr, \overline{lengthInKMtr}\},\\ \Phi_{\mathcal{G}_2} := \{kmtrsPerMile, \overline{kmtrsPerMile}, \neq_{FLOAT}, =_{FLOAT}\}),\end{array}$$

where $\overline{kmtrsPerMile}^{\mathbf{D}} = \mathsf{dom}(kmtrsPerMile, 1) \times \mathsf{dom}(kmtrsPerMile, 2) \setminus kmtrsPerMile^{\mathbf{D}} = lengthInKmtr^{\mathbf{D}} \times lengthInMile^{\mathbf{D}} \setminus kmtrsPerMile^{\mathbf{D}}$. ◇

There is a close relationship between a conforming datatype group with only one base datatype and an admissible concrete domain.

Lemma 4 An admissible concrete domain $(\Delta_{\mathcal{D}}, \Phi_{\mathcal{D}})$ is a conforming datatype group $\mathcal{G} = (\Delta_{\mathbf{D}},\ \mathbf{D}_{\mathcal{G}} := \{\mathcal{D}\},\ \Phi^1_{\mathcal{G}}, \Phi_{\mathcal{G}})$, where $\Phi^1_{\mathcal{G}}$ is the set of unary predicate names in $\Phi_{\mathcal{D}}$ and $\Phi_{\mathcal{G}}$ is set of n-ary $(n > 1)$ predicate names in $\Phi_{\mathcal{D}}$, if there exists a binary inequality predicate $\neq_{\mathcal{D}} \in \Phi_{\mathcal{D}}$.

Proof. Immediate consequence of Definition 1 and 3. □

Now we show how two conforming datatype groups $\mathcal{G}_1$ and $\mathcal{G}_2$ can be combined to form a new datatype groups $\mathcal{G}_1 \oplus \mathcal{G}_2$. It turns out (Lemma 6 and 7) that the combination is also conforming in many cases.

Definition 5 Assume that $\mathcal{G}_1$ and $\mathcal{G}_2$ are conforming datatype groups. Then $\mathcal{G}_1 \oplus \mathcal{G}_2$ can be constructed as $(\Delta_{\mathbf{D}},\ \mathbf{D}_{\mathcal{G}_1 \oplus \mathcal{G}_2} := \mathbf{D}_{\mathcal{G}_1} \cup \mathbf{D}_{\mathcal{G}_2},\ \Phi^1_{\mathcal{G}_1 \oplus \mathcal{G}_2} := \Phi^1_{\mathcal{G}_1} \cup \Phi^1_{\mathcal{G}_2},\ \Phi_{\mathcal{G}_1 \oplus \mathcal{G}_2} := \Phi_{\mathcal{G}_1} \cup \Phi_{\mathcal{G}_2})$. ◇

Note that for a predicate p in either $\mathcal{G}_1$ or $\mathcal{G}_2$ $\overline{p}^{\mathbf{D}}$ doesn't changed after the combination.

Lemma 6 If $\mathcal{G}_1$ and $\mathcal{G}_2$ are conforming datatype groups where $\mathbf{D}_{\mathcal{G}_1} \cap \mathbf{D}_{\mathcal{G}_2} = \emptyset$, then $\mathcal{G}_1 \oplus \mathcal{G}_2$ is also a conforming datatype group.

Proof. Obviously $\mathcal{G}_1 \oplus \mathcal{G}_2$ satisfies the first two conditions of a conforming datatype group. Now we only focus on the third condition. Assume that a predicate conjunction

$$\mathrm{C} = \bigwedge_{j=1}^{k} P_j(v_1^{(j)}, \ldots, v_{n_j}^{(j)})$$

is given, where P_j are predicates of $\mathcal{G}_1 \oplus \mathcal{G}_2$.

1. If a variable v occurs as an argument of (possibly unary) predicates from both datatype groups, then C is not satisfiable, because $\mathbf{D}_{\mathcal{G}_1}$ and $\mathbf{D}_{\mathcal{G}_2}$ are disjoint.
2. Otherwise, C can be split into two predicate conjunctions C_1 and C_2 such that they are predicate conjunctions in $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively and no variable occurs in both conjunctions. Therefore, we observe that C is satisfiable iff the satisfiability tests of the respective datatype groups succeed for C_1 and C_2.

□

Note that we don't need to cope with disjunctions in the combination, while in the corresponding Lemma for $\mathcal{D}_1 \oplus \mathcal{D}_2$ (Lemma 2.4 in [2]) in the concrete domain approach, disjunctions must be handled because of type errors.

If the set of datatypes and predicates in a conforming datatype group $\mathcal{G}_1$ is a sub-set of the set of datatypes and predicates in another conforming datatype group $\mathcal{G}_2$, then trivially $\mathcal{G}_1 \oplus \mathcal{G}_2$ is also conforming.

Lemma 7 If $\mathcal{G}_1$ and $\mathcal{G}_2$ are conforming datatype group where $\mathbf{D}_{\mathcal{G}_1} \cup \Phi^1_{\mathcal{G}_1} \cup \Phi_{\mathcal{G}_1} \subseteq \mathbf{D}_{\mathcal{G}_2} \cup \Phi^1_{\mathcal{G}_2} \cup \Phi_{\mathcal{G}_2}$, then $\mathcal{G}_1 \oplus \mathcal{G}_2$ is also a conforming datatype group.

Proof. Immediate consequence of Definition 5, obviously $\mathcal{G}_1 \oplus \mathcal{G}_2$ is equivalent to $\mathcal{G}_2$. □

Lemma 6 and 7 give guidelines on how to build complex conforming datatype groups from simple ones. Based on the definition of a datatype group, we can define a type checker which works with a DL reasoner to answer datatype queries.

Definition 8 A *type checker* is a program that takes as input a finite predicate conjunction C over (one of) the conforming datatype group(s) it supports, and answers *satisfiable* if C is satisfiable and *unsatisfiable* otherwise. ◇

It is possible for a DL reasoner to work with many type checkers. Firstly, in the datatype group approach, the interpretation of $\overline{p}$ ($p \in \mathbf{D}_\mathcal{G} \cup \Phi^1_\mathcal{G} \cup \Phi_\mathcal{G}$) doesn't change when new datatypes are introduced, so the interpretation of each $\overline{p}$ supported by a type checker won't be affected by the existence of other type checkers. Secondly, assuming that the set of base datatypes of the conforming datatype group supported by each type checker is disjoint from each other, Lemma 6 shows that the combined datatype group of all these datatype groups supported by the type checkers is also conforming.

4 $\mathcal{SHOQ}(\mathbf{D_n})$

In this section, we give the definition of the $\mathcal{SHOQ}(\mathbf{D_n})$ DL that supports reasoning with datatype groups. Note that in DLs we talk about *concepts* and *roles* where in Web ontology languages we usually talk about *classes* and *properties*.

Definition 9 *($\mathcal{SHOQ}(\mathbf{D_n})$ Syntax and Semantics.)* Let $\mathbf{C}$, $\mathbf{R} = \mathbf{R}_A \uplus \mathbf{R_D}$, $\mathbf{I}$ be disjoint sets of concept, abstract and concrete role and individual names. For R and S roles, a *role axiom* is either a role inclusion, which is of the form $R \sqsubseteq S$ for $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R_D}$, or a transitivity axiom, which is of the form $\mathsf{Trans}(R)$ for $R \in \mathbf{R}_A$. A *role box* $\mathcal{R}$ is a finite set of role axioms. A role R is called *simple* if, for $\sqsubseteq^*$ the transitive reflexive closure of $\sqsubseteq$ on $\mathcal{R}$ and for each role S, $S \sqsubseteq^* R$ implies $\mathsf{Trans}(S) \notin \mathcal{R}$.

The set of concept terms of $\mathcal{SHOQ}(\mathbf{D_n})$ is inductively defined. As a starting point of the induction, any element A of $\mathbf{C}$ is a concept term (atomic terms). Now let C and D be concept terms, o be an individual, R be a abstract role name, $T_1, \ldots, T_n$ be concrete role names, S be a simple role name, P be an n-ary datatype predicate name. Then complex concepts can be built using the operators shown in Figure 2.

The semantics is defined in terms of an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ (the abstract domain) is a nonempty set and and $\cdot^\mathcal{I}$ (the interpretation function) maps atomic and complex concepts, roles and nominals according to Figure 2. Note that $\sharp$ denotes set cardinality, $\Delta_\mathbf{D}$ is the datatype domain and $\mathsf{dom}(P, i)$ is the domain function in a datatype group.

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ satisfies a role inclusion axiom $R_1 \sqsubseteq R_2$ iff $R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$, and it satisfies a transitivity axiom $\mathsf{Trans}(R)$ iff $R^\mathcal{I} = (R^\mathcal{I})^+$. An interpretation satisfies a role box $\mathcal{R}$ iff it satisfies each axiom in $\mathcal{R}$. A $\mathcal{SHOQ}(\mathbf{D_n})$-concept C is satisfiable w.r.t. a role box $\mathcal{R}$ iff there is an interpretation $\mathcal{I}$ with $C^\mathcal{I} \neq \emptyset$ that satisfies $\mathcal{R}$. Such an interpretation is called a *model* of C w.r.t. $\mathcal{R}$. A concept C is *subsumed* by a concept D w.r.t. $\mathcal{R}$ iff $C^\mathcal{I} \sqsubseteq D^\mathcal{I}$ for each interpretation $\mathcal{I}$ satisfying $\mathcal{R}$. Two concepts are said to be equivalent (w.r.t. $\mathcal{R}$) iff they mutually subsume each other (w.r.t. $\mathcal{R}$). ◇

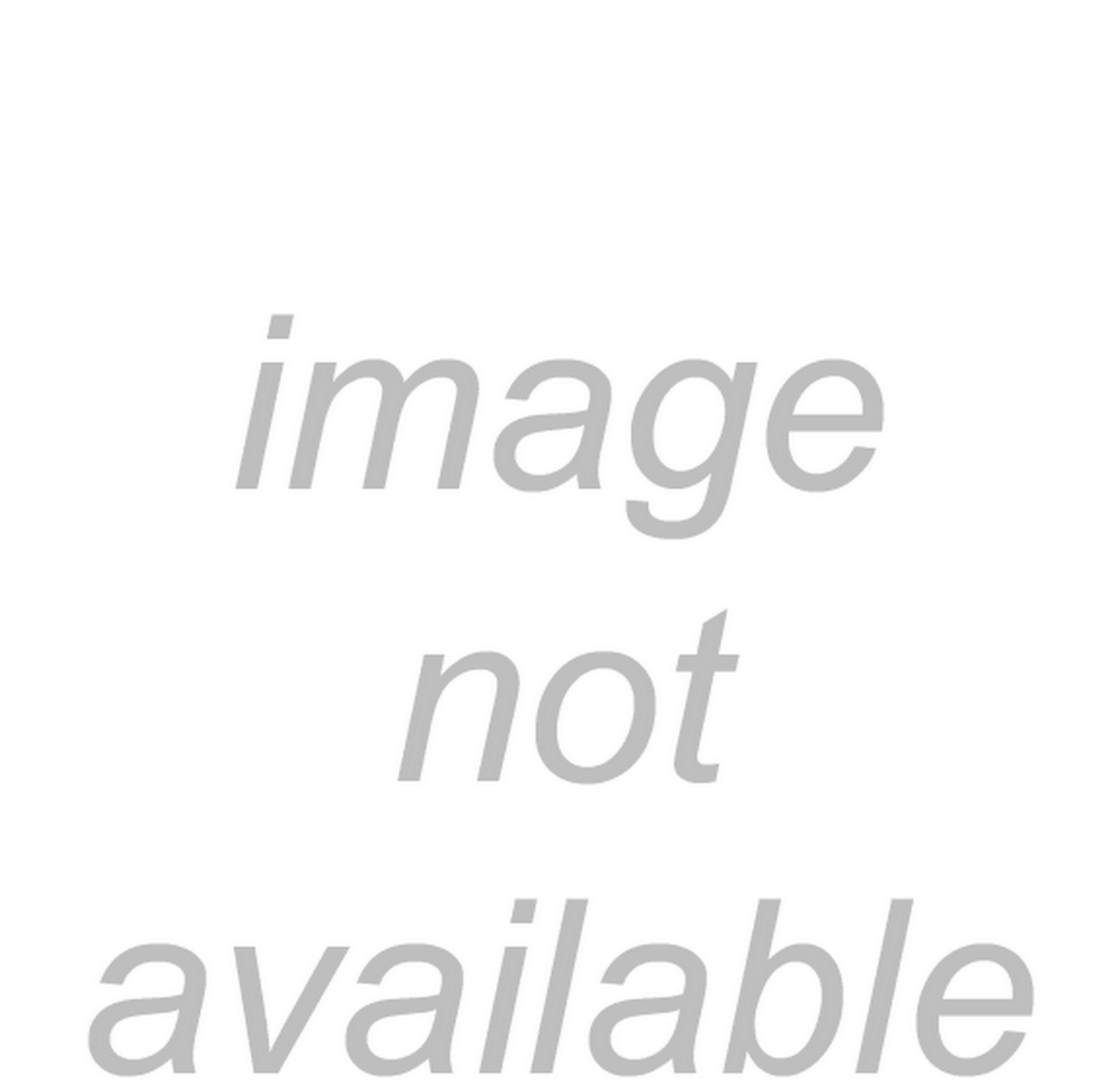


image
not
available

reader is referred to the online technical report[13] [13] for full details and proofs of the algorithm's soundness and completeness.

As with any tableau algorithm, the basic idea is to try to prove the satisfiability of a concept C (w.r.t. a role box $\mathcal{R}$) by building a model of C, i.e., (a structure that closely corresponds to) an interpretation $\mathcal{I}$ that satisfies $\mathcal{R}$ and in which $C^{\mathcal{I}}$ is not empty. The algorithm works on a (set of) tree(s), where nodes are labeled with sets of sub-concepts of C, and edges are labeled with sets of roles occurring in C. Nodes (edges) in the tree correspond to elements (tuples) in the interpretation of the concepts (roles) with which they are labeled. Normally, a single tree is initialised with a root node labeled $\{C\}$.

The algorithm exhaustively applies tableau rules that decompose the syntactic structure of the concepts in node labels, either expanding node labels, adding new edges and nodes to the tree(s), or merging edges and nodes. The application of a rule effectively explicates constraints on the interpretation implied by the concepts to which the rule was applied. E.g., if $A \sqcap B$ is in the label of a node x then the $\sqcap$-*rule* adds both A and B to the label, explicating the fact that if $x \in (A \sqcap B)^{\mathcal{I}}$, then both $x \in A^{\mathcal{I}}$ and $x \in B^{\mathcal{I}}$. Similarly, if $\exists R.A$ is in the label of a node x, then the $\exists$-*rule* adds a new node y labelled $\{A\}$ with an edge between x and y labeled $\{R\}$, explicating the fact that if $x \in (\exists R.A)^{\mathcal{I}}$, then there must exist a node y such that $\langle x, y\rangle \in R^{\mathcal{I}}$ and $y \in A^{\mathcal{I}}$.

An attempt to build a model fails if an obvious contradiction, often called a *clash*, is generated, e.g., if the label of some node contains both D and $\neg D$ for some concept D; it is successful if no more rules can be applied, and there are no clashes. It is relatively straightforward to prove that a concept is satisfiable if and only if the rules can be applied in such a way that a model is successfully constructed. The computational complexity of the algorithm stems from the fact that some rules are non-deterministic (e.g., the rule dealing with disjunctions); in practice this is dealt with by backtracking when a clash is detected, and trying a different non-deterministic rule application.

Various refinements of this basic technique are required in order to deal with a logic as expressive as $\mathcal{SHOQ}(\mathbf{D_n})$. In the first place, the algorithm operates on a forest of trees, as an additional tree must be constructed for each nominal in C (see Figure 2); a form of cycle check called *blocking* must also be used in order to guarantee termination [6]. In the second place, the algorithm needs to use a type checker to check constraints on the interpretation derived from datatype exists, value, atleast and atmost concepts.

5.1 Datatype Reasoning

Logics like $\mathcal{SHOQ}(\mathbf{D})$ and $\mathcal{SHOQ}(\mathbf{D_n})$ are designed so that reasoning about datatypes and values can be separated from reasoning about concepts and roles – this is the reason for the strict separation of the domains and of abstract and concrete roles. The result of the separation is that node labels can contain either concepts or datatypes and values, but never a mixture. This allows node labels

[13] `http://DL-Web.man.ac.uk/Doc/shoqdn-proofs.pdf`

containing datatypes and values to be checked using a separate type checker, with inconsistencies in such labels being treated as an additional clash condition. E.g., if a node label includes the concepts $\exists T.\texttt{string}$ and $\forall T.\texttt{real}$, then a new concrete node will be generated labeled string, real, and when checked with the type checker this would (presumably) return a clash on the grounds that there is no element that is in the interpretation of both `string` and `real`.

With $\mathcal{SHOQ}(\mathbf{D_n})$ the situation is more complex because it is necessary to deal with both n-ary predicates and datatype cardinality constraints that may be qualified with n-ary predicates. The algorithm deals with n-ary predicates by keeping track of tuples of concrete nodes that must satisfy datatype predicates, and it deals with datatype cardinality constraints by keeping track of inequalities between tuples of concrete nodes that were generated by the datatype $\geqslant_P -rule$ in order to explicate datatype atleast concepts (merging such tuples could lead to non-termination as the $\geqslant_P -rule$ might be applied again and cause new tuples to be generated).

The predicate relationships between (the values represented by) concrete nodes are taken into consideration by the type checker, which can check predicate conjunctions (see Definition 8). In the algorithm described in [12], the type checker must also ensure that the solution is consistent with inequalities between tuples of concrete nodes. This means that it must be extended to deal with non-deterministic reasoning, because $\langle t_{j1}, \ldots, t_{jn}\rangle \dot{\neq} \langle t_{k1}, \ldots, t_{kn}\rangle$ is equivalent to

$$(t_{j1} \dot{\neq} t_{k1}) \cup \ldots \cup (t_{jn} \dot{\neq} t_{kn}) \tag{4}$$

In the new algorithm, this non-deterministic reasoning is pushed back into the tableau reasoner, which is already able to cope with non-determinism arising, e.g., from disjunction concepts. This is done by adding concepts "equivalent" to (4) to the node labels containing the relevant datatype atleast concept. For this purpose, we use concepts of the form

$$(\forall T_1^{C_{j1}}, T_1^{C_{k1}} . \neq_{d_{P_n,i}}) \sqcup \ldots \sqcup (\forall T_n^{C_{jn}}, T_n^{C_{kn}} . \neq_{d_{P_n,i}}), \tag{5}$$

where $C = \ \geqslant mT_1, \ldots, T_n.P_n$, is the datatype atleast concept in question, $T_1^{C_{j1}}, T_1^{C_{k1}}, \ldots, T_n^{C_{jn}}, T_n^{C_{kn}}$ are *superscripted concrete roles* and $\neq_{d_{P_n,i}}$ is the inequality predicate for the datatype $\mathsf{dom}(P_n, i)$ (recall that each datatype in a datatype group must be equipped with an inequality predicate). The superscripted concrete roles are generated by the $\geqslant_P -rule$ and used to link the node x containing C to the new concrete nodes that the rule generates. The form of the superscript means that a superscripted role acts as a unique (w.r.t. the node x) name for a given concrete node. The result is that if ever two tuples created by applying the $\geqslant_P -rule$ to C are merged, then the type checker will return a clash. This is because, whichever way the $\sqcup - rule$ is applied to the disjunction (5), the predicate relationships will include $x \neq x$ for some concrete node x.

6 Discussion

As we have seen, using datatypes within Semantic Web ontology languages (such as DAML+OIL and OWL) presents new requirements for DL reasoning services. We have presented the datatype group approach, which extends the type system approach with n-ary predicates and a new treatment of predicate negation, so as to make it possible to use type checkers with DL reasoners. We have also sketched an improved algorithm for reasoning with $\mathcal{SHOQ}(\mathbf{D_n})$ using datatype groups. Type checkers for datatype groups should be easy to implement as we do not have to deal with disjunctions of predicate terms. Moreover, the similarity of conforming datatype groups and admissible concrete domains can be exploited in order to identify suitable datatype groups.

The resulting framework is both robust and extensible. On the one hand, most complex reasoning tasks take place within the well understood and provably correct tableau algorithm. On the other hand, it is relatively easy to add support for new datatypes and predicates, and this does not require any changes to the tableaux algorithm itself. An implementation of the algorithm along with a simple type checker (supporting integers and strings) is currently underway (based on the FaCT system), and will be used to evaluate empirical performance.

Although existing Web ontology languages such as DAML+OIL and OWL do not support n-ary predicates, we believe that they are useful/essential in many realistic applications, and will be a prime candidate for inclusion in future extensions of these languages. The algorithm we have presented could be used to provide reasoning support for such extended Web ontology languages.

Acknowledgements. We would like to thank Ulrike Sattler, since the work presented here extends the original work on $\mathcal{SHOQ}(\mathbf{D})$. Thanks are also due to Carsten Lutz for his helpful discussion on inequality predicates.

References

[1] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider, editors. *Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002.

[2] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[3] Paul V. Biron and Ashok Malhotra. Extensible Markup Language (XML) Schema Part 2: Datatypes – W3C Recommendation 02 May 2001. Technical report, World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/xmlschema-2/`.

[4] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommentdation, `URL http://www.w3.org/TR/rdf-schema`, Mar. 2000.

[5] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[6] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, pages 161–180, 1999.

[7] Ian Horrocks and Peter F. Patel-Schneider. The generation of DAML+OIL. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 30–35. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-49/`, 2001.

[8] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[9] C. Lutz. Description logics with concrete domains – a survey. In *Advances in Modal Logics Volume 4*. World Scientific Publishing Co. Pte. Ltd., 2002.

[10] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[11] Jeff Z. Pan. Web Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D_n})$ Description Logic. In *Carlos Areces and Maartin de Rijke, editors,Proceedings of the Methods for Modalities 2 (M4M-2)*, Nov 2001. ILLC, University of Amsterdam.

[12] Jeff Z. Pan and Ian Horrocks. Reasoning in the SHOQ(Dn) Description Logic. In Ian Horrocks and Sergio Tessaris, editors, *Proc. of the 2002 Int. Workshop on Description Logics (DL-2002)*, Apr. 2002.

[13] Jeff Z. Pan and Ian Horrocks. Reasoning in the $\mathcal{SHOQ}(\mathbf{D_n})$ Description Logic (Online Proofs), 2003.

Merging Topics in Well-Formed XML Topic Maps

Richard Widhalm and Thomas A. Mueck

University Of Vienna
Institute for Computer Science and Business Informatics
Rathausstr. 19
1010 Vienna, Austria
rwidhalm@gmx.at
mueck@ifs.univie.ac.at

Abstract. Topic Maps are a standardized modelling approach for the semantic annotation and description of WWW resources. They enable an improved search and navigational access on information objects stored in semi-structured information spaces like the WWW. However, the according standards ISO 13250 and XTM (XML Topic Maps) lack formal semantics, several questions concerning e.g. subclassing, inheritance or merging of topics are left open. The proposed TMUML meta model, directly derived from the well known UML meta model, is a meta model for Topic Maps which enables semantic constraints to be formulated in OCL (object constraint language) in order to answer such open questions and overcome possible inconsistencies in Topic Map repositories. We will examine the XTM merging conditions and show, in several examples, how the TMUML meta model enables semantic constraints for Topic Map merging to be formulated in OCL. Finally, we will show how the TM validation process, i.e., checking if a Topic Map is well formed, includes our merging conditions.

1 Introduction

The Topic Maps standard ISO 13250 [7] as well as XTM (XML Topic Maps) [12] provide for the semantic characterization of information objects across the WWW or company-controlled intranet or extranet platforms. They externally describe the underlying information objects (documents, web pages, etc.) without changing them. With Topic Maps, which are syntactically based on XML (or SGML for ISO13250), semantic networks consisting of topics and their relationships can be built, thus enhancing the flexibility of search queries and navigational access on the underlying information objects. They can be used as a flexible, generic index for knowledge bases. In a topic map, every real world subject (equal if a WWW resource or an abstract thing) is represented by a *topic*. Each topic may be an instance of several *topic types* (which are also topics), may have several lexical *names* in different *scopes* (where a scope is the area where the according name for the topic is said to be valid and each scope

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 64–79, 2003.

is described by a set of topics) and may show several *occurrences* in different WWW resources. An occurrence is a link, typed by a topic, to an information object. Topics can be interrelated via n-ary *associations*, where each topic plays a certain *role* which is again expressed by another topic. Associations can be typed via a topic. Further, it is possible to create generalization hierarchies for topic types, association types and occurrence types. Occurrences, names and associations can be placed within a scope. Each topic T has an *identificator*, which can be a WWW resource or another topic, indicating or itself being the subject for which T stands. Furthermore, Topic Maps define two topics T_1 and T_2 as identical, if they are identified by the same resource or if they both exhibit the same name N in the same scope S. In that case, they shall be merged to a single topic bearing all the characteristics of the original topics. In this paper, we will focus on the issue of merging topics. We will show how the TMUML meta model lets one formulate the actual XTM merging conditions with the OCL (*object constraint language*), defined within the UML specification [9]. We will also show specific circumstances under which merging would not be desirable, although the XTM criteria would be met. Therefore, we will inspect the consequences of the merging of two topics of a different kind (like an association type merged with an occurrence type). Consequently, we present suggestions for additional OCL constraints showing how the TMUML meta model can help to overcome such issues. Of course, the presented constraints are only considered as suggestions that may further be extended or only partially applied according to specific needs. The following section briefly introduces the TMUML meta model, succeeded by an overview about related work. Afterwards, we will describe the synonymy relation used for merging in the TMUML meta model. Chapter 5 expresses the XTM merging rules in OCL and adds a further, implicated merging rule. Afterwards, we will inspect situations of merging different kinds of topics (like association types, occurrence types, topic types or topic instances) and show the processing steps within our XTM validation and the functionality of the OCL checker. The conclusion will point out the meaningfulness of merging in largely designed Topic Maps systems and give a prospect on our further work.

2 Topic Maps and the TMUML Meta Model

In [19] and [20], the TMUML meta model has been presented, including a detailed description as well as the according class diagrams. It is a meta model for Topic Maps, directly derived from the UML meta model. It does not make use of all of the UML meta model components, only those which are also relevant for Topic Maps (like `Association` or `GeneralizableElement`). Note that topics are divided into the meta classes `TopicType` (for topics used as types for other topics), `TopicObject` (topics not functioning as types but as instances of types), `AssociationTopicType` (similar to the UML `AssociationClass`, representing types for association instances) and `OccurrenceType` (types for occurrences). The `TopicAssociationEnd` defines, like the `AssociationEnd` in UML, the allowed `TopicType` for a role within the according `TopicAssociation`, and also

It makes use of the operation `identificator()`, which is defined in the context of `Topic` and retrieves all `Identificators` that are related to the context `Topic` via an `Identification` relationship in the TMUML meta model. An `Identificator` may be a subject constituting resource, a subject indicating resource or another topic (see [12]).

```
identificator() : Set(Identificator) =
 self.identifiedBy->collect(i : Identification | i.identificator)->asSet
```

3 Related Work

An early work on constraining Topic Maps is [14], where several suggestions for constraints at the instance layer of a topic map are given (without a formal method). An overview about the idea behind constraints for Topic Maps in general can be found in [6]. In [11], Ontopia describes their own solution for a Topic Maps constraint language, called OSL (*Ontopia Schema Language*). It allows for a more constrained and precisely defined Topic Maps schema definition, but again is not usable for constraining the Topic Maps meta model. With OSL, the types and roles allowed within certain association types may be specified, but, for example, the effects of subclassing association types in general are not treated (the semantics of inheritance). Our method is best suitable for solving the second kind of problem, while it may also be used for the first kind. A similar approach is sketched by the TMCL [13], the *Topic Maps Constraint Language*, although it has not yet surpassed the level of a requirements suggestion. [2] presents a formal model for Topic Maps, consisting only of topics and associations, which are contained in a *homogenous* hypergraph. Meta associations like instancing, subclassing or scoping are contained in a *shift* hypergraph, that compounds components of the homogenous hypergraph and thus establishes semantic layers (e.g. separating role types, topic types or association types). Although this approach applies formal semantics to Topic Maps using hypergraphs, it does not mention additional constraints on the Topic Maps meta model or how they can be applied. Our approach goes beyond that and provides means for the automation of semantic constraints using OCL and USE (or comparable OCL tools). Moreover, the hypergraph model does only distinguish between topics and associations, while names, occurrences or resources have to be modelled by topics. Actually, names have different properties compared to topics in the Topic Maps meta model. Names can not be merged or have occurrences or have names etc. Further, there may be difficulties due to the fact that the semantics of the associations in the shift hypergraph (denoted as θ) is only determined by the semantic layers of the connected topics. For example, there may be different meanings of connections from a role type to a topic type, one may mean a "subclass" relation, the other may mean a type constraint for the topic members playing the role type in the according association type.

Additional information on Topic Maps can be found in [8], [18], [1] and [3]. The first gives an introduction to the fundamental concepts of Topic Maps and introduces a system architecture for applications based on a distributed

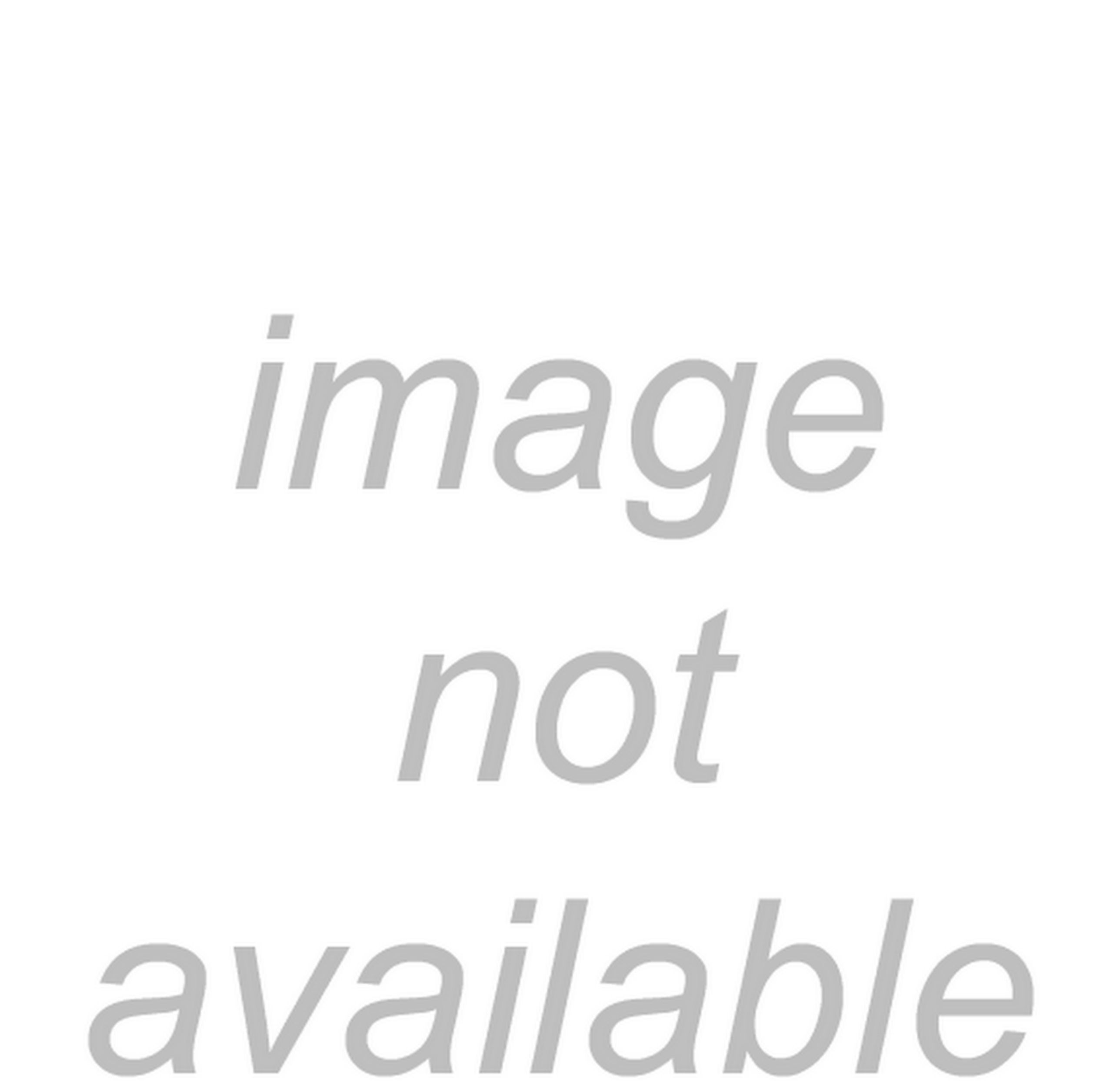


image
not
available

containing T and the `Topics` returned by recursively calling `synonymesRec` (which takes S extended by T) for all `Topics` directly connected to T via the `Synonymy` relation.

```
Topic::synonymesRec(s : Set(Topic)) : Set(Topic) =
 Set{self}->union (
  self.topicOf.synonymeFor->select (f |
   s->excludes(f)).synonymesRec (s->including(self))
  )->union (
  self.topicFor.synonymeOf->select (f |
   s->excludes(f)).synonymesRec (s->including(self))
  )->asSet
```

Thus, by calling `T.closure()`, we receive T and all other `Topics` that are synonymous to T. Further, to determine whether two `Topics` are equal, we can not simply compare the two `Topics`, we have to find out if one is contained within the closure of the other. We therefore provide the operation `Topic::equals()`, which is defined as follows.

```
Topic::equals(t : Topic) : Boolean =
 self.closure()->exists (c_t | c_t = t)
```

5 Merging Conditions

XTM defines two merging conditions. The *naming constraint-based merge* [12] says that two `Topics` T_1 and T_2 have to be merged if T_1 has a base name N_1 in the scope S_1, and T_2 has a base name N_2 in scope S_2, and N_1 and N_2 are equal strings and S_1 and S_2 are equal sets of `Topics`. We define this condition as an OCL operation called `Topic::nameBasedMerge(b: Topic)`, which checks whether the context `Topic` may be merged with b due to the naming constraint-based merge.

```
Topic::nameBasedMerge(b : Topic) : Boolean =
 (self.closure().nameForTopic->exists (s_n |
  b.closure().nameForTopic->exists (b_n |
   s_n.value = b_n.value and s_n.equalScope (b_n))))
```

Here, `ScopableElement::equalScope (s: ScopableElement)` is used to compare the scopes of two `ScopableElements`, which are names in our case (`Occurrences` and `Associations` are also `ScopableElements`. It uses two other operations, which are also defined within the context of `ScopableElement`.

```
ScopableElement::scope() : Set(ScopingElement) =
 self.scopedBy.scope->asSet
ScopableElement::compareScope (other: ScopableElement) : Boolean =
 self.scope()->forAll (
  self_t | other.scope()->select(
   other_t | other_t.oclAsType(Topic).equals(self_t.oclAsType(Topic))
  )->size = 1)
ScopableElement::equalScope (s : ScopableElement) : Boolean =
 self.compareScope (s) and s.compareScope (self)
```

Note that `Topic.nameForTopic` yields all `TopicNames` for a `Topic`, `TopicName.value` is the lexical represenation of a name, and `ScopableElement.scope()` yields the set of `Topics` (which are `ScopingElements`) making up the scope of a `ScopableElement`.

The other XTM merging condition is called *subject-based merge* [12] and says that two `Topics` T_1 and T_2 have to be merged if T_1 has an `Identificator` I_1, and T_2 an `Identificator` I_2, where the URIs of I_1 and I_2 are equal or $T_1 = I_2$ or $T_2 = I_1$. Note that also in this operation we have to excessively use `Topic::closure()`.

```
Topic::subjectBasedMerge(b : Topic) : Boolean =
 (self.closure().identificator()->select (i |
  i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->includes (b))
 or (self.closure().identificator()->exists (
   s_i | b.closure().identificator()->exists (
    b_i | b_i.URI = s_i.URI ))
  )
 or (self.closure().identificator()->select (i |
   i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->exists (
    s_i | b.closure().identificator()->select (i |
     i.oclIsTypeOf (Topic)).oclAsType(Topic).closure()->exists (
      b_i | b_i.URI = s_i.URI ))
  )
```

Further, note that the determination of URI equality is not discussed in this paper and therefore, for the time being, simplified by comparing the URI strings. Instead, the information objects that can be reached by the specific URIs should be compared, and also time (timestamps) may play an important role. See also [15] for more details on URIs.

We will call two `Topics` T_1 and T_2, which meet one of the two XTM merging conditions, *candidates for merging*, as they should be merged according to XTM, but this merging and its consequences has to be examined and possibly prohibited due to some resulting complications.

A situation where merging is still not possible, even if one of the aforementioned conditions would be met, is, when the two candidate `Topics` T_1 and T_2 both have a `SubjectConstitutingResource` (which means, the resource is the subject itself) as `Identificator` - call them SCR_1 and SCR_2 - and SCR_1 is different from SCR_2 (they have different URIs). In XTM, only one `SubjectConstitutingResource` is possible for a `Topic`, which is quite intuitive. Fig. 3 shows this issue schematically. There, T_1 is synonymous to T_2 due to having an equal base name in the same scope $\{T_s\}$. Because they both exhibit a `SubjectConstitutingResource` as `Identificator`, differing from each other, the OCL checker has to raise an error.

We have to check this situation in our validation process and therefore suggest an appropriate operation called `Topic::noDifferentSCR (b:Topic)`. It uses an operation called `Topic::ownSCRIdentificators()`, which is defined in the following.

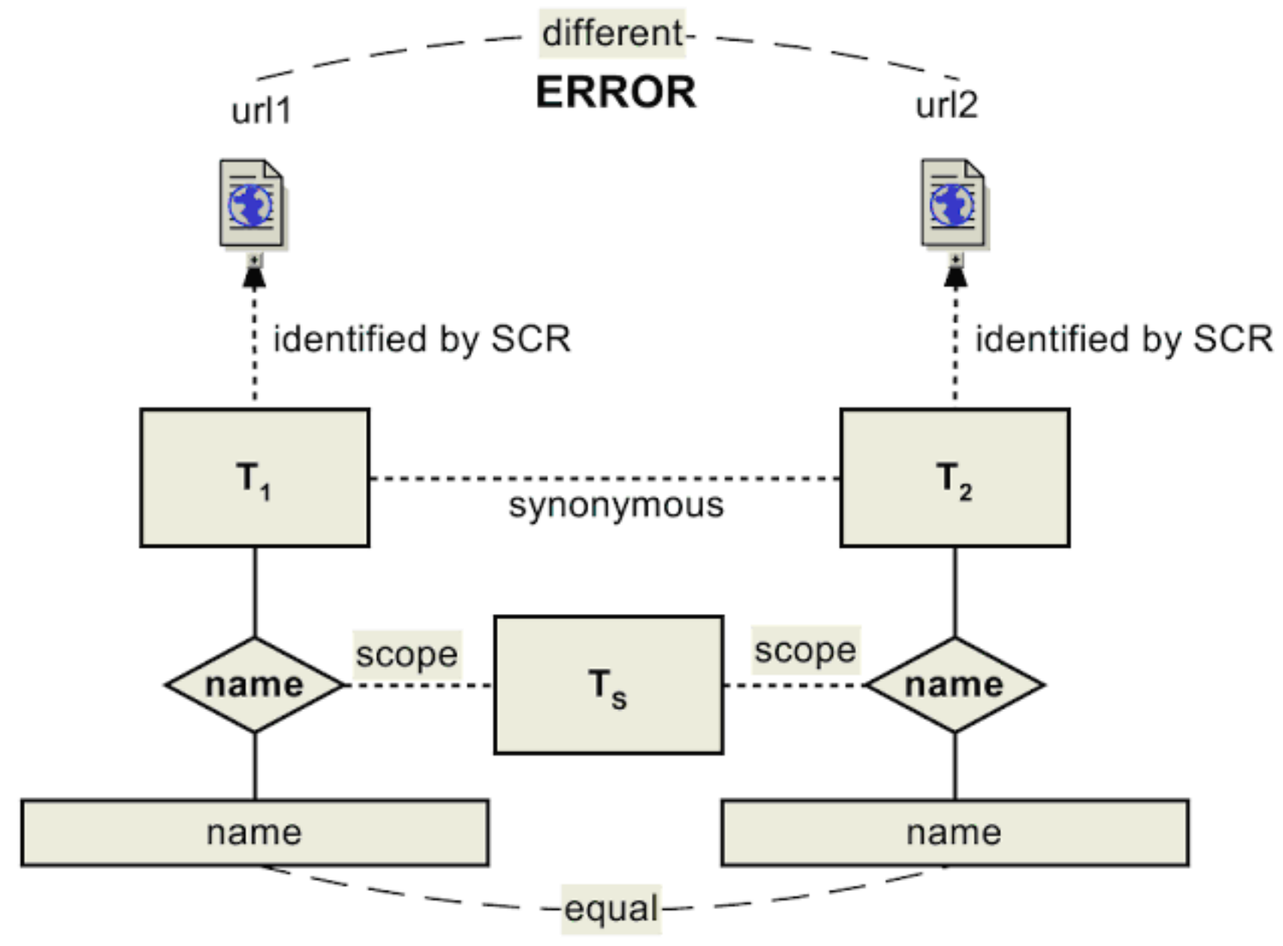


Fig. 3. Different SubjectConstitutingResources as Identificator

```
Topic::noDifferentSCR(b : Topic) : Boolean =
 not self.ownSCRIdentificators()->exists (i_self |
  b.ownSCRIdentificators()->excluding(i_self)->exists (i_b |
   i_self.URI != i_b.URI ))

Topic::ownSCRIdentificators () : Set (SubjectConstitutingResource) =
 self.closure().identificator()->select (i |
  i.oclIsTypeOf(SubjectConstitutingResource)).
   oclAsType(SubjectConstitutingResource)->asSet
```

6 Merging Different Kinds of Topics

What is yet unconsidered in the merging criteria is the fact that the two `Topics` T_1 and T_2, which are candidates for merging due to one of the merging conditions, can be of a different kind. The possible kinds are: `TopicType (TT)`, `TopicObject (TO)`, `AssociationTopicType (AT)` and `OccurrenceType (OT)`. We will use the abbreviations (in brackets) for the four kinds of `Topics`.

In the following, we will examine many of the possible combinations (refer to our further work for a full description of all combinations, including an extensive description of the combination AT - AT) and find out for any combination, if merging shall be prohibited due to the possibility of a semantic inconsistence or not.

6.1 TT - TT, TO - TO, OT - OT

First, we will examine the merging of two `TopicTypes` TT_1 and TT_2. A `TopicType` TT_1 may define a `TopicAssociationEnd` of an AT, where the `multiplicity_min` (the lower bound of the multiplicity) is greater than 0. Look at the example in fig. 4. Some constraint, which will not be shown in this work

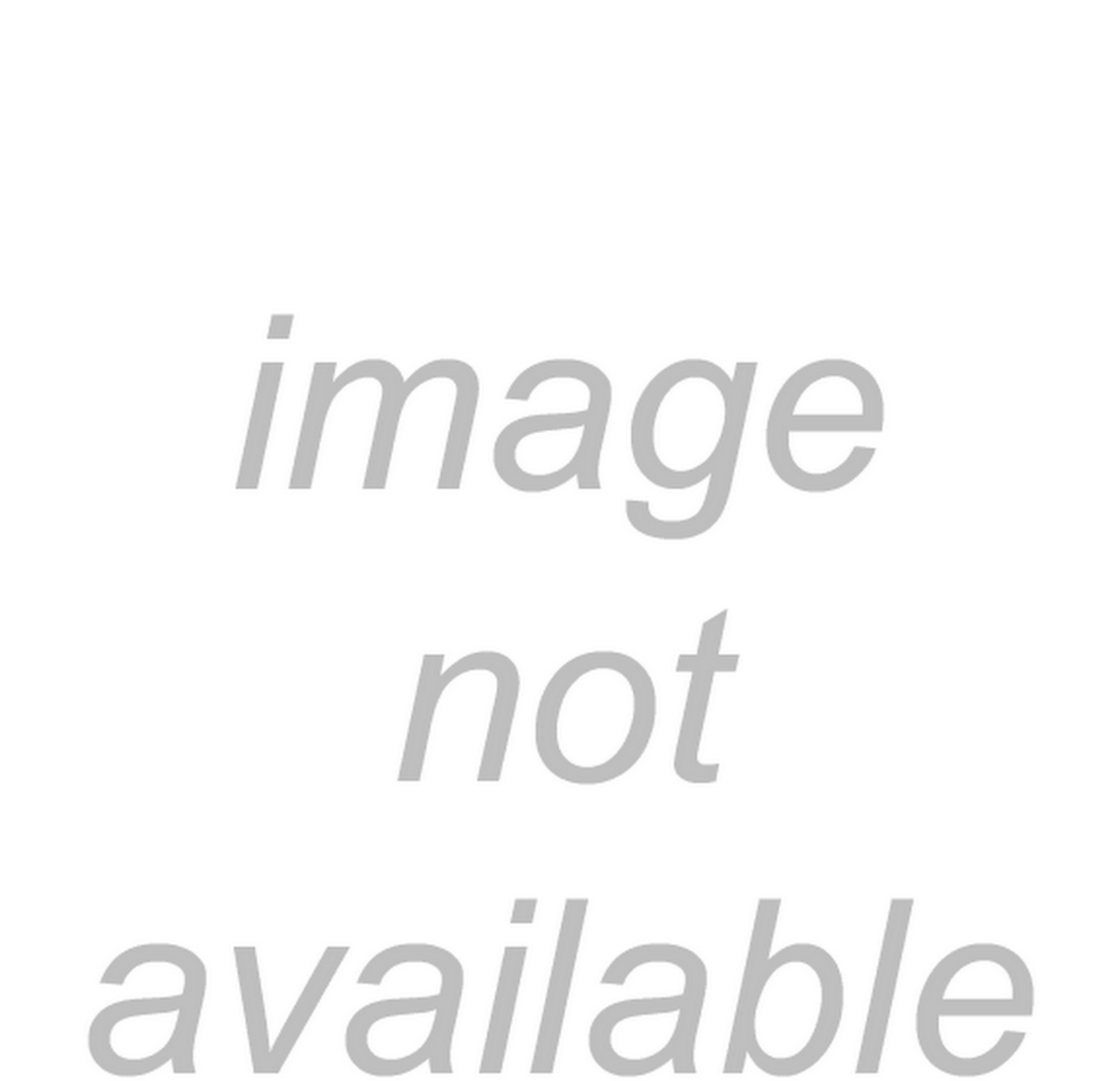


image
not
available

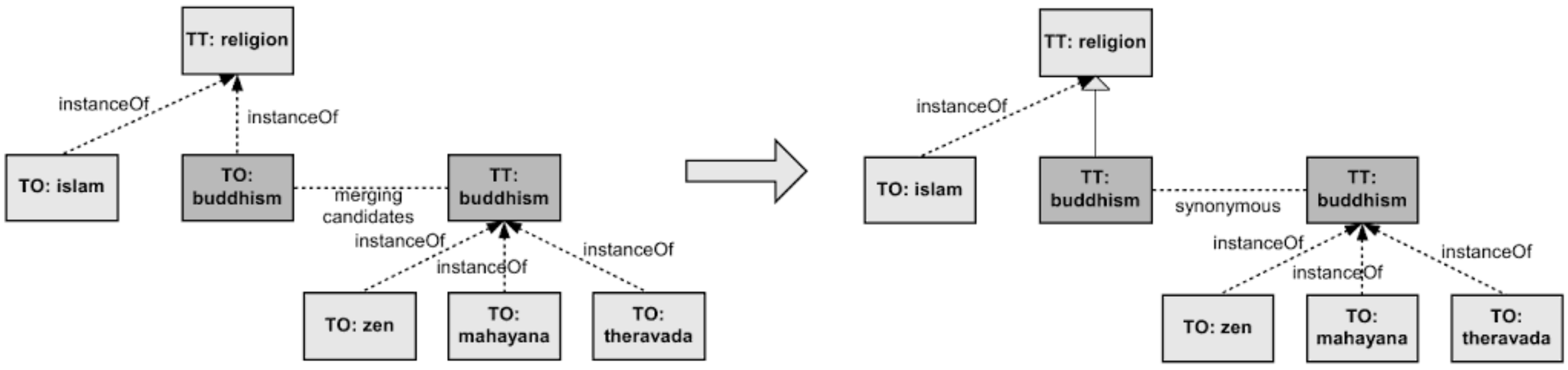


Fig. 6. Merging TT - TO

At the moment, there is no mechanism to constrain an OT to be only applicable to several TTs (like the `TopicAssociationEnds` do for ATs). Therefore, the merging of two `OccurrenceTypes` OT_1 and OT_2 should be no problem.

6.2 TT - TO, AT - TO, OT - TO

When merging two `Topics` of a different kind, the question arises whether the resulting `Topics` should be of the one kind or of the other. Similarly, when establishing a `Synonymy` relation in our TMUML meta model for two different kinds of `Topics`, we have to agree upon one kind for both `Topics`. In the case of merging TT_1 with TO_2, it is not possible to change TT_1 to TO_1, because TT_1 could have subtypes and may be used in `TopicAssociationEnds` as type constraints. But changing TO_2 to TT_2 would be an appropriate measurement, requiring every class-instance relation from TO_2 to some TT_{2super} to be transformed by the merging operation to a subtype relation from the resulting TT_2 to the original TT_{2super}. This works, because a TO does not have any further capabilities in the TMUML meta model than a TT has, and a TT may also play a role within an association instance (see also [19]). Fig. 6 shows an example. There, the `Topics` "TT:buddhism" and "TO:buddhism" are candidates for merging. "TO:buddhism" has to be converted to "TT:buddhism", and the class-instance relation between "TO:buddhism" and "TT:religion" is converted to a subtype relation.

Note that also in the TT-TO case, a *UniqueRolesProblem* may occur after merging. The following OCL operation is called `Topic::changeToTT (t: TopicType)`, where the context `Topic` is the TO, t is the TT and `TopicObject._class` is a TMUML relation to the TTs of the context TO.

```
Topic::changeToTT(t : TopicType)
 pre changeToTT_pre:
  self.oclIsTypeOf (TopicObject)
  and self.closure()->includes(t)
 post changeToTT_post:
  self.oclIsTypeOf (TopicType) and
  self.oclAsType(TopicType).parent().oclAsType(TopicType)->
asSet->includesAll(
    self@pre.closure()->excluding(t).oclAsType(TopicObject)._class)
```

3. Baird, C.: Topic Map Cartography - a discussion of Topic Map authoring. Proceedings XML Europe 2000. GCA, Paris (2000)
4. Gogolla, M., Richters, M.: On constraints and queries in UML. In: Schader, M., Korthaus, A. (eds.): The Unified Modeling Language - Technical Aspects and Applications, p. 109-121. Physica, Heidelberg (1998)
5. Gogolla, M., Richters, M.: Development of UML Descriptions with USE. In: Tjoa A.M., Shafazand, H., Badie K. (eds.): Proc. 1st Eurasian Conf. Information and Communication Technology (EURASIA'2002). LNCS, Springer Verlag, Berlin Heidelberg New York (2002)
6. Gronmo, G.O.: Creating semantically valid topic maps. Proceedings XML Europe 2000. GCA, Paris (2000)
7. ISO/IEC 13250: Information technology - SGML Applications - Topic Maps. International Organization for Standardization, Geneve, Switzerland (1999)
8. Mueck, T.A., Widhalm, R.: Schlagwort - Topic Maps. Wirtschaftsinformatik, 3:297-300. Verlag Vieweg, Wiesbaden (2001)
9. Object Management Group: OMG UML Specification Version 1.3. http://www.omg.org./uml/, last visited 1. 8. 2002
10. Object Management Group: XMI XML Metadata Interchange, Version 1.1. http://cgi.omg.org/docs/ad/99-10-02.pdf, last visited 2. 8. 02
11. Ontopia: The Ontopia Schema Language Reference Specification, Version 1.3. http://www.ontopia.net/omnigator/docs/schema/spec.html, last visited 23. 10. 2002
12. Pepper, S., Moore, G. et al: XML Topic Maps (XTM) 1.0. Topic Maps Authoring Group (2001). http://www.topicmaps.org/xtm/1.0/, last visited 19. 6. 2001
13. Pepper, S.: Draft requirements, examples and a "low bar" proposal for Topic Map Constraint Language. ISO/IEC JTC 1/SC34/WG3 (2001). http://www.y12.doe.gov/sgml/sc34/document/0226.htm, last visited 1. 8. 2002
14. Rath, H.H.: Technical Issues on Topic Maps. Proceedings MetaStructures 99. GCA, Alexandria, VA (1999)
15. IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax. IETF (1998). http://www.ietf.org/rfc/rfc2396.txt, last visited 17. 1. 2003
16. Richters, M., Gogolla, M.: On formalizing the UML object constraint language OCL. In: Tok-Wand, L. (ed.): Proc. 17th Int. Conf. Conceptual Modeling (ER'98), p. 449-464. LNCS Vol. 1507, Springer Verlag, Berlin Heidelberg New York (1998)
17. Warmer, J., Kleppe, A.: The object constraint language - precise modeling with UML. Addison-Wesley (1999).
18. Widhalm, R., Mueck, T.A.: Topic Maps - Semantische Suche im Internet. Springer Verlag, Heidelberg (2002).
19. Widhalm, R., Mueck, T.A.: Web metadata semantics - on the road to well-formed topic maps. In: Ozsu T. et al. (eds.): Proc. 2nd Int. Conf. On Web Information Systems Engineering (WISE), vol. 2, p. 141-150. IEEE Computer Society, Los Alamitos, CA (2002).
20. Widhalm, R., Mueck, T.A.: Well-formed Topic Maps. Submitted to SoSyM, Journal on Software & System Modeling. Springer Verlag, Berlin Heidelberg New York (2003)

Semantic Processing of the Semantic Web*

Kunal Patel and Gopal Gupta

Applied Logic, Programming-Languages and Systems (ALPS) Laboratory
Department of Computer Science
University of Texas at Dallas
{kkp025000,gupta}@utdallas.edu

Abstract. We develop a semantics based approach to process information on the semantic web. We show how Horn logic can be used to denotationally capture the semantics of mark-up languages designed for describing resources on the semantic web (such as RDF). The same approach can also be used to specify the semantics of query languages for the semantic web. The semantics of both the resource description languages and the query languages are executable and when put together can be used to compute answers to semantic web queries. The main advantage of this semantic-based approach to processing the semantic web is that these executable semantics can be developed extremely quickly. Thus, as the semantic web mark-up languages evolve rapidly, their implementations can be developed at the same pace. In this paper, we present our approach based on denotational semantics and Horn logic. Our approach is quite general, and applicable to any description format (XML, RDF, DAML, etc.), though in this paper we illustrate it via RDF (Resource Description Framework).

1 Introduction

With increased importance of the Web in our daily lives and national economy, researchers have been looking for ways to make the WEB documents more expressive. That is, designing ways to mark-up WEB documents so that more information can be automatically inferred from them compared to what is possible with HTML. These efforts have resulted in XML (the eXtensible Mark Language) family of notation. While XML allows one to go beyond HTML, it does not take one quite far enough, since the meaning of various tags in a particular XML have to still be agreed upon. The Semantic Web [2] effort goes beyond XML by including metadata information in the document itself.

A problem with any interchange format for the semantic web is that it has to be turned into an "executable" notation, so that new information that is logically implied by the information given in the document can be automatically inferred (computed). Turning a mark-up language into an "executable" entity usually requires writing a compiler like program that maps the language to a

* Authors are partially supported by US NSF grants CCR 99-00320, CCR 98-20852, EIA 01-09347, INT 99-04063, and the US Environmental Protection Agency.

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 80–95, 2003.

notation whose semantics is mathematical; information that is implied can then be inferred automatically from this mathematical semantics.

In this paper, we develop a systematic framework for rapidly translating semantic web formats to notations that are executable. Our framework relies on Horn logic and denotational semantics. Essentially, the denotational semantics of the mark up notation is written in Horn logic. If the semantics is executable, the *denotation* of a document (i.e., the meaning assigned to the document by the semantics) written in that notation is also executable and can be used to infer information implied in that document. Denotational semantics of a language consists of three components: syntax, semantic algebras, and valuation functions. All three components can be specified in Horn logic. The interesting aspect about using Horn logic for specifying denotational semantics is that both the syntax as well as the semantic specification is executable. The syntax specification validates a document, while the semantic specification maps it to its executable mathematical semantics (called its denotation). The denotation of the document can then be employed for inferring implied information (i.e., querying the document). Since both the syntax and semantics are denotationally specified in Horn logic, they are declarative, and thus can be developed very rapidly.

As a result, as the mark-up language or a resource description notation rapidly evolves, its executable semantics can be developed with the same rapid pace. Thus, our framework can be used for computing the executable semantics of XML [19], RDF [25], as well as DAML [1] and OWL [15]. Providing automatic means of translating RDF descriptions into notations that are executable, not only specifies the meaning of the descriptions, but also produces a representation of the descriptions from which inferences can automatically be made using traditional automatic theorem provers, problem solvers, and other inference engines.

In the rest of this paper, we expound on our approach to semantic processing of the semantic web. For illustration purposes we use RDF and show how RDF documents can be rapidly translated into Horn logic (pure Prolog) using the Horn logical denotational method. This Prolog code can be queried to infer (or compute) information that is implied by the document. In a similar vain we show that query languages designed for resource description formats (e.g., RDQL for RDF) can be translated into Horn logic using a Horn logical denotational approach. A RDQL query is thus translated into a Prolog query, and if the semantics of RDF and RDQL are developed in a consistent manner, than the Prolog query obtained from RDQL can be executed on Prolog data obtained from the RDF document.

Horn logic has played an important role in the Semantic Web enterprise and has been used in the past to develop systems for querying RDF documents [34, 16]. The novel contribution of this paper is to show that a semantics based approach that relies on Horn Logic is sufficient to automatically (and rapidly) obtain such systems.

2 A Semantics Based Approach to Translation

If more information is to be inferred from a document written using a mark-up language, then it has to be semantically mapped into an (executable) notation which is amenable to drawing inferences, e.g., via querying. That is, the document has to be syntactically transformed into an executable notation in a manner that preserves its semantics. A relatively easy approach to accomplish this is to express the formal semantics of the mark-up language in terms of this executable notation. Such an approach has been developed in [23] using Horn Logic and denotational semantics. In this approach, to translate one notation, say $\mathcal{L}_s$ to $\mathcal{L}_t$, the denotational semantics of $\mathcal{L}_s$ is given in terms of the language constructs of $\mathcal{L}_t$. All components of this semantics are coded in Horn logic rendering it executable. Given a program/document coded in $\mathcal{L}_s$, its denotation under this semantics is the program/document coded in $\mathcal{L}_t$. The executable semantics thus acts as a translator. Because this semantics is denotational (declarative), it can be specified quickly. Additionally, since the translator is obtained automatically from semantics, it is provably correct w.r.t. the specification.

Thus, assuming that we have RDF as the mark-up language, the semantics of RDF can be given denotationally in terms of an executable notation such as Horn Logic (pure Prolog). Likewise, given a query language such as RDQL, its semantics can also be given denotationally in terms of Horn Logic. This Prolog translated query can be executed w.r.t. the Prolog translated RDF document to compute the query results.

2.1 Denotational Semantics

Denotational Semantics [32] is a well-established methodology for the design, description, and analysis of programming languages. In the denotational semantics approach, the semantics of a programming language/notation is specified in terms of mathematical entities, such as sets and functions. The denotational semantics of a language $\mathcal{L}$ has three components: (i) *syntax specification:* maps sentences of $\mathcal{L}$ to parse trees; it is commonly specified as a grammar in the BNF format; (ii) *semantic algebra:* represents the mathematical objects used for expressing the meaning of a program written in the language $\mathcal{L}$; these mathematical objects typically are sets or domains (partially ordered sets, lattices, etc.) along with associated operations to manipulate the elements of the sets; (iii) *valuation functions:* these are functions mapping parse trees to elements of the semantic algebras.

Given the denotational semantics of a language $\mathcal{L}$, and a program $\mathcal{P}_\mathcal{L}$ written in $\mathcal{L}$, the denotation of $\mathcal{P}_\mathcal{L}$ w.r.t. the denotational semantics can be obtained by applying the top-level valuation function to the parse tree of $\mathcal{P}_\mathcal{L}$. The denotation of $\mathcal{P}_\mathcal{L}$ is an entity that is amenable to formal mathematical processing, and thus has a number of applications. For example, it can be used to prove properties of $\mathcal{P}_\mathcal{L}$, or it can be transformed to obtain other representations of $\mathcal{P}_\mathcal{L}$ (e.g., a compiled representation that can be executed more efficiently [20]). In this

paper we assume that the reader is familiar with formal semantics. A detailed exposition can be found in [32].

2.2 Horn Logical Semantics

Traditional denotational definitions express syntax as BNF grammars, and the semantic algebras and valuation functions using λ-calculus. The Horn Logical Semantics of a language uses Horn-clause logic (or pure Prolog) to code all three components of the denotational semantics of a language [20]. This simple change in the notation for expressing denotational semantics, while resulting in loss of some declarative purity, leads to a number of applications [20]. There are two major advantages [20]. First, a parser can be obtained from the syntax specification with negligible effort: the BNF specification of a language $\mathcal{L}$ can be trivially translated to a *Definite Clause Grammar* (DCG) [35,31]. This syntax specification, coded as a DCG, can be loaded in a Prolog system, and a parser automatically obtained. This parser can be used to parse programs written in $\mathcal{L}$ and obtain their parse trees. The semantic algebra and valuation functions can also be coded in Horn clause logic. Second, Horn logical semantics can be used for automatic verification [20].

Since both the syntax and semantics of the specification are expressed as logic programs, they are both executable. These syntax and semantic specifications can be loaded in a logic programming system and "executed," given a program written in $\mathcal{L}$. This provides us with an interpreter that computes the semantics of programs written in the language $\mathcal{L}$.

2.3 Semantics-Based Language Translation

Horn logical semantics also provides a formal basis for *language translation*. Specification of a translator can be seen as an exercise in semantics. Essentially, the meaning or semantics of the language $\mathcal{L}_s$ can be given in terms of the constructs of the language $\mathcal{L}_t$. This meaning consists of both syntax and semantic specifications. If these syntax and semantic specifications are executable, then the specification itself acts as a translation system, providing a provably correct translator. The task of specifying the filter from $\mathcal{L}_s$ to $\mathcal{L}_t$ consists of specifying the definite clause grammar (DCG) for $\mathcal{L}_s$ and $\mathcal{L}_t$ and the appropriate valuation predicates which essentially relate (map) parse tree patterns of $\mathcal{L}_s$ to parse tree patterns of $\mathcal{L}_t$. Let $\mathcal{P}_s(S_s, T_s)$ be the top level predicate for the DCG of $\mathcal{L}_s$ that takes a sentence S_s of $\mathcal{L}_s$, parses it and produces the parse tree T_s for it. Let $\mathcal{P}_t(S_t, T_t)$ be the top level predicate for the DCG of $\mathcal{L}_t$ that takes a sentence S_t of $\mathcal{L}_t$, parses it and produces the parse tree T_t for it. Let $\mathcal{M}_{st}(T_s, T_t)$ be the top level valuation predicate that relates parse trees of $\mathcal{L}_s$ and $\mathcal{L}_t$. Then the relation

$$\texttt{translate}(S_s, S_t) \texttt{ :- } \mathcal{P}_s(S_s, T_s),\ \mathcal{M}_{st}(T_s, T_t),\ \mathcal{P}_t(S_t, T_t).$$

declaratively specifies the equivalence of the source and target sentence under the semantic mapping given. The `translate` predicate can be used for obtaining S_t given S_s (and *vice versa*).

```
    (propertyelt(propname(qname(nsprefix(s), name(publisher))),
          value(string("World Wide Web Consortium)),
          propname(qname(nsprefix(s), name(publisher)))),
     propertyelt(propname(qname(nsprefix(s), name(title))),
          value(string(W3C Home Page)),
          propname(qname(nsprefix(s), name(publisher)))),
     propertyelt(propname(qname(nsprefix(s), name(date))),
          value(string(1998-10-03T02:27)),
          propname(qname(nsprefix(s), name(date)))
))) ) )
```

3.2 Semantic Specification

RDF statements can be represented as logical expressions. However, instead of defining a triple `(P, S, O)` directly we adopt the approach of [26] and define it as a ternary relation "property": `property(S, P, O)`.
which means resource `S` has property `P` with value `O`. The reason is that developing statements about properties would lead to unwanted higher order statements. Our goal is to keep the semantics first order, not simply because we are interested in using a logic programming based framework, but also because higher order logics quickly become intractable and inefficient. Therefore, we represent RDF document as a set of assertions having three arguments. These assertions are generated during semantic processing, which uses the parse tree produced from the syntax specification as input. The output parse tree obtained is semantically processed to identify the subject, predicate and object, which in turn, are used to generate the ground facts. The semantic function is specified denotationally in Horn Logic. The semantic function (actually, a predicate) maps the parse tree to a database of ground facts.

The following example illustrate fragments of rules used to map property elements to a list of ground facts. The rules essentially map the corresponding parse tree obtained in the semantic phase to the subject, predicate and object values in order to generate the Prolog fact database. The property elements are processed recursively to obtain the list of facts.

```
description(descr(IAA,PE),IAASTR,PROSTR,VALSTR) :-
        idaboutattr(IAA,IAASTR), propertyelts(PE,PROSTR,VALSTR,IAASTR).

propertyelts((PE,PEs),PROSTR,VALSTR,IAASTR) :-
        propertyelt(PE,IAASTR,PROSTR,VALSTR),
        assemble(IAASTR,PROSTR,VALSTR)
        propertyelts(PEs,PROSTR1,VALSTR1,IAASTR).
```

In the first rule, the value of `IAASTR`, which corresponds to the "subject" argument of the property predicate, is obtained from the parse tree. The value of `IAASTR` is then used to retrieve the values of the "predicate" argument and the "object" argument of the property predicate to **`assemble`** the related facts. We can regard the predicate **`property`** as an operation in the semantic algebra.

The following examples illustrate some RDF statements and their corresponding denotation generated as ground facts by our semantic specification.

```
RDF statement 1:
   <rdf:RDF>
       <rdf:Description about="http://www.w3.org/Home/Lassila">
           <s:Creator>
               <rdf:Description about="http://www/w3.org/staffID/85740">
                  <v:Name>Ora Lassila</v:Name>
                  <v:Email>lassila@w3.org</v:Email>
               </rdf:Description>
           </s:Creator>
       </rdf:Description>
   </rdf:RDF>

DENOTATION of statement 1:
   property("http://www.w3.org/Home/Lassila", creator,
                              "http://www.w3.org/staffID/85740").
   property("http://www.w3.org/staffID/85740", name, ora lassila).
   property("http://www.w3.org/staffID/85740", email, lassila@w3.org).
```

The RDF model also defines the concept of containers. Frequently, it is necessary to refer to a collection of resources; for example, to say that a work was created by more than one person, or to list the students in a course. RDF containers are used to hold such lists of resources or literals. A container object can be a *bag*, a *sequence* or an *alternative*. To semantically map the container tag, we make use of structures which holds such list of resources or literals. The name of the structure identifies the type of the container (a bag, a sequence or an alternative). The first argument of the structure is reserved for the ID of the container. If there is no ID for the container then this argument takes the default value of `noid`. The second argument of the structure is a list of resources or literals contained within the container. The following example illustrates the bag container.

Consider an RDF rendition of the information contained in the following sentence: "The students in course 6.001 are Amy, Tim, John and Mary." The corresponding RDF document and its denotation generated by our semantics is shown below.

```
RDF statement 2:
    <rdf:RDF>
       <rdf:Description about="http://mycollege.edu/courses/6.001">
          <s:students>
             <rdf:Bag>
                <rdf:li resource="http://mycollege.edu/students/Amy"/>
                <rdf:li resource="http://mycollege.edu/students/Tim"/>
                <rdf:li resource="http://mycollege.edu/students/Mary"/>
             </rdf:Bag>
          </s:students>
       </rdf:Description>
</rdf:RDF>
```

```
DENOTATION of statement 2:
   property("http://mycollege.edu.courses/6.001", students,
            bag("StudentsOf6.001", ["http://mycollege.edu/students/Amy",
                        "http://mycollege.edu/students/Tim",
                        "http://mycollege.edu/students/Mary")))
```

In the above example the ID of the Bag container is "StudentsOf6.001" and the Bag contains three resources.

In addition to making statement about web resources, RDF can also be used for making statements about other RDF statements. These are referred to as *higher order statements*. In order to express higher order statements RDF defines a model of a statement. This model of a statement has four properties, namely, subject, predicate, object and type, and is also known as a reified statement. The semantics of reified statements is defined with the help of a a new predicate called `statement`. By assigning internal ids, higher order statements can be represented in Horn logic. To achieve this, the first argument of the "statement" is bound to a unique id which links this reified statement to its other properties. The second argument of "statement" is the subject of the reified statement, the third argument is the predicate of the reified statement and the fourth argument is the object of the reified statement. Other additional statements made about this reified statement use the unique id to refer to the reified statement. The following example shows the semantic mapping of a higher order statement.

Consider the statement: "Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila." The RDF document generated for this statement and the corresponding denotation generated by our semantics is shown below. Note that the two predicates in the denotation are joined via the unique id `id007`.

```
RDF statement 3:
 <rdf:RDF>
   <rdf:Description>
     <rdf:subject resource="http://www.w3.org/Home/Lassila"/>
     <rdf:predicate resource="http://description.org/schema/Creator"/>
     <rdf:object>Ora Lassila</rdf:object>
     <rdf:type resource="www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
     <a:attributedTo>Ralph Swick</a:attributedTo>
   </rdf:Description>
 </rdf:RDF>
```

```
DENOTATION of statement 3:
   statement(id007, "http://www.w3.org/Home/Lassila",
                "http://description.org/schema/Creator", Ora Lassila)
   property(id007, attributedTo, Ralph Swick)
```

The semantics of other RDF constructs are similarly defined. Name-spaces are also easily handled by associating name-space identifiers with their corresponding URI via a predicate. Details are omitted due to lack of space. The

whole semantic specification for RDF has indeed been developed. The semantic specification coupled with syntax specification can produce the denotation of any arbitrary RDF document as a set of logic programming ground facts.

3.3 Syntax and Semantics of RDQL

Several RDF Query languages and systems have been recently proposed. These proposals either take the "database approach" or the "knowledge base approach" to query language design (see [9]). The database approach interprets an RDF document primarily as an XML document, which leads to several drawbacks described in [9]. For this reason, all of the recently presented RDF Query languages and systems adhere to the knowledge base approach, explicitly dealing with RDF triples instead of document trees. It is easy to see that the knowledge base approach of RDF querying fits our approach quite well: queries can be expressed as (Horn logic) rules, which can be processed with respect to the denotation of the RDF document using a Logic Programming engine.

The semantics of RDF Query languages like RDQL [9], RQL [3], RDFDBQL [30] or SQUISH [12] can be expressed in Horn logic, using the approach described above. We choose RDQL as an example RDF Query language. RDQL(RDF Data Query Language) is an implementation of SQL-like query language for RDF. It treats RDF as data and provides query with triple patterns and constraints over a single RDF model. An RDQL query is of the form:

```
SELECT vars
FROM documents
WHERE expressions
AND filters
USING Namespace declarations
```

where `SELECT` clause indicates which RDQL variables should be returned by the query; the `FROM` clause of the query indicates the RDF sources to be queried (each source is enclosed by angle brackets(<&>)); while the `WHERE` clause indicates the constraints that RDF triples (subject, predicate, object) that constitute the solution must satisfy. The `WHERE` part is expressed by a list of restrictions separated by commas, each restriction takes the form (subject, predicate, object) where the subject, predicate and object can be a literal value or a RDQL variable. Finally, the `USING` clause declares all the name spaces that will be used for RDF properties.

RDQL queries are also converted to Horn Logic queries, using the same method as described for RDF statements. A syntax specification for RDQL as a DCG is developed based on the RDQL formal grammar described in [9]. The denotational semantics of RDQL as a mapping from parse trees to logic programming facts and queries is also specified. Given a RDQL query, its denotation can be viewed as a query coded in logic programming. The logic programming coded query can be executed on top of the logic programming coded database obtained denotationally from the RDF document. The following example shows a sample RDQL query and its corresponding denotation that is generated:

```
RDQL Query:
    SELECT ?a ?b
    WHERE (?a, pred, ?b)
    AND    ?b > 250
```

```
DENOTATION:
  :- property(X, pred, Y), Y > 250.
```

3.4 An Example

The following example shows some RDF statements and RDQL queries. The Prolog denotation corresponding to these statements and queries is also shown.

```
RDF statement:
   <rdf:RDF>
        <rdf:Description about="http://www.abc.org">
             <s:Author>John</s:Author>
             <s:Title>ABC Home Page</s:Title>
             <s:Hits>200</s:Hits>
        </rdf:Description>
        <rdf:Description about="http://www.pqr.org">
             <s:Author>Tom</s:Author>
             <s:Title>PQR Home Page</s:Title>
             <s:Hits>350</s:Hits>
        </rdf:Description>
   </rdf:RDF>

RDF DENOTATION:
    property("http://www.abc.org", author, john).
    property("http://www.abc.org", title, abc home page).
    property("http://www.abc.org", hits, 200).
    property("http://www.pqr.org", author, Tom).
    property("http://www.pqr.org", title, pqr home page).
    property("http://www.pqr.org", hits, 350).

Query in RDQL (constraints):
    SELECT ?a ?b
    WHERE (?a, hits, ?b)
    AND    ?b > 250

RDQL DENOTATION:
    :- property(X, hits, Y), Y > 250.
```

Once the denotation of RDF and RDQL is obtained in terms of logic programming, the RDQL query can be executed on top of the data provided by RDF using a logic programming engine. For example, execution of the query

```
:- property(X, hits, Y), Y > 250.
```

will yield the answer

```
X = "http://www.pqr.org", Y = 350.
```

```
<?xml version='1.0'?> <rdf:RDF (...)>
<iw:NodeSet rdf:about='../sample/IW1.daml#IW1'>
    <iw:NodeSetContent>
      <iw:KIF>
         <iw:Statement>(wines:COLOR W1 ?x)</iw:Statement>
      </iw:KIF>
    </iw:NodeSetContent>
    <iw:isConsequentOf rdf:parseType='daml:collection'>
         (a NodeSet can be associated to a set of Inference steps)
     <iw:InferenceStep>
        <iw:hasInferenceRule rdf:parseType='daml:collection'>
          <iw:InferenceRule rdf:about='../registry/IR/GMP.daml'/>
      </iw:hasInferenceRule>
      <iw:hasInferenceEngine rdf:parseType='daml:collection'>
         <iw:InferenceEngine rdf:about='../registry/IE/JTP.daml'/>
       </iw:hasInferenceEngine>
       (...)
      <iw:has Antecedent rdf:parseType='daml:collection'>
          (inference step antecedents are IW files with their own URIs)
        <iw:NodeSet rdf:about='../sample/IW3.daml#IW3'/>
        <iw:NodeSet rdf:about='../sample/IW4.daml#IW3'/>
      </iw:hasAntecedent>
      <iw:hasVariableMapping
              rdf:type='http://www.daml.org/2001/03/daml+oil#List'/>
      (...)
  </iw:InferenceStep>
    </iw:isConsequentOf>
  </iw:NodeSet>
</rdf:RDF>
```

Fig. 2. An Inference Web Proof.

5.2 Registry

The IW registry is a hierarchical interconnection of distributed repositories of information relevant to proofs and explanations. Entries in the registry contain the information linked to in the proofs. Every entry in the registry is a file written in DAML+OIL. Also, every entry is an instance of a registry concept. *InferenceEngine*, *Language* and *Source* are the core concepts in the registry. Other concepts in the registry are related to one of these core concepts.

In order to interact with the IW registry, the IW provides a web agent registrar that supports users in updating or browsing the registry. The registrar may grant update or access privileges on a concept basis and it may define and implement policies for accessing the registry. The current demonstration registrar is available at: http://onto.stanford.edu:8080/iwregistrar/.

The *InferenceEngine* is a core concept since every inference step should have a link to at least one entry of *InferenceEngine* that was responsible for instantiating the inference step itself. For instance, Figure 2 shows that the `iw:hasInferenceEngine` property of `iw:InferenceStep` has a pointer to `JTP.-`

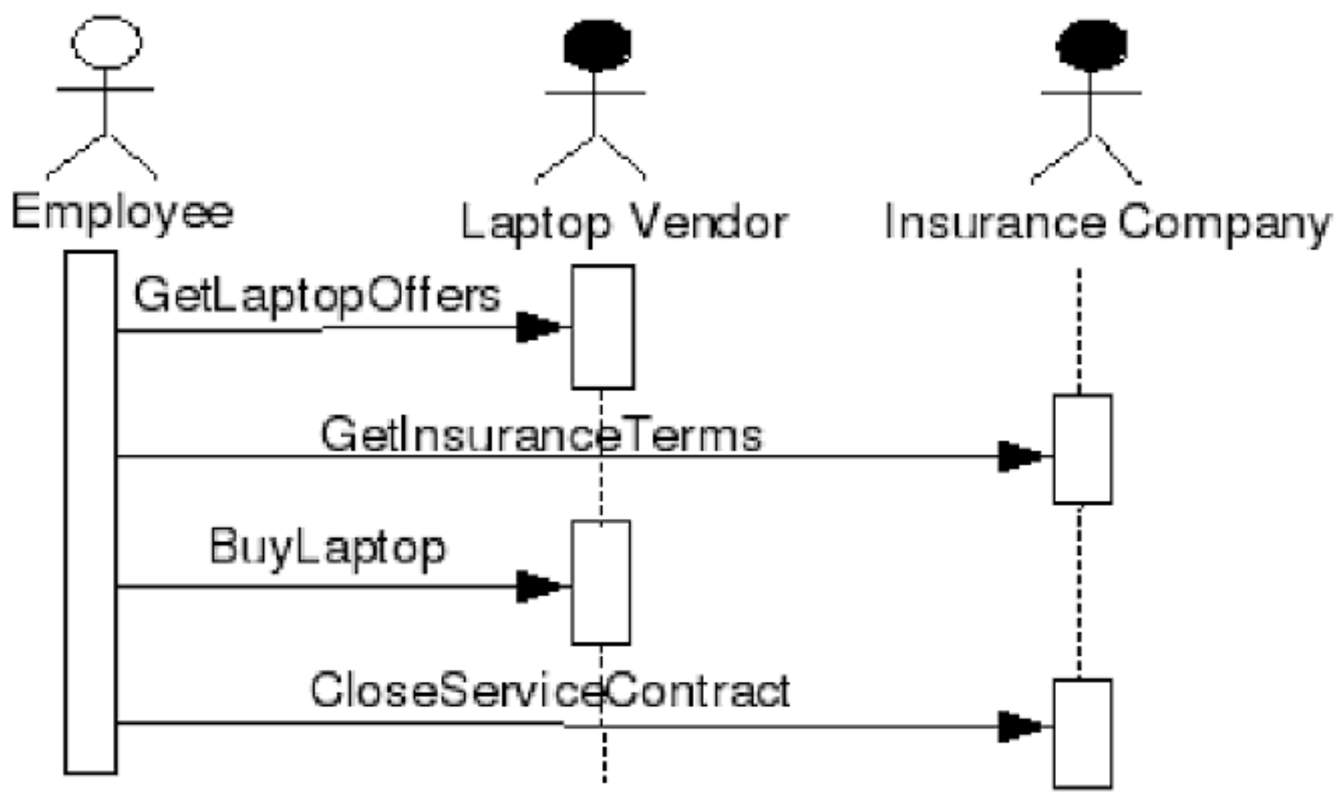


Fig. 1. Sequence Diagram for the Use Case

```
<?xml version="1.0" encoding="UTF-8"?> <definitions
name="LaptopService"
    targetNamespace="http://laptop.wsdl/laptop/"
    <types>
       <rdf:RDF>
          <rdfs:Class rdf:ID="Laptop">
             <rdfs:label>Laptop</rdfs:label>
          </rdfs:Class>
          <rdf:Property rdf:ID="diskSpace">
             <rdfs:label>diskSpace</rdfs:label>
             <rdfs:range rdf:resource="&rdfs;Literal"/>
             <rdfs:domain rdf:resource="#Laptop"/>
          </rdf:Property>
           ...
          <rdf:Property rdf:ID="price">
             <rdfs:label>price</rdfs:label>
             <rdfs:range rdf:resource="&rdfs;Literal"/>
             <rdfs:domain rdf:resource="#Laptop"/>
          </rdf:Property>
           ...
      </rdf:RDF>
</types>
<message name="getOffersRequest">
    <part name="processorSpeed" type="rdf:ID=processorSpeed"/>
    <part name="diskSpace" type="rdf:ID=diskSpace"/>
</message>
<message name="getOffersResponse">
    <part name="laptopOffers" type="rdf:ID=laptops"/>
</message>
    ...
    <portType name="LaptopService">
        <operation name="getLaptopOffers" parameterOrder="processorSpeed diskSpace">
            <input message="tns:getOffersRequest" name="getOffersRequest"/>
            <output message="tns:getOffersResponse" name="getOffersResponse"/>
        </operation>
        ...
    </portType>
</definitions>
```

Fig. 2. Web Service Description of Laptop Vendor

same. (The number of leading zeros required is computed from the number of blank nodes in the RDF graph being considered).

3.2 Semantic

The techniques in this paper rely on being able to make meaningless changes to an RDF graph. This is done in accordance with the RDF formal semantics [10], by using a special property, which we conventionally refer to as **c14n:true** defined here:

```
<rdf:RDF xml:base="&c14n;" xmlns:c14n="&c14n;#">
  <rdfs:Property rdf:ID="true">
     <rdfs:description>This property is true whatever
resource is its subject, and whatever literal is its object.
    Thus triples with literal objects, and c14n:true as
predicate, can arbitrarily be added to and deleted from an RDF
graph without changing its meaning. </rdfs:description>
  </rdfs:Property>
<rdf:RDF>
```

(Omitting namespace declarations and the definition of the entity **&c14n;** as the URL `http://www-uk.hpl.hp.com/people/jjc/rdf/c14n`).

By specifying this predicate as being always true, adding or deleting triples with this predicate does not alter the entailments under the RDF model theory. Formally the semantics of the document are unchanged.

If desired, an alternative would be to use OWL (Full) to define a trivially true predicate which holds between all resources and all long integers. The following cardinality constraint, in the OWL abstract syntax 11, expresses this condition:

```
class(rdfs:Resource complete restriction(c14n:true cardinality=2^64))
dataValuedProperty( c14n:true range( xsd:long ) )
```

3.3 N-Triples to Canonical XML

In this paper, we concentrate on creating canonical representations of RDF graphs in N-Triples [4].

However, for full integration with tools built on Canonical XML [12] it is necessary to transform these files into XML, and, moreover, we wish the XML produced to canonically depend on the original RDF graph.

This needs to be done by specifying a canonical method for turning an N-Triple document into RDF/XML. This must: preserve the order of the triples; prohibit the many variants in the grammar; specify the whitespace precisely.

3.4 RDF C14N without Blank Nodes

To create a canonical N-Triples file for an RDF graph without any blank nodes we can simply reorder the lines in the N-Triples document to be in lexicographic order.

```
<qowl:Query rdf:ID="">
  <qowl:sender rdf:resource="http://cs.cmu.edu/~nsadeh"/>
</qowl:Query>
<mc:Person rdf:about="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://sadehlab.cs.cmu.edu/Variable#location" />
</mc:Person>
```

Fig. 4. Query issued by the user 'nsadeh' requesting the location of user 'fgandon'

```
<sowl:ReadAccessRule>
  <rdf:label>people can only know whether or not I am on campus</rdf:label>
  <sowl:target>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="&variable;#location"/>
    </mc:Person>
  </sowl:target>
  <sowl:check>
   <rowl:And>
     <rowl:condition>
       <mc:E-Wallet rdf:about="&variable;#e-Wallet">
         <mc:owner>
          <mc:Person rdf:about="&variable;#owner"/>
         </mc:owner>
       </mc:E-Wallet>
     </rowl:condition>
     <rowl:condition>
       <mc:Place rdf:about="http://www.cmu.edu">
          <mc:include rdf:resource="&variable;#location" />
       </mc:Place>
     </rowl:condition>
     <rowl:not-condition>
       <qowl:Query rdf:about="&variable;#query">
         <qowl:sender rdf:resource="&variable;#owner" />
       </qowl:Query>
     </rowl:not-condition>
   </rowl:And>
  </sowl:check>
  <sowl:revision>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="http://www.cmu.edu"/>
    </mc:Person>
  </sowl:revision>
</sowl:ReadAccessRule>
```

Fig. 5. Privacy rule obfuscating the location of the owner

```
<wowl:ServiceRule wowl:salience="50">
  <rdf:label>provide location for IP Address</rdf:label>
  <wowl:output>
    <mc:Entity rdf:about="&variable;#entity">
      <mc:location rdf:resource="&variable;#location" />
    </mc:Entity>
  </wowl:output>
  <wowl:precondition>
    <mc:Entity rdf:about="&variable;#entity"><mc:ip>&variable;#ip</mc:ip>
    </mc:Entity>
  </wowl:precondition>
  <wowl:call>
   <wowl:Service wowl:name="call-web-service">
     <wowl:parameter>http://cmu.edu/location_tracking#</wowl:parameter>
     <wowl:parameter>&variable;#ip</wowl:parameter>
   </wowl:Service>
  </wowl:call>
</wowl:ServiceRule>
```

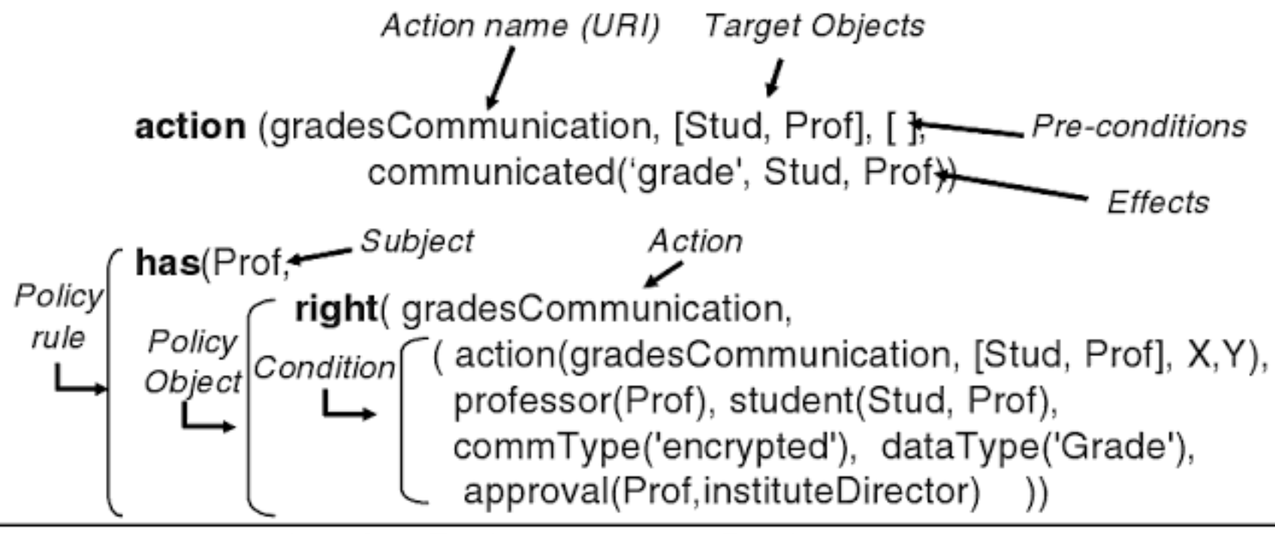
Fig. 6. Service rule for location-tracking invocation

Table 2. Policy Specification in KAoS, Rei, and Ponder

KAoS

```
<?xml version='1.0'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
         xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
         xmlns:policy="http://ontology.coginst.uwf.edu/Policy.daml#"
         xmlns="http://ontology.coginst.uwf.edu/ExamplePolicies/PolicyExample.daml#">
<daml:Ontology rdf:about="">
    .....
<daml:Class rdf:ID="ExaminationGradePolicyAction">
    <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="http://ontology.coginst.uwf.edu/Action.daml#EncryptedCommunicationAction"/>
     <daml:Restriction>
        <daml:onProperty rdf:resource="http://ontology.coginst.uwf.edu/Action.daml#performedBy"/>
        <daml:toClass rdf:resource="http://ontology.coginst.uwf.edu/ActorClasses.daml#AgentProfessors"/>
     </daml:Restriction>
     <daml:Restriction>
        <daml:onProperty rdf:resource="http://ontology.coginst.uwf.edu/Action.daml#hasDestination"/>
        <daml:toClass rdf:resource="http://ontology.coginst.uwf.edu/ActorClasses.daml#AgentStudents"/>
     </daml:Restriction>
     <daml:Restriction>
        <daml:onProperty rdf:resource="http://ontology.coginst.uwf.edu/ Action.daml#hasApproval"/>
       <daml:toClass
            rdf:resource="http://ontology.coginst.uwf.edu/ActorClasses.daml#AgentInstituteDirector"/>
     </daml:Restriction>
    </daml:intersectionOf>
</daml:Class>
<policy:PosAuthorizationPolicy rdf:ID="ExaminationGradePolicy">
    <policy:controls rdf:resource="#ExaminationGradePolicyAction"/>
    <policy:hasSiteOfEnforcement rdf:resource="http://ontology.coginst.uwf.edu/Policy.daml#SubjectSite"/>
     <policy:hasPriority>10</policy:hasPriority>
     <policy:hasUpdateTimeStamp>446744445544</policy:hasUpdateTimeStamp>
</policy:PosAuthorizationPolicy >
```

Rei



Ponder

```
domain prof = /SysEntities/Agents/ProfessorAgents;
domain stud = /SysEntities/Agents/StudentsAgents;

inst auth+ ExamGradeComm {
   subject  s= prof;
   target  t = /SysEntities/SysServices/CommunicationChannel;
   action t.communication ("Encrypted", data, destination) ;
   when data.getType = "Grade"
     && destination == (stud -> select (st | st.professor == s))
     && s.receivedApproval(s.getInstituteDirector()) == 'true' ;
}
```

4.2 Policy Reasoning

In KAoS, various inferences can be performed for a variety of purposes, such as detecting and harmonizing conflicting policies (e.g., is any negative authorization

21. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. In *IEEE Computer*, Vol. 25, No. 3. March, 1992.

Appendix: Complete Example

```
// 0. rdf schema semantics
FORALL Mdl @rdfschema(Mdl) {
  FORALL O,P,V   O[P->V] <- O[P->V]@Mdl.
  FORALL O,P,V   O[P->V] <- EXISTS S
      (S[rdfs:subPropertyOf->P] AND O[S->V]).
  FORALL O,P,V   O[rdfs:subClassOf->V] <-
      EXISTS W   (O[rdfs:subClassOf->W] AND W[rdfs:subClassOf->V]).
  FORALL O,P,V   O[rdfs:subPropertyOf->V] <-
      EXISTS W   (O[rdfs:subPropertyOf->W] AND W[rdfs:subPropertyOf->V]).
  FORALL O,T   O[rdf:type->T] <-
      EXISTS S   (S[rdfs:subClassOf->T] AND O[rdf:type->S]).
}

// 1. learning object ontology plus some resources at WUW

@wuw:ont {

   // ontology
   wuw:LearningResource[rdfs:subClassOf -> rdfs:Resource].
   wuw:Book[rdfs:subClassOf -> wuw:LearningResource].
   wuw:Exercise[rdfs:subClassOf -> wuw:LearningResource].
   wuw:OpenQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:FillInQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:YesNoQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> wuw:Exercise].

   // some instances
   question1_1[rdf:type -> wuw:OpenQuestion; wuw:difficulty -> low].
   question1_2[rdf:type -> wuw:FillInQuestion; wuw:difficulty -> medium].
   question1_3[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> high].
   question2_1[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> low].
   question2_2[rdf:type -> wuw:MultipleChoiceQuestion; wuw:difficulty -> medium].
   question3_1[rdf:type -> wuw:MultipleChoiceQuestion; wuw:difficulty -> high].
   book1[rdf:type -> wuw:Book].
}

// 2. exercise types ontology

@exty:ont {
    exty:LearningProgressTest[rdfs:subClassOf -> rdfs:Resource].
    exty:Auto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:NonAuto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:MarkReader[rdfs:subClassOf -> exty:Auto].
    exty:NonMarkReader[rdfs:subClassOf -> exty:Auto].
}

// 3. mapping definitions

@exty:mappings {

   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:YesNoQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:FillInQuestion[rdfs:subClassOf -> exty:Auto].
   wuw:Exercise[rdfs:subClassOf -> exty:LearningProgressTest].
   wuw:OpenQuestion[rdfs:subClassOf -> exty:NonAuto].

   FORALL R  R[rdf:type -> exty:TestQuestion] <-
       ((R[rdf:type -> wuw:MultipleChoiceQuestion] OR
         R[rdf:type -> wuw:YesNoQuestion] OR
         R[rdf:type -> wuw:FillInQuestion])  AND
```

Table 1. Narrative units and their associated rules (simplified for readability)

PersonalData

#	Current Role	Relation	Progression	New Role	Relevant Properties
1	MainCharacter	none	none	none	[descriptionText, portrait, . . .]
. . .					

PrivateLife

#	Current Role	Relation	Progression	New Role	Relevant Properties
2	MainCharacter	none	none	none	[name, birthDate, . . .]
3	MainCharacter	isMarried	privateLife	spouse	[marriedPlace, marriedDate]
4	MainCharacter	parentOf	none	child	[birthPlace, birthDate]
5	spouse	none	none	none	[name, daughterOf]
. . .					

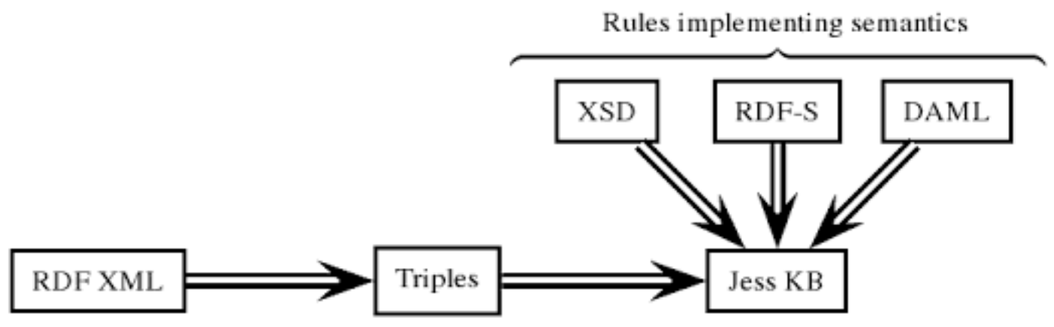
Career

#	Current Role	Relation	Progression	New Role	Relevant Properties
6	MainCharacter	none	none	none	[workPlace, . . .]
7	MainCharacter	taughtBy	career	master	[stylePeriod]
8	master	style	career	technique	[styleDescription]
. . .					

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns    ="http://www.cwi.nl/~media/ns/discourse#"
         xmlns:d  ="http://www.cwi.nl/~media/ns/discourse#">

    <discourse d:type='biography'>
        <narrativeunit rdf:parseType='Resource'>
            <type>PersonalData</type>
            <actant>Rembrandt_van_Rijn</actant>
            <role>MainCharacter</role>
    <data rdf:parseType='Resource'>
        <descriptiveText>Rembrandt Harmensz. van Rijn is the most famous...</descriptiveText>
        <portrait rdf:resource='http://www.rijksmuseum.nl/SK-A-4691.Z.jpg'/>
    </data>
</narrativeunit>
<narrativeunit rdf:parseType='Resource'>
    <type>Career</type>
    <actant>Rembrandt_van_Rijn</actant>
    <role>Painter</role>
    <data>....</data>
    <narrativeunit rdf:parseType='Resource'>
        <type>Career</type>
        <actant>Chiaroscuro</actant>
        <role>Technique</role>
        <data rdf:parseType='Resource'>
            <descriptiveText>Clair-obscur (French) and chiaroscuro (Italian)...</descriptiveText>
            <painting rdf:resource='http://www.rijksmuseum.nl/SK-A-1935.ORG.jpg'/>
            <painting rdf:resource='http://www.rijksmuseum.nl/SK-A-4691.Z.jpg'/>
        </data>
    </narrativeunit>
</narrativeunit>
<narrativeunit rdf:parseType='Resource'>
    <type>PrivateLife</type>
    <actant>Rembrandt_van_Rijn</actant>
    <role>PrivatePerson</role>
    <data>....</data>
    <narrativeunit rdf:parseType='Resource'>
                <type>PrivateLife</type>
                <actant>Saskia_Uylenburgh</actant>
                <role>Spouse</role>
                <data rdf:parseType='Resource'>
                    <descriptiveText>In 1634 Rembrandt married...</descriptiveText>
                    <portrait rdf:resource='http://www.rijksmuseum.nl/SK-A-4057.jpg'/>
                </data>
            </narrativeunit>
        </narrativeunit>
    </discourse>
</rdf:RDF>
```

Fig. 3. RDF/XML representation of the biography



(a) Overview of DAMLJessKB process

```
<rdfs:Class rdf:about="#Artifact" />

<rdfs:Class rdf:about="#Part">
  <rdfs:subClassOf rdf:resource="#Artifact"/>
</rdfs:Class>

<eng:Part rdf:about="#Cog17" />
```

(b) RDF snippet in XML form

```
⟨ ⟨rdf:type eng:Artifact rdfs:Class⟩
⟨rdf:type eng:Part rdfs:Class⟩
⟨rdfs:subClassOf eng:Part eng:Artifact⟩
⟨rdf:type ex:Cog17 eng:Part⟩ ⟩
```

(c) Corresponding triple model

$$\forall a, b, i \ \langle \texttt{rdfs:subClassOf}\ a\ b\rangle \wedge \langle \texttt{rdf:type}\ i\ a\rangle \supset \langle \texttt{rdf:type}\ i\ b\rangle$$

(d) Rule implied by RDF-S semantics

```
⟨ ⟨rdf:type eng:Artifact rdfs:Class⟩
⟨rdf:type eng:Part rdfs:Class⟩
⟨rdfs:subClassOf eng:Part eng:Artifact⟩
⟨rdf:type ex:Cog17 eng:Part⟩
⟨rdf:type ex:Cog17 eng:Artifact⟩ ⟩
```

(e) Model derived from facts and rule

Fig. 2. The basic approach of DAMLJessKB.

on its ability to make interesting inferences, and consequently on its ability to accept valid inputs. DAMLJessKB avoids these problems by providing for such inferences. We see as one of its strengths the ability to add value to even simple Semantic Web applications by providing an easy to use DAML inference tool coupled with the excellent application building features of the underlying production system.

4 Approach

The DAMLJessKB mapping is illustrated in Figure 2. One of the basic properties of RDF is that no matter how complex its XML form, the underlying model can be seen simply as a list of triples. Each triple asserts a relation between a subject and an object through a predicate. DAMLJessKB maps these into facts in a production system.

Procedure. Given an XML source document (Figure 2(b)), an RDF parser generates a stream of triples (Figure 2(c)). These triples are asserted into the production system and rules derived from the semantics of the language are applied (Figure 2(d)) to populate the knowledge base with the additional facts which can be entailed from the input (Figure 2(e)). Additional methods are then invoked to carry out extra-logical aspects of the languages, such as loading ontologies from `daml:imports` statements, as well as tasks such as ontology debugging through looking for valid but probably unintended inputs.

The advantage of this approach is that information extracted and inferred from input documents is immediately available for use in the underlying production sytem. Such

```
(defrule mininclusive-classification
 (PropertyValue
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  ?dt http://www.daml.org/2001/03/daml+oil#Datatype)
 (PropertyValue
  http://www.w3.org/2000/10/XMLSchema#minInclusive
  ?dt ?anon)
 (PropertyValue
  http://www.w3.org/1999/02/22-rdf-syntax-ns#value
  ?anon ?value)

 (PropertyValue
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  ?inst
  http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal)
 (PropertyValue
  http://www.w3.org/1999/02/22-rdf-syntax-ns#value
  ?inst ?ival)
 (test (and (integerp ?ival) (integerp ?value)
   (>= ?ival ?value)))
=> (assert
 (PropertyValue
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  ?inst ?dt)))
```

(a) Sample rule for classifying literals

```
(defrule mininclusive-subclassing
  (PropertyValue
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   ?dt1 http://www.daml.org/2001/03/daml+oil#Datatype)
  (PropertyValue
   http://www.w3.org/2000/10/XMLSchema#minInclusive
   ?dt1 ?anon1)
  (PropertyValue
   http://www.w3.org/1999/02/22-rdf-syntax-ns#value
   ?anon1 ?value1)

  (PropertyValue
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   ?dt2&~?dt1
   http://www.daml.org/2001/03/daml+oil#Datatype)
  (PropertyValue
   http://www.w3.org/2000/10/XMLSchema#minInclusive
   ?dt2 ?anon2)
  (PropertyValue
   http://www.w3.org/1999/02/22-rdf-syntax-ns#value
   ?anon2 ?value2)
  (test (and (integerp ?value1) (integerp ?value2)
    (>= ?value1 ?value2)))
=>
  (assert (PropertyValue
    http://www.w3.org/2000/01/rdf-schema#subClassOf
    ?dt1 ?dt2)))
```

(b) Sample rule for terminological reasoning on datatypes

Fig. 5. Examples of DAMLJessKB's treatment of literals and datatypes

```
(defrule subclass-instances
  (PropertyValue
    http://www.w3.org/2000/01/rdf-schema#subClassOf
    ?child ?parent)
  (PropertyValue
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    ?instance ?child)
=>
 (assert
  (PropertyValue
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    ?instance ?parent)))
```

(a) Rule implementing basic notion of subclassing.

```
(defrule rdfs-domain
  (declare (salience -100))
  (PropertyValue
    http://www.w3.org/2000/01/rdf-schema#domain
    ?p ?c)
  (PropertyValue ?p ?i ?o)
  (not (PropertyValue
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    ?i ?c))
=>
  (assert (PropertyValue
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    ?i ?c))
  (gentle-warning "Set object '" ?i "' type to '" ?c
    "' due to a domain restriction on '" ?p "'"))
```

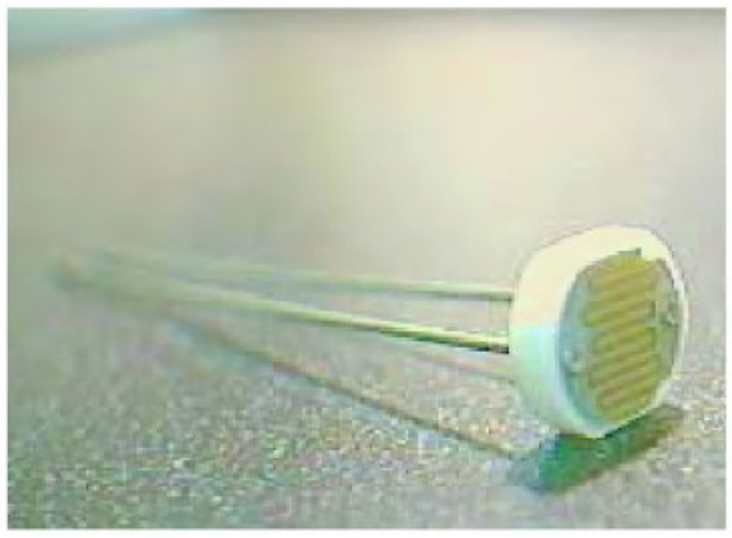
(b) One rule corresponding to semantics of `rdfs:domain`.

Fig. 6. Examples of DAMLJessKB's instance reasoning

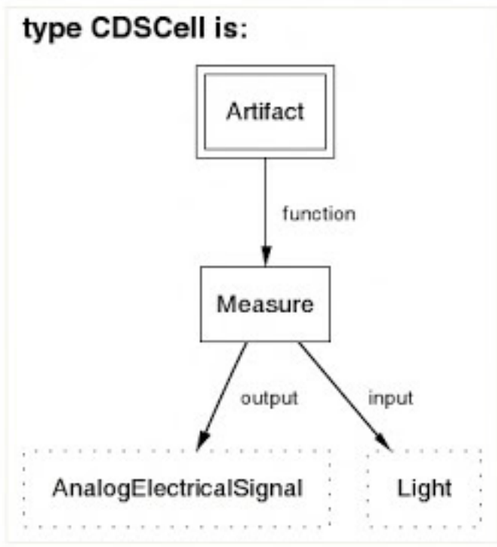
ations to create new classes. In addition, special subsumption relationships are defined based on the particular semantics of the datatypes. This is very similar to the use of `daml:Restriction` classes, discussed later. Figure 5(b) demonstrates a rule implementing subsumption between `minInclusive` constraints. Given the two definitions in Figure 4(a), this rule will assert that `AtLeast6` is a subclass of `AtLeast4`.

5.3 Instance Data Reasoning

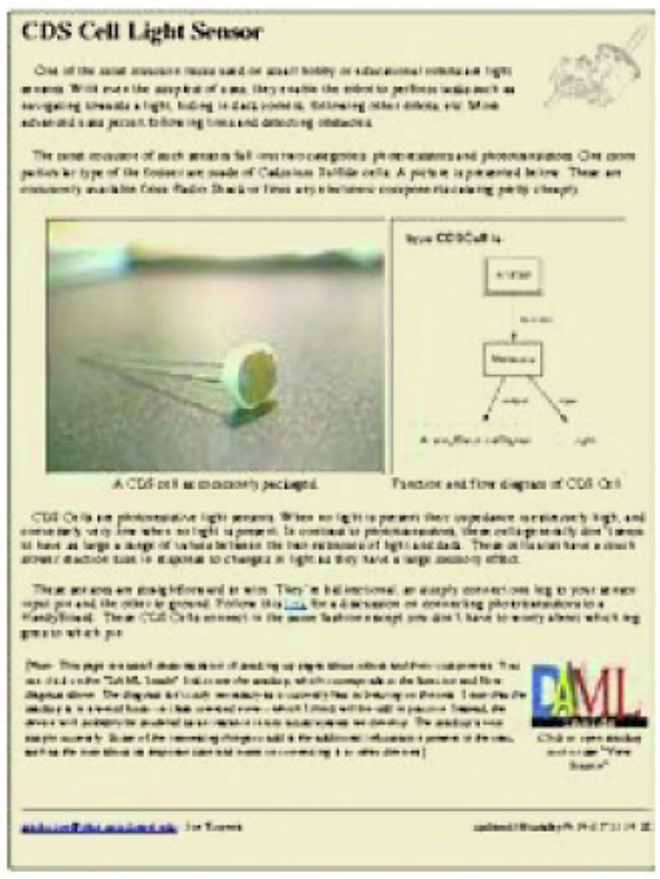
The set of rules implementing DAMLJessKB's reasoning can be roughly seen as falling into two categories. One concerns reasoning on instances of classes. The second concerns terminological reasoning, determining relationships between the classes themselves. This is roughly analogous to ABox and TBox reasoning in description logic systems.



(a) A Cadmium Sulfide (CDS) cell



(b) CDS cell function and flow diagram



(c) Webpage for CDS cell

```
<daml:Class rdf:about="#Sensor">
 <daml:intersectionOf
   rdf:parseType="daml:collection">
  <daml:Class rdf:about="&eng;#Artifact" />

  <daml:Restriction>
   <daml:onProperty rdf:resource="&eng;#function"/>
   <daml:minCardinalityQ rdf:value="1" />

   <daml:hasClassQ>
    <daml:Class>
     <daml:intersectionOf
       rdf:parseType="daml:collection">
      <daml:Class rdf:about="&func;#Measure" />

      <daml:Restriction>
       <daml:onProperty
         rdf:resource="&eng;#output" />
       <daml:minCardinalityQ rdf:value="1" />
       <daml:hasClassQ
         rdf:resource="&flow;#MeasureSignal" />
      </daml:Restriction>

      <daml:Restriction>
       <daml:onProperty rdf:resource="&eng;#input"/>
       <daml:minCardinalityQ rdf:value="1" />
       <daml:hasClassQ rdf:resource="&flow;#Flow"/>
      </daml:Restriction>

     </daml:intersectionOf>
    </daml:Class>
   </daml:hasClassQ>
  </daml:Restriction>
 </daml:intersectionOf>
</daml:Class>
```

(d) DAML description of sensor class

Fig. 9. Artifact function and flow modeling and a comparison of two assembly models

Real-World Application: Variational Engineering Design. As outlined in Section 2, we are interested in reasoning about electromechanical assemblies and components. This includes tasks such as determining if an artifact performs a given function and searching for artifacts in design repositories to find similarities.

Consider the problem of designing LEGO robots, such as with the very popular Lego Mindstorms robot kits[3]. Figure 9(a) shows a typical light sensor component used in Lego robot kits; Figure 9(b) shows an engineering function and flow diagram [16,17] for this component. Function and flow diagrams provide an abstract representation of an assembly, its components and their intended behavior. This diagram notes that the sensor measures an input light source and outputs an electrical signal as the measure. However,

[3] http://mindstorms.lego.com/

```
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="Country">
   <rdfs:domain rdf:resource="#Park"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
```

```
Garden Ontology:
<daml:Class ID="Garden">
   <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
         <daml:onProperty rdf:resource="#Name"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
<daml:DatatypeProperty rdf:ID="Name">
   <rdfs:domain rdf:resource="#Garden"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="Place">
   <rdfs:domain rdf:resource="#Garden"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
```

The second example is about service advertisement of web service. One is about ServiceProfile Ontology of Service Domain in http://www.daml.com/Service/Profile-Ont. The other is about ServiceAdvertisement Ontology of Agent Domain in http://www.ichi.sjtu.edu.cn/Agent/Advertisement-Ont.

```
ServiceProfile Ontology:
<daml:Class ID="ServiceProfile">
   <daml:subClassOf rdf:resource="#Profile"/>
</daml:Class>
<daml:DatatypeProperty rdf:ID="ServiceName">
   <rdfs:domain rdf:resource="#ServiceProfile"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="ServiceType">
   <rdfs:domain rdf:resource="#ServiceProfile"/>
   <rdfs:range rdf:resource="#Classification"/>
</daml:DatatypeProperty>
<daml:Class rdf:ID="Classification">
   <daml:oneOf rdf:parseType="daml:collection">
      <Type rdf:ID="Database-Query"/>
      <Type rdf:ID="E-Commence"/>
      <Type rdf:ID="Language-Translation"/>
   </daml:oneOf>
```

```
</daml:Class>
<daml:DatatypeProperty rdf:ID="Input">
   <rdfs:domain rdf:resource="#ServiceProfile"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="Output">
   <rdfs:domain rdf:resource="#ServiceProfile"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
```

ServiceAdvertisement Ontology:

```
<daml:Class ID="ServiceAdvertisement">
   <daml:subClassOf rdf:resource="#Profile"/>
</daml:Class>
<daml:DatatypeProperty rdf:ID="ServiceName">
   <rdfs:domain rdf:resource="#ServiceAdvertisement"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="ServiceType">
   <rdfs:domain rdf:resource="#ServiceAdvertisement"/>
   <rdfs:range rdf:resource="#Category"/>
</daml:DatatypeProperty>
<daml:Class rdf:ID="Category">
   <daml:subClassOf rdf:resource="#Classification"/>
</daml:Class>
<daml:DatatypeProperty rdf:ID="Precondition">
   <rdfs:domain rdf:resource="#ServiceAdvertisement"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="Postcondition">
   <rdfs:domain rdf:resource="#ServiceAdvertisement"/>
   <rdfs:range rdf:resource="#string"/>
</daml:DatatypeProperty>
```

3.2 Ontology-Mapping Service

Furthermore, we discuss how to implement concept transformation through two concrete query examples. Supposed a user in Administration Domain wants to search for information about Yu Yuan Garden, and he sends Query A via a user agent to a query agent that will assist user in searching for information. A specific notation is used to identify concepts and specify ontology in users' query, for example, Query A is formalized by Garden Ontology with properties Name and Place. SEARCH <Administration:Garden> means that this user expects to search for objects of class Garden including all properties that are defined for this class, and searching results must be restricted to those objects with properties Name "Yu Yuan Garden" and Place "Shanghai, China".

```
<daml:ObjectProperty rdf:ID="mother">
  <rdfs:subPropertyOf rdf:resource="#parent"/>
  <rdfs:range rdf:resource="#Female"/>
</daml:ObjectProperty>
```

With these basic frame primitives defined, we are ready to look at an example using the Criminal Process frames.

An Example: The Criminal Process Frame

The basic frame is the CRIMINAL PROCESS Frame. It is a type of background frame. CP is used as a shorthand for this frame.

```
<daml:Class rdf:ID="CriminalProcess">
  <rdfs:subClassOf rdf:resource="#Frame"/>
</daml:Class>

<daml:Class rdf:ID="CP">
  <daml:sameClassAs rdf:resource="#CriminalProcess"/>
</daml:Class>
```

The CRIMINALPROCESS frame has a set of associated roles. These roles include that of COURT, DEFENDANT, PROSECUTION, DEFENSE, JURY, and CHARGES. Each of these roles may have a filler with a specific semantic type restriction. FrameNet does not specify the world knowledge and ontology required to reason about Frame Element filler types. We believe that one of the possible advantages in encoding FrameNet data in DAML+OIL is that as and when ontologies become available on the web (uch as OpenCYC), we can link to them for this purpose.

In the example fragment below we use the CYC *Court-Judicial* collection to specify the type of the COURT and the CYC *Lawyer* definition to specify the type restriction on the frame element DEFENSE. For illustrative purposes, the DAML+OIL markup below shows the use of a different ontology (from CYC) to restrict the defendant to be of type PERSON as defined in the example ontology. This restriction uses the DAML+OIL example from http://www.daml.org/2001/03/daml+oil-ex)

```
<daml:ObjectProperty rdf:ID="court">
  <rdfs:subPropertyOf rdf:resource="#FE"/>
  <rdfs:domain rdf:resource="#CriminalProcess"/>
  <rdfs:range rdf:resource="&CYC;#Court-Judicial"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="defense">
  <rdfs:subPropertyOf rdf:resource="#FE"/>
  <rdfs:domain rdf:resource="#CriminalProcess"/>
  <rdfs:range rdf:resource="&CYC;#Lawyer"/>
</daml:ObjectProperty>
```

```
<rdf:RDF
 ... (Definition of namespaces)>
  <ppqs:CompoundScale rdf:ID="utcAndKelvin">
   <ppqs:unitSymbol>UTC, K</ppqs:unitSymbol>
   <rdfs:label xml:lang="en">UTC and Kelvin</rdfs:label>
   <rdfs:domain rdf:resource="#InstantInTimeAndTemperature"/>
   <rdfs:range>
    <snt:SymbolVectorSpace>
     <snt:cartesianProductOf>
      <snt:SymbolVector>
       <snt:hasTerms rdf:parseType="Collection">
        <snt:SymbolVector rdf:about="&time;#UtcRepresentation"/>
        <snt:MathsSpace rdf:about="&snt;Real"/>
       </snt:hasTerms>
      </snt:SymbolVector>
     </snt:cartesianProductOf>
    </snt:SymbolVectorSpace>
   </rdfs:range>
   <snt:compoundPropertyCompoundDomainComponents">
    <snt:PropertyVector>
     <snt:hasTerms rdf:parseType="Collection">
      <ppqs:Scale rdf:about="&time;utc"/>
      <ppqs:Scale rdf:about="&siUnits;kelvin"/>
     </snt:hasTerms>
    </snt:PropertyVector>
   </snt:compoundPropertyCompoundDomainComponents>
  </ppqs:CompoundScale>
</rdf:RDF>
```

Fig. 18. Definition of a compound scale

```
<rdf:RDF
  ... (Definition of namespaces)>
  <snt:CompoundPropertySingleDomain rdf:ID="timeOfInstantAndMeasuredTemp">
   <rdfs:label xml:lang="en">time of instant and measured temperature</rdfs:label>
   <rdfs:domain rdf:resource="&po;#ProductAtInstant"/>
   <rdfs:range rdf:resource="InstantInTimeAndTemperature"/>
    <snt:compoundPropertySingleDomainComponents>
     <snt:PropertyVector>
      <snt:hasTerms rdf:parseType="Collection"/>
       <rdf:Property rdf:about="&time;timeOfProductAtInstant"/>
       <rdf:Property rdf:about=&myProperties;measuredTemperature"/>
      </snt:hasTerms>
     </snt:PropertyVector>
    </snt:compoundPropertySingleDomainComponents>
  </snt:CompondPropertySingleDomain>
</rdf:RDF>
```

Fig. 19. Definition of a compound physical property

7 Data Quality

Except for the measured data value and the time of measurement the quality of the measured data is of relevance and must be considered in an ontology for the engineering domain. The ScadaOnWeb approach for defining quality is based on the