Compiling Complex Terminologies for Query Processing

Heiner Stuckenschmidt
Vrije Universiteit Amsterdam
heiner@cs.vu.nl

Abstract

It is widely accepted that the Semantic Web will be based on machine-readable metadata describing the content of resources. These descriptions are designed to enable intelligent agents to locate and filter relevant information with a higher level of accuracy. The Resource Description Framework (RDF) has been developed as universal language for encoding content-related metadata and recently a number of query languages have been proposed to extract information from metadata models. Current applications of RDF and RDF query languages only use very simple metadata like simple concept hierarchies (The Open Directory) or pre-defined attribute value pairs (Dublin Core). In this paper we address the problem of encoding and querying complex metada using RDF models and queries. In our approach we consider ontologies with complex concept definitions in the spirit of DAML+OIL and propose a pre-processing method that enables us to access these models using RDF query languages without losing information.

1 Motivation

In todays semantic web research there is still a big gap between theoretical considerations about the use of rich background knowledge in terms of ontologies and real implementations that are actually used in applications on the web. While theoreticians claim that rich knowledge models (as supported by the ontology language DAML+OIL or recently OWL) provide the expressive power and reasoning support needed in many domains, the application of such rich models is hampered by the high complexity of reasoning that makes them unlikely to scale up to a realistic setting. As a consequence, most existing implementations of semantic web infrastructure such as JENA [1], RDFSuite [2] or Sesame [3] rely on more light-weight solutions restricting them selves to RDF, RDF schema or a small subset of the OWL language.

Our concern is now to close the gap between these views and to provide ways of exploiting the expressive power of rich ontology languages in combination with efficient implementations that rely on light weight solutions. In this paper, we concentrate on the problem of querying information on the Semantic Web. Most existing implementations support querying RDF models, some also supporting schema aware querying. However, it has been argued that the presence of rich background knowledge requires deductive reasoning for achieving complete answers [Horrocks and Tessaris, 2000]. To our knowledge, none of the existing implementations of semantic web infrastructure supports this kind of query processing [4]. Therefore, if we want to make use of the expressive power of languages like DAML+OIL and the querying facilities of existing infrastructure, we have to find ways of pre-processing DAML+OIL models in such a way that they can be handled by methods developed for light-weight approaches.

The obvious way of dealing with DAML+OIL models in a lightweight setting, ignoring all language features not supported by the light weight approach has serious drawbacks as it always means loosing information. Therefore, rather than weakening the background model, our approach is based on the idea of compiling out implicit knowledge in the background model and enriching the plain RDF model with this additional information. As finding the hidden information again requires expensive logical reasoning, this compilation step is done off-line. The actual query answering is done on the enriched model already. This approach, known as knowledge compilation, is well known in Artificial Intelligence research [Cadoli and Donini, 1997]. The novelty of this work is to apply the idea in the specific context of the semantic web and its representations.

The contribution of this paper is two-fold:

- We show how existing reasoning techniques can be used to compile implicit information about named objects of a domain model into a plain RDF description.
- We propose a heuristic approach for enhancing this

[1] `http://www.hpl.hp.com/semweb/jena.htm`
[2] `http://139.91.183.30:9090/RDF/`
[3] `http://sesame.aidministrator.nl/`

[4] There is a prototypical implementation of the query answering approach of Horrocks and Tessaris, but it does not provide a real infrastructure for handling semantic web data

description with information about anonymous objects than can be derived from the background knowledge.

The paper is organized as follows. We first review some basic notions of RDF, mainly referring to its model theoretic semantics. We pay special attention to the interpretation of unlabeled nodes as existentially quantified variables referring to anonymous objects and on their role in query answering. Afterwards we review existing work on explicating implicit information contained in background knowledge. We consider the case of RDF schema and of DAML+OIL model. Finally, we discuss the issue of implicit relational information that is only partially covered by existing approaches. We show how the query answering approach of Horrocks and Tessaris [Horrocks and Tessaris, 2000] can be used to compile relational information about named objects and we sketch an approach for dealing with unnamed objects in the compilation process. We summarize with a discussion of open questions.

2 RDF Querying

The ability to query RDF models in an efficient way is an essential aspects with respect to building semantic web applications. As RDF is designed as a language for describing the content of an information source, being able to find a specific piece of information is equivalent to being able to find the corresponding RDF statement. In the following, we present a general view on answering queries based on the RDF data model introduced above and briefly review the RDF data model and some existing query languages.

2.1 RDF Query langauges

The RDF data model described above and its associated semantics provides us with a basis for defining queries on RDF models in a straightforward way. The idea that has been proposed elsewhere and is adopted here is to use graphs with unlabeled nodes as queries. The unlabeled elements in a query graph can be seen as variables of the query. In this approach, answers to a query can be defined in two ways:

- A sub-graph of the given RDF model that is an instantiation of the query graph.
- The set of resources that if they are used to instantiate unlabeled nodes in the query graph result in a sub-graph of the given RDF model

From a theoretical point of view these two definitions are exchangeable as one can easily be derived from the other by either extracting instantiated resources from the an answer graph or by instantiating the query graph with the resources from the answer set, respectively. Due to this correspondence, we will use the definitions synonymously.

Another characteristic feature of RDF is the possibility to refer to entire RDF graphs and not only to single resources. At first sight, the ability to refer to entire statements destroys the graph Another feature of graph-based queries is that answering queries about RDF models can be based on the same techniques that are used for reasoning about such model. In especially, graph matching is a basic technique with respect to reasoning and query answering (see e.g. [Caroll, 2002]). Adopting this technique, query answering is done by finding a match between the query graph and the RDF model, allowing unlabeled nodes in the query graph to be matched against any node in the model. Once a match is found, the query graph can be instantiated with the matching nodes on the model or those nodes can be returned as an answer.

With the increasing interest in RDF, a number of RDF query languages have been developed and integrated in the available infrastructure. One of the most widely adopted ones that is based on graphs as queries is SquishQL [Miller *et al.*, 2002]. Different Implementations of SquishQL based languages exist as part of RDF APIs or persistent storages and can therefore be assumed to be widely used. In the following, we consider RDQL, an implementation of SquishQL that is part of the Jena RDF Toolkit and the RDF storage and query system Sesame. Following the basic definitions of SquishQL a query has the following components:

SELECT the select clause identifies variables form the query graph that should be returned as the result of the query.

FROM the from clause specifies the RDF model to be used as a basis for querying. The model is identified by its URI.

WHERE the where clause describes the query graph in terms of a conjunction of RDF triples connected by variables.

AND the and clause contains a Boolean expression that constraints possible variable instantiations in the query graph.

USING the using clause can be used to define abbreviations used in the specification of the query.

These constructs allow the user to formulate queries in an SQL like style which is well known by most application developers.

2.2 An Example

We use a toy example from the domain of family relationships to illustrate the way queries are formulated. For the sake of simplicity, we omit the FROM, and USING clauses.

```
SELECT ?p ?c
WHERE
     (?p has-child ?c)
     (?p type ?g)
     (?p age ?a)
AND
     (?g eq #female)
```

The above query asks for mothers and their children. The result of a query will be a table with the identifiers of persons that satisfy the requirements specified in the query, namely that they are in the has-child relation where the first member of the relation is of type female. We further use the following RDF model of mothers to be queried:

```
<rdf:Description rdf:about="alice">
  <type rdf:resource="#female">
```

```
  <has-child rdf:resource="#betty>
</rdf:Description>

<rdf:Description rdf:about="jane">
  <mother-of rdf:resource="#tom">
</rdf:Description>

<rdf:Description rdf:about="eve">
  <type rdf:resource="#female">
  <has-son rdf:resource="#charles">
  <has-mother rdf:resource="#betty">
</rdf:Description>

<rdf:Description rdf:about="betty">
</rdf:Description>

<rdf:Description rdf:about="carol">
  <type rdf:resource="mother">
  <has-child rdf:resource="#doris">
</rdf:Description>

<rdf:Description rdf:about="doris">
  <type rdf:resource="#mother">
</rdf:Description>

<rdf:Description rdf:about="mary">
  <type rdf:resource="#virgin-mary">
</rdf:Description>
```

Applying the query specified above we would get the pair (alice, betty) as a result because alice is defined to be female at having a child. This result is not very satisfying as it is easy to see that all of the defined persons are actually mothers and should therefore be in the answer to the query. The reason for the incomplete result is the lack of information about the meaning of the descriptions.

3 Ontology-Aware Querying

One of the main features of the semantic web is the idea to enrich metadata descriptions with explicit models of their intended meaning. These models range from simple schema definitions to complex ontologies that define concepts and describe them by necessary and sufficient conditions thus enabling intelligent applications to reason about their members and their relation to each other. On the semantic web, we want to exploit this semantic information for query answering as it provides us with background information for computing more complete results. In the example above we also want to retrieve instances as an answer to ?p that have the following properties:

- are of type female and have a daughter or a son, because this implies having a child
- have a child and are human and not male because not being male implies being female.
- are of type mother, because this implies being female and having a child.
- are of type virgin-mary as this implies being the mother of christ.

Expressive language have been developed that can be used to encode the kind of background knowledge needed to draw the conclusions mentioned above. In especially, RDF schema provide means for define simple schematic information [Brickley and Guha, 2003]. DAML+OIL extends RDF schema towards a more expressive language for defining the meaning of classes [van Harmelen *et al.*, 2001a]. Unfortunately, simple query language like RDQL are not able to make use of the background knowledge per se. We rather have to do some pre-processing on the RDF model to be queried in order to get all intended results. In the following, we describe how this pre-processing step can be done in order to query metadata that is based on an RDF schema and on a DAML+OIL ontology, respectively, and refer to existing approaches that use these methods to support querying. Afterwards, we point out to remaining problems that are the main motivation for the compilation approach, we propose as an extension of existing proposals.

3.1 Schemas

RDF schema provide a language for encoding structural background information about the vocabulary used in an RDF Model. This structural information provides insights into the relations between the different elements of the model and helps to draw conclusions that could not be found from the plain model. In particular, a schema defines hierarchies of classes and relations as well as restrictions on the range and the domain of relations. In the family domain, we use in our example, the schema may contain the following information about the classes and relations used in the model:

```
<rdf:property id="has-son">
  <rdfs:subPropertyOf rdf:resource="#has-child"/>
</rdf:Property>

<rdf:property id="has-mother">
  <rdfs:range rdf:resource="#female"/>
</rdf:property>

<rdf:property id="mother-of">
  <rdfs:subPropertyOf rdf:resource="#has-child"/>
  <rdfs:domain rdf:resource="#female"/>
</rdf:property>
```

This schema defines that having a son is a special case of having a child, that mother is a subclass of female persons which is the range of the relation has-mother and the domain of the relation mother-of, which in turn is a special case of having a child. We immediately see that this information is relevant for answering our query as from it we can conclude that Jane and Eve fulfill the requirements specified in the query. Formally, this is established by the axiomatic semantics of RDF schema given in [Hayes, 2003]. Applying these axioms to our example RDF model, we can add a number of new facts that can be matched by the query engine. In particular, we get the following more complete definitions of jane, eve and betty:

```
<rdf:Description rdf:about="jane">
```

```
  <mother-of rdf:resource="#tom">
  <has-child rdf:resource="#tom">
  <type rdf:resource="#female">
</rdf:Description>

<rdf:Description rdf:about="eve">
  <type rdf:resource="#female">
  <has-son rdf:resource="#charles">
  <has-child rdf:resource="#charles">
  <has-mother rdf:resource="#betty">
</rdf:Description>

<rdf:Description rdf:about="betty">
  <type rdf:resource="#female">
</rdf:Description>
```

Now that the information about these resources has been made explicit, posting the example query against the expanded model will also return jane and eve as an answer to the query. Betty, however, is now only known to be female, but there is no explicit statement about her child which disables the query engine to recognize her as matching the query. In fact, this approach of first expanding a model using schematic information and then evaluating queries against the completed model is a common approach that has for example been implemented in Sesame [Broekstra *et al.*, 2002].

3.2 Ontologies

The example domain already shows that there are many aspects of terminological knowledge that can not be captured by RDF schema. These aspects include the following facts that might be relevant for answering queries about our domain:

- the same person may not be male and female
- a female person automatically becomes a mother when having a child
- virgin Mary is the mother of christ

Including these facts in a model of information semantics requires a far more expressive language. DAML+OIL [van Harmelen *et al.*, 2001a] is such a language that has been defined on top of RDF schema, extending it with additional operators for defining classes by constraining possible members. For our domain a DAML+OIL model might contain the following definitions:

```
<daml:Class rdf:ID="human">
  <daml:equivalentTo>
    <daml:unionOf>
      <daml:Class rdf:resource="#male"/>
      <daml:Class rdf:resource="#female"/>
    </daml:unionOf>
  </daml:equivalentTo>
</daml:Class>

<daml:Class rdf:ID="male">
  <rdfs:subClassOf>
    <daml:complementOf>
      <daml:Class rdf:resource="#female"/>
    </daml:complementOf>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="mother"> <rdfs:subClassOf>
  <daml:complementOf>
    <daml:Class rdf:resource="#male"/>
  </daml:complementOf>
  <daml:restriction>
    <daml:onProperty rdf:resource="#has-child"/>
    <daml:hasClass rdf:resource="#human"/>
  </daml:restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="virgin-mary">
  <rdfs:subClassOf>
    <daml:Class rdf:resource="female">
    <daml:restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#has-child"/>
      <daml:hasValue rdf:resource="#jesus-christ"/>
    </daml:restriction>
  </rdfs:subClassOf>
</daml:Class>
```

The model adds further semantic information about the domain to our model, namely the fact that the class of all humans is exactly the union of male and female person. That male persons cannot be female at the same time, that a mother is a female person having a child and that virgin Mary is a female person having exactly one child which is jesus christ.

In [van Harmelen *et al.*, 2001b] a formal semantics for DAML+OIL is described. The semantics is based on an interpretation mapping into an abstract domain. More specifically, every concept name is mapped on a set of objects, every property name is mapped on a set of pairs of objects. Individuals (in or case resources) are mapped on individual objects in the abstract domain. Formally, an interpretation is defined as follows:

Definition 1 (Interpretation) *An Interpretation consists of a pair $(\Delta, .^{\mathcal{E}})$ where Δ is a (possibly infinite) set and $.^{\mathcal{E}}$ is a mapping such that:*

- $x^{\mathcal{E}} \in \Delta$ *for every individual* x
- $C^{\mathcal{E}} \subseteq \Delta$ *for all classes* C
- $R^{\mathcal{E}} \subseteq \Delta \times \Delta$ *for all roles* R

We call $.^{\mathcal{E}}$ the extension of an individual, concept or role, respectively.

This notion of an interpretation is a very general one and does not restrict the set of objects in the extension of a concept. This is done by the use of operators for defining classes. These kinds of operators restrict the possible extensions of a concept. These kinds of restriction are the basis for deciding whether a class definition is equivalent, more specialized or more general than another. Formally, we can decide whether one of the following relations between two expressions hold:

subsumption: $C_1 \sqsubseteq C_2 \iff C_1^{\mathcal{E}} \subseteq C_2^{\mathcal{E}}$

membership: $x : C \iff x^{\mathcal{E}} \in C^{\mathcal{E}}$

Based on the definitions of subsumption and membership, we can use terminological reasoners to compute these relations from a given model and the corresponding ontology. The main results for the given model are that mother is a subclass of female as all human beings that are not male are known to be female (the class human is a partition of male and female persons). This of course implies that all members of the class mother are also members of the class female. The resulting relations that correspond to the rdfs:subClassOf and the rdf:type statements can now be added to the RDF model and its schema. The result are extended descriptions of Carol, Doris and Mary as given below:

```
<rdf:Description rdf:about="carol">
  <type rdf:resource="#mother"/>
  <type rdf:resource="#female"/>
  <has-child rdf:resource="#doris"/>
</rdf:Description>

<rdf:Description rdf:about="doris">
  <type rdf:resource="#mother">
  <type rdf:resource="#female"/>
</rdf:Description>

<rdf:Description rdf:about="mary">
  <type rdf:resource="#female"/>
  <type rdf:resource="#virgin-mary">
</rdf:Description>
```

We see that the additional information obtained by explicitly adding implicit subclass and type relations to the model extends set of answers to our query with Carol, because it is now explicitly stated that she is female and has a child named Doris. This approach goes further than the use of schema information, however, it does not solve all problems, because we still cannot find out that Doris and Mary are also answers to our query.

4 Relational Knowledge

The reason while present approaches fail to compile the ontology in such a way that all intended answers can be given from the RDF model is caused by their limited abilities to compile relational knowledge. In the example it followed from the ontology that Mary is related to jesus-christ by the has-child relation. In the case of Doris, it is implied that she is connect to some, maybe unknown object via the same relation. Description logic reasoners allow to reason with these kinds of relational information by constructing models in terms of possible objects, their type and their relations. However, this information is only used internally to establish subsumption and membership relations. In order to overcome this problem we can use techniques for deductive query answering in the off-line compilation phase and store the retrieved answers as explicit knowledge in our RDF model.

4.1 Compiling Object Relationships

Horrocks and Tessaris propose a deductive approach for answering conjunctive queries over description logic knowledge bases. The idea of the approach of Horrocks and Tessaris now to translate the query into an equivalent concept expression, classify this new concept and use standard inference methods to check whether an object is an instance of the query expression. This approach makes use of the fact that binary relations in a conjunctive query can be translated into an existential restriction in such a way that logical consequence is preserved after a minor modification of the A-Box. Details are given in the following theorem.

Theorem 1 (Role Roll-Up (Horrocks and Tessaris 2000)) *Let $\langle \mathcal{C}], \mathcal{R}, \mathcal{A}\rangle$ be a description logic knowledge base with concept definitions $\mathcal{C}$, relation definitions $\mathcal{R}$ and assertions A. Let further R be a role, C_I Concept names in T and a, b be individual names in A. Given a new concept name P_b not appearing in T, then*

$$\langle \mathcal{C}, \mathcal{R}, \mathcal{A}\rangle \models (a, b) : R \wedge b : C_1 \wedge \cdots \wedge b : C_k$$

if and only if

$$\langle \mathcal{C}, \mathcal{R}, \mathcal{A} \cup \{b : P_b\}\rangle \models a : \exists R(P_b \sqcap C_1 \sqcap \cdots \sqcap C_k)$$

As DAML+OIL can be seen as a specific variant of description logics, the query answering approach can be applied to DAML+OIL ontologies (compare [Horocks and Tessaris, 2002]). In especially, we can directly ask for objects that are related by the has-child relation using the following very simple query:

$$Q(X, Y) \leftarrow (X, Y) : \texttt{has} - \texttt{child}$$

Using the theorem of Horrocks and Tessaris, we can now do the translation of the conjunct (X, Y) : $\texttt{has} - \texttt{child}$ into a concept expression called role-up. In order to actually retrieve related objects, we do this translation for instantiations of the general conjunct where the Y variable is replaced by an object contained in the model. Substituting X for example by the object $\texttt{jesus} - \texttt{christ}$ we get the conjunct $(X, jesus - christ) : \texttt{has} - \texttt{child}$ that translates into the concept expression $\exists has - child.P_{jesus-christ}$ This concept specifies all objects that are related to the object $\texttt{jesus} - \texttt{christ}$ by the $\texttt{has} - \texttt{child}$ relation. We can use existing description logic reasoners in order to retrieve all objects belonging to this concept. For the example instantiation, the reasoner will return mary telling us that we can add the information that mary and jesus-christ are in the has child relation into the RDF model. The new definition of mary will be the following:

```
<rdf:Description rdf:about="mary">
  <type rdf:resource="#female"/>
  <type rdf:resource="#virgin-mary">
  <has-child rdf:resource="#jesus-christ"/>
</rdf:Description>
```

In order to compile all the object relations implied by an ontology, we have to iterate this process over all instances and over all relations mentioned. The corresponding algorithm is depicted as Algorithm 1.

Given a set of relations and objects, the algorithm compiles out all relational information about them that is implied by

Algorithm 1 Compile Relations

Require: A Terminological Knowledge Base T
Require: A set of relations R
Require: A set of objects O
  A := $\emptyset$
  for all $r \in R$ **do**
    for all $o \in O$ **do**
      C := $(\exists r.\{o\})$
      R := $\{o' \in O | T, A \models o : C\}$
      for all $x \in R$ **do**
        A := $A \cup \{(x\ r\ o)\}$
      end for
    end for
  end for
  return A

Figure 1: Basic algorithm for compiling object relations

a terminological knowledge base. When compiling an RDF model that is based on a DAML+OIL model, the terminological knowledge base will be provided by the DAML+OIL model. The sets of relations and objects will consist of all known relations and objects that are contained either in the RDF model or the DAML+OIL model.

4.2 Anonymous Objects

While the use of the query answering approach of Horrocks and Tessaris enables us to compile out information about relations that exists between objects in a model, there is still implicit relational information that is not captured by this approach. The reason for this is that the logical nature of DAML+OIL allows us to capture incomplete information about the domain of interest in the sense that it is not required to always name related objects in a concept definition. There are also ways of talking about the existence and the number of related objects in the domain without actually naming these objects. The approach of Horrocks and Tessaris ignore this information, because their approach aims at answering questions about named objects in a model. We think, however, that in the in the context of the semantic web there are also may situations where we are also interested in this incomplete information. This argument is based on the open world assumption. The fact that the model does not contain the name of the child of a female person in our example does not mean that there is no information about this child. It just happens not to be contained in this specific model. Therefore, the answer that doris by virtue of being a mother has a child we do not know the name of is also a valuable answer to our example question. Following this argument, we sketch an approach for also compiling relational information that contains anonymous objects. In the resulting RDF model these anonymous objects will be represented by blank nodes.

In order to determine what kind of anonymous objects we have to deal with, we have to have a look at the possibilities DAML+OIL provides us for describing relations to objects that are not mentioned themselves. There are two operators that are directly connected with anonymous objects (compare figure 2). First of all, there is the daml:hasClass operator claiming that all objects of a class are related some object of a certain type. Further, there are cardinality statements claiming that all objects of a class are related to at least, at most or exactly a certain number of objects of a certain type. The case where no special type of the related objects is required is a special case and can therefore also be treated by the approach. The same holds for the first mentioned daml:hasClass operators, because it is equivalent to stating that the minimal number of object of certain type that is required equals one. Further, requiring an exact number of related objects (daml:CardinalityQ) can be expressed by requiring a minimal and a maximal number of related objects of a certain type where both numbers equal the required number of objects. Therefore, the main relational construct, we have to focus on when investigating the problem of compiling relational knowledge are the following constructs:

```
<daml:restriction daml:minCardinalityQ="n">
  <daml:onProperty rdf:resource="#p"/>
  <daml:hasClassQ rdf:resource ="#C"/>
</daml:restriction>

<daml:restriction daml:maxCardinalityQ="m">
  <daml:onProperty rdf:resource="#p"/>
  <daml:hasClassQ rdf:resource ="#C"/>
</daml:restriction>
```

In the next section we introduce and extension of the compilation algorithm that deals with anonymous objects making use of these constructs.

4.3 Compilation with Anonymous Objects

As motivated above, our approach for compiling relational information with anonymous objects relies on the use of qualified number restrictions. We assume that all existential restrictions and unqualified number restrictions have been translated into qualified number restrictions in the way sketched in the last section. Further, we assume that the background knowledge is consistent in itself and that the information model is consistent with this background model. In especially, this guarantees that there is no conflict between the upper and lower bounds of related objects with respect to the qualified number restrictions we use for compilation. This requirement can be checked using existing reasoning systems. For every object o and relation r in the model we now perform a four step compilation process:

Step 1: Collect Bounds In the first step, we determine the possible range of number of objects related to o by the relation r. We collect all class descriptions o can be proven to be a member of. From these description, we extract all qualified number restrictions that refer to r. Finally, we take the largest lower bound and the smallest upper bound with respect to each class name2 occurring in the restrictions and store these numbers together with the class name.

Step 2: Verify Bounds In the second step, we check whether the bounds determined for the individual classes

in the hierarchy are consistent with the bound determined for their its subclasses. For every class c in the hierarchy starting at the bottom of the hierarchy, we sum up the lower bounds determined for all direct subclasses. We then check, whether this sum is smaller than the upper bound determined for that class.

Step 3: Adjust Bounds In this step, we adjust the lower bound on related objects in the light of collected information about more specific information about related objects that imply some of the information contained in the current bounds.

a) As a first step, we set the lower bound to the current upper bound if the upper bound is lower is lower than the sum of the lower bounds determined for all direct subclasses.

b) As a second step we subtract the number of all known and already compiled anonymous objects of this type related to o. This ensures that only the most specific available information is actually compiled out and prevents us from adding redundant information.

Step 4: Compile Relations In the last step, we add information about relations of the object o to anonymous objects to the RDF model by adding the corresponding triples. For each class name, we look up the finally determined lower bound of objects related to o via r. This bound is an natural number l that specifies the number of necessarily existing relations excluding already known relational information and necessary relations to objects of a more specific type (compare step 2). We generate l anonymous objects ai and add the triples (o r ai) and (ai type c) for each of these objects.

It is important to recall, that different from the other compilation methods described in this paper, this compilation process is of a heuristic nature and does not claim to produce logically sound results. The reason is that the current version of the compilation approach does not take into account all information contained in the background knowledge.

4.4 A Simple Example

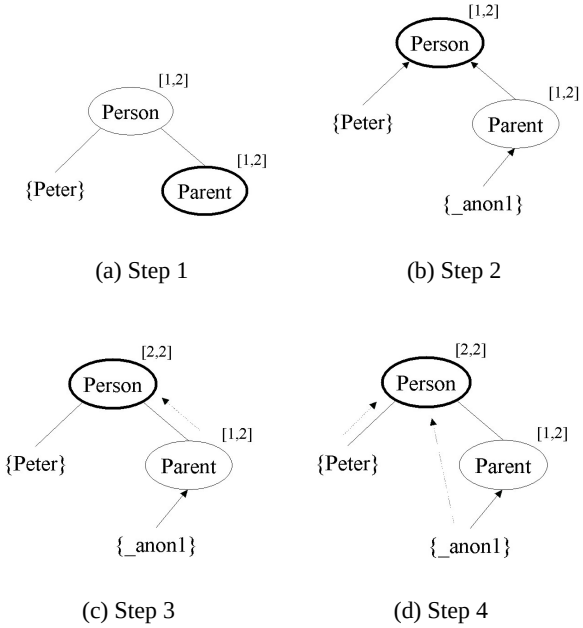
We illustrate our compilation approach using a simple example. Consider the following model describing the instance betty:

```
<daml:Class rdf:ID="grandma">
  <rdfs:subClassOf>
    <daml:Class rdf:resource="mother">
    <daml:restriction>
      <daml:onProperty rdf:resource="#has-child"/>
      <daml:hasClass rdf:resource="#parent"/>
    </daml:restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="relaxed-parent">
  <rdfs:subClassOf>
    <daml:restriction daml:maxCarninality="2">
      <daml:onProperty rdf:resource="#has-child"/>
      <daml:hasClassQ rdf:resource="#person"/>
    </daml:restriction>
  </rdfs:subClassOf>
</daml:Class>

<rdf:Description rdf:about="betty">
  <type rdf:resource="#grandma"/>
  <type rdf:resource="#relaxed-parent"/>
  <has-child rdf:resource="#peter"/>
</rdf:Description>

<rdf:Description rdf:about="peter">
  <type rdf:resource="#person"/>
</rdf:Description>
```



(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

Figure 2: Compuring Bounds in a Concept Hierarchy

When we look at the definition of betty, we see that there are different sources of relational information, we are interested in. From her being of type mother (implied from the fact of being a grandmother) we derive a lower bound of one for the has-child relation with respect to the class person. Further, being a grandma also implies a lower bound of one on the has-child relation with respect to the class parent, which is a subclass of person. An upper bound of two on the relation has-child is provided by the membership to the class relaxed-parent. Finally, there is the the explicitly mentioned child peter who is of type person. The result of collecting these bounds is shown in figure 3a.

After processing the class parent which is lower in the hierarchy, we see that the lower bound on the number of related objects of type parent is higher that the number of related individuals of that type (i.e. one compared to zero). As a reaction, we add an anonymous object of type parent to the

RDF model (compare figure 2b). In the next iteration of the process, we process the concept person by adding the lower bound of all sub-concepts and checking it against the upper bound (figure 2c). From the resulting lower bound of two we subtract the number of all related objects of this type. In our case there are two objects of this kind: peter and the anonymous object created in the previous iteration. As the result is zero we conclude that the necessary number of related objects is already present in the model. The triples added in the first iteration of the process lead to a more complete definition of the instance elly with respect to the has-child relation as shown below:

```
<rdf:Description rdf:about="betty">
  <type rdf:resource="#happy-grandma"/>
  <type rdf:resource="#relaxed-parent"/>
  <has-child rdf:resource="#peter"/>
  <has-child rdf:resource="_anonX"/>
</rdf:Description>

<rdf:Description rdf:about="_anonX">
  <type rdf:resource="#parent"/>
</rdf:Description>
```

The added information can now be used in querying enabling us to derive that elly is necessarily connected to an instance of type parent by the has child relation using a plain RDF query language. In fact, this object of type parent may be identical with peter who we already know, but the background model does not provide us with enough information to prove whether this is true or not.

5 Discussion

In this paper, we addressed the problems that arise from the existing gap between theoretical investigations and practical implementations of semantic web technology. For the specific problem of querying RDF models with background knowledge we presented an approach to enhance plain RDF descriptions by information implied by the background knowledge. To this end, we discussed some already existing approaches and presented new methods for compiling complex relational knowledge using the query answering approach proposed by Horrocks and Tessaris. All the methods mentioned produce a provably correct and complete result with respect to answering queries about named objects in an RDF model.

We argued that an open world assumption as we face to on the semantic web also requires to consider anonymous objects in answering queries. We described a first step towards an adequate treatment of these objects in the compilation step. The approach presented already produces some reasonable results, but it cannot give any soundness or completeness guarantees. We do not expect to come up with a provably complete algorithm for this problem, however, there are still many options for improving the result presented most of them concerned with features for defining complex relational structures (e.g. role hierarchies, transitivity). Further, we need more experiences with applying these methods to realistic scenarios in order to develop heuristics for improving the compilation result.

References

[Brickley and Guha, 2003] Dan Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF schema. Working draft, W3C, 2003. http://www.w3.org/TR/rdf-schema/.

[Broekstra *et al.*, 2002] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *The Semantic Web - ISWC 2002* [2002], pages 54–68.

[Cadoli and Donini, 1997] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

[Caroll, 2002] J. Caroll. Matching RDF graphs. In *The Semantic Web - ISWC 2002* [2002], pages 5–15.

[Hayes, 2003] Patrick Hayes. RDF semantics. Working draft, W3C, 2003. http://www.w3.org/TR/rdf-mt/.

[Horocks and Tessaris, 2002] I. Horocks and S. Tessaris. Querying the semantic web: A formal approach. In *The Semantic Web - ISWC 2002* [2002], pages 177–191.

[Horrocks and Hendler, 2002] I. Horrocks and J. Hendler. *The Semantic Web - ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*. Springer, 2002.

[Horrocks and Tessaris, 2000] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI*, pages 399–404, 2000.

[Miller *et al.*, 2002] L. Miller, A. Seaborne, and A. Reggiori. Three implementations of SqishQL, a simple RDF query langauge. In *The Semantic Web - ISWC 2002* [2002], pages 423–435.

[van Harmelen *et al.*, 2001a] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the daml+oil (march 2001) ontology markup language. http://www.daml.org/2001/03/reference.html, march 2001.

[van Harmelen *et al.*, 2001b] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A model-theoretic semantics for daml+oil (march 2001). http://www.daml.org/2001/03/model-theoretic-semantics.html, march 2001.