RDFSculpt: Managing RDF Schemas under Set-like Semantics

Zoi Kaoudi[1,2], Theodore Dalamagas[2], and Timos Sellis[2]

[1] Dept. of Electronic and Computer Engineering
Technical University of Crete, Greece
zoi@intelligence.tuc.gr

[2] School of Electr. and Comp. Engineering,
National Technical University of Athens, Greece
{zkaoudi,dalamag,timos}@dblab.ece.ntua.gr

Abstract. The Semantic Web is an extension of the current Web in which information is given well-defined meaning to support effective data discovery and integration. The RDF framework is a key issue for the Semantic Web. It can be used in resource discovery to provide better search engine capabilities, in cataloging for describing the content of thematic hierarchies in thematic catalogs and digital libraries, in knowledge sharing and exchange of Web agents, etc. Up to now, RDF schemas have been treated rather as sets of individual elements (i.e. model primitives like classes, properties, etc.). Under that view, queries like "find the part of a portal catalog which is not present in another catalog" can be answered only in a procedural way, specifying which nodes to select and how to get them. For this reason, we argue that answering such queries requires treating schemas as a whole rather than as sets of individual elements. We introduce a set of operators with set-like semantics to manage RDF schemas. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. We also present RDFSculpt, a prototype system that implements our framework.

1 Introduction

The Semantic Web [1] is an extension of the current Web in which information is given well-defined meaning to support effective data discovery and integration. The Semantic Web will provide the necessary infrastructure for Web pages, database systems, services, scripts, sensors,etc., to consume and produce data on the Web. For the Semantic Web to function, the information on the Web should become more machine-understandable. For this reason, new languages and models have been proposed to semantically enrich data on the Web. The RDF framework[3] is a foundation for processing metadata, that is data for the meaning of data. In an RDF document, one can make statements about particular Web resources, that is Web pages, page authors, scripts, etc. The RDF

[3] http://www.w3c.org/RDF

schema [14] provides mechanisms for describing groups of related resources and the relationships between these resources, acting as a semantic extension of RDF. The RDF schema description language is based on *classes* and *properties*, and is similar to the type system of object-oriented programming languages.

The RDF framework is a key issue for the Semantic Web. It can be used in resource discovery to provide better search engine capabilities, in cataloging for describing the content of thematic hierarchies in thematic catalogs and digital libraries, in knowledge sharing and exchange of Web agents, etc. In [6] benchmarks are presented to provide structural and statistical analysis for volumes of RDF data collected from the Web.

Up to now, RDF schemas have been treated rather as sets of individual elements (i.e. model primitives like classes, properties, etc.). However, in the Web environment, where searching in a knowledge domain requires information processing in many sources related to that domain, new query requirements arise for manipulating RDF schemas as a whole, like for example (Q_1) `find the part of Movie Catalog` 1 `which is not present in Movie Catalog` 2 (see Figure 1(a) with examples of Movie Catalogs). Such a query has a 'difference' flavor

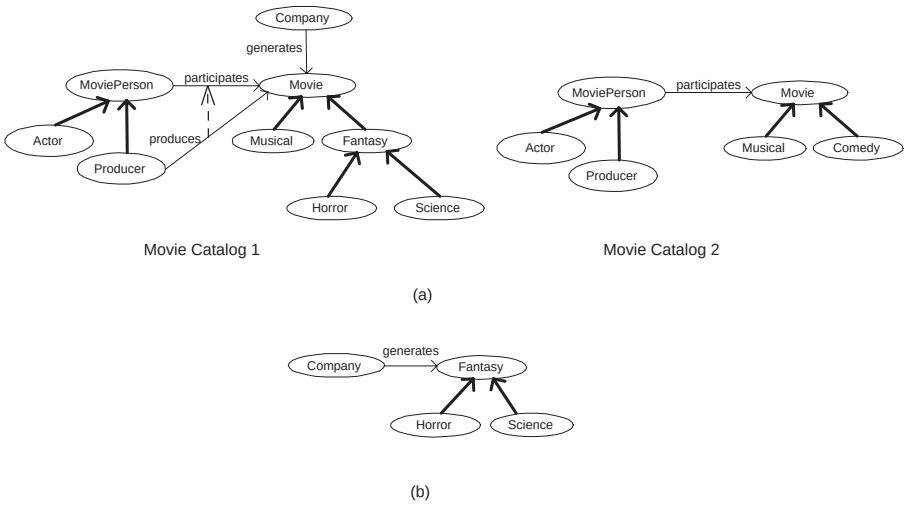

Fig. 1. (a) Parts of Movie Catalog 1 and 2. (b) The part of Movie Catalog 1 which is not present in Movie Catalog 2.

and its answer should include schema information present in Movie Catalog 1 but not in Movie Catalog 2. Figure 1(b) shows the result of Q_1. The resulting catalog has `Company`, `Fantasy`, `Horror` and `Science`, a categorization found only in Movie Catalog 1. Other example queries follow:

– (Q_2) `find the integrated catalog provided by Movie Catalog` 1 `and Movie Catalog` 2 (with a 'union' flavor),

– (Q_3) `find the common part of Movie Catalog 1 and Movie Catalog 2` (with an 'intersection' flavor),
– (Q_4) `find the integrated catalog using the common part of Movie Catalog 1 and Movie Catalog 2, and another third Movie Catalog` (a sequence of 'intersection' and 'union' subqueries).

Viewing the RDF schemas as a set of individual elements requires the usage of RDF query languages like RQL [5] in a procedural way. The user should specify which RDF nodes to select and how to get them to answer queries like Q_1, Q_2, Q_3, Q_4. For this reason, we argue that answering such queries requires treating schemas as full-fledge objects rather than as sets of individual elements, and introducing a set of operators applied on RDF schemas as a whole.

Queries like the above produce *integrated* RDF schemas. Integration of schemas, in general, is considered as the task which produces a global schema to cover all involved schemas and is a widely studied research topic [10]. Our work, on the other hand, supports the integration of RDF schemas based on union, intersection and difference semantics provided by a set of operators. The integrated schema is the output of such operators applied on the involved schemas.

We classify such kind of query requirements as part of the *generic model management* framework presented in [2, 8]. According to this framework, models are manipulated as abstractions rather than sets of individual elements, using *model-at-a-time* and *mapping-at-a-time* operators.

Contribution. This paper introduces operators to manipulate RDF schemas. The operators are applied on RDF schema graphs and produce new, integrated RDF schema graphs. The key feature of our framework is that such integration is based on set-like semantics. We define three binary operators (union, intersection, difference) that can be applied on RDF schema graphs as a whole, and produce new ones. We also define a unary operator that can be applied on one RDF schema graph and return a part (subset) of it. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. We have implemented the operators in RDFSculpt, a prototype system for managing RDF schemas.

Related Work. Recently, there have been suggested quite a few RDF query languages. RQL [5] is a typed functional language (in the form of OQL) to uniformly query both RDF data descriptions and schemas. In [7], RVL is presented as an extension of RQL that supports views on RDF. RDQL [12] is an SQL-like RDF query language, manipulating RDF data as triple patterns. Sesame [3] is an RDF management system. SeRQL (the native language of Sesame) is used to query both RDF data descriptions and schemas. Views on RDF data are provided using CONSTRUCT queries. Triple [13] is a layered and modular rule language based on Horn-logic which is syntactically extended to support features for querying and transforming RDF models. Most of the above languages support algebraic operations on RDF statements (i.e. data) and not on RDF schemas. A survey of RDF query languages is presented in [4]. The operators suggested in this paper

can be included in any query language to answer queries that require operations on RDF schemas as full-fledge objects rather than as sets of individual elements.

Operators for ontology composition have been studied in [15, 9]. These operators are based on articulation rules, that is rules to establish correspondence between concepts in different ontologies. Our work differs since we emphasize on the semantics of RDF schema graphs. Also, we compose RDF schemas which are related to a global RDF schema through the subset relation that we define, and not by using articulation rules.

Outline. The rest of this paper is structured as follows. In Section 2, we discuss modelling issues for RDF schemas and we introduce RDF schema subsets and projections. Section 3 defines three RDF schema operators: union, intersection and difference. In Section 4, we describe the RDFSculpt prototype system that implements the suggested operators. Finally, Section 5 concludes this paper and discusses further work.

2 Modelling Issues

RDF schemas provide a type system for RDF. The primitives of RDF schemas are classes and properties. Classes describe general concepts and entities. Properties describe the characteristics of classes. They also represent the relationships that exist between classes. Classes and properties are primitives similar to those of the type system of object-oriented programming languages. The difference is that properties in RDF schemas are considered as first-class citizens and are defined independently from classes.

Classes are described using the RDF schema resources `rdf:Class` and `rdfs:subClassOf`, while properties are described using the RDF class `rdf:Property` and the property `rdfs:subPropertyOf`. The `rdfs:domain` property is used to indicate that a particular property applies to a designated class. The `rdfs:range` property is used to indicate that the values of a particular property are instances of a designated class. RDF schemas can be modelled as directed labelled graphs. For example, consider the graph shown in Figure 2. The oval labelled nodes represent classes. The rectangular labelled nodes denote literals, like string, integer, etc. The plain labelled edges represent properties. The thick edges define an *isA* hierarchy (class/subclass) of classes, while the thick, dashed edges define an *isA* hierarchy (property/subproperty) of properties. For example, the class *Actor* is a subclass of $MoviePerson$, while the property *plays* is a subproperty of *participates*. The class $MoviePerson$ is the domain of the property *participates*, while the class $Movie$ is the range of *participates*. Formally, an RDF schema is defined as follows:

Definition 1. *An RDF schema (RDFS) is a 5-tuple* (C, L, P, SC, SP) *representing a graph, where:*

1. *C is a set of labelled nodes. Each node in C represents an RDF class.*

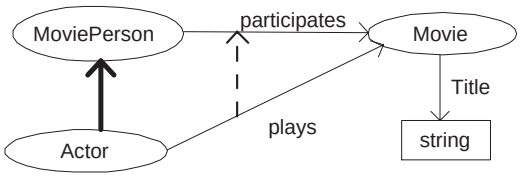

Fig. 2. An example of an RDF schema

2. *L is a set of nodes labelled with data types defined in XML schema [11], e.g. integer, string etc. Each node in L represents a literal.*
3. *P is a set of directed labelled edges* (c_1, c_2, p) *from node* c_1 *to node* c_2 *with label p, where* $c_1 \in C$ *and* $c_2 \in C \cup L$*. Each edge in P represents an RDF property p with domain* c_1 *and range* c_2*.*
4. *SC is a set of directed edges* (c_1, c_2) *from node* c_1 *to node* c_2*, where* $c_1, c_2 \in C$*. Each edge in SC represents an isA relationship between classes* c_1 *and* c_2 *(i.e.* c_1 *is a subclass of* c_2*).*
5. *SP is a set of directed edges* $((c_1, c_2, p_1), (c_3, c_4, p_2))$ *from edge* (c_1, c_2, p_1) *to edge* (c_3, c_4, p_2)*, where* (c_1, c_2, p_1)*,* $(c_3, c_4, p_2) \in P$*. Each edge in SP represents an isA relationship between property* (c_1, c_2, p_1) *and property* (c_3, c_4, p_2) *(i.e. that is* (c_1, c_2, p_1) *is a subproperty of* (c_3, c_4, p_2)*).*

Let $\preceq_C$ be a relation on C: $c_1 \preceq_C c_2$ holds if c_1 is a subclass of c_2. With $\preceq_C^+$ we denote the transitive closure of $\preceq_C$. We consider c_1 to be an *ancestor* of c_2 (or c_2 to be a *descendant* of c_1) if $c_2 \preceq_C^+ c_1$. Similarly, let $\preceq_P$ be a relation on P: $(c_1, c_2, p_1) \preceq_P (c_3, c_4, p_2)$ holds if (c_1, c_2, p_1) is a subproperty of (c_3, c_4, p_2). With $\preceq_P^+$ we denote the transitive closure of $\preceq_P$. We consider (c_1, c_2, p_1) to be an *ancestor* of (c_3, c_4, p_2) (or (c_3, c_4, p_2) to be a *descendant* of (c_1, c_2, p_1)) if $(c_3, c_4, p_2) \preceq_P^+ (c_1, c_2, p_1)$.

2.1 RDF Schema Subsets

We next introduce the concept of the *subset* relation for RDF schemas. Intuitively, an RDF schema R_1 is a subset of an RDF schema R_2 when R_1 contains some of the elements (i.e. classes, properties, etc.) of R_2, and it does not violate the isA hierarchy of classes and properties maintained in R_2.

Definition 2. *Let* $R_i = (C_i, L_i, P_i, SC_i, SP_i)$ *and* $R_j = (C_j, L_j, P_j, SC_j, SP_j)$ *be two RDF schemas.* R_i *is a subset of* R_j*, denoted by* $R_i \subseteq R_j$*, if:*

1. $C_i \subseteq C_j$*.*
2. $L_i \subseteq L_j$*.*
3. *for each edge* $(c_1, c_2, p_1) \in P_i$ *there is an edge* $(c_3, c_4, p_2) \in P_j$ *with* $(c_1 \equiv c_3$ *or* $c_1 \preceq_{C_j}^+ c_3)$ *and* $(c_2 \equiv c_4$ *or* $c_2 \preceq_{C_j}^+ c_4)$ *and* $p_1 = p_2$*.*

4. *for each pair of nodes* $c_1, c_2 \in C_i$,
   if $c_1 \preceq_{C_i} c_2$ *then* $c_1 \preceq^+_{C_j} c_2$ *and*
   if $c_1 \preceq^+_{C_j} c_2$ *then* $c_1 \preceq^+_{C_i} c_2$.
5. *for each pair of edges* $(c_1, c_2, p_1), (c_3, c_4, p_2) \in P_i$,
   if $(c_1, c_2, p_1) \preceq^+_{P_i} (c_3, c_4, p_2)$ *then* $(c_1, c_2, p_1) \preceq^+_{P_j} (c_3, c_4, p_2)$ *and*
   if $(c_1, c_2, p_1) \preceq^+_{P_j} (c_3, c_4, p_2)$ *then* $(c_1, c_2, p_1) \preceq^+_{P_i} (c_3, c_4, p_2)$.

Figure 3(a) shows the RDF schema R_1 which is a subset of R, since it satisfies all conditions of the definition. For example, having $C_1 = \{A, B, C, E, G\}$ and $C = \{A, B, C, D, E, F, G\}$, $C_1 \subseteq C$. Also, for each pair of nodes in C_1 the fourth condition of the above definition holds (e.g. $A \preceq_{C_1} E$ and $A \preceq^+_C E$ hold, and $A \preceq^+_C E$ and $A \preceq^+_{C_1} E$ hold as well for nodes A, E in C_1). On the other hand, the RDF schema R_2 is not a subset of R in Figure 3(b). The fourth condition of the definition is violated; although $D \preceq^+_C E$ holds, $D \preceq^+_{C_2} E$ does not hold.

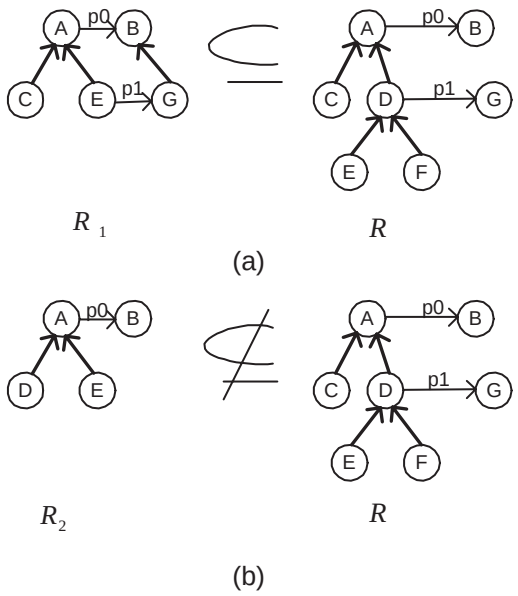

Fig. 3. Examples of RDF schema subsets.

In this work we manipulate RDF schemas which are subsets of a given RDF schema, called *global RDF schema*.

Definition 3. *Let* $\mathcal{S} = \{R_1, R_2, \ldots R_n\}$ *be a set of RDF schemas. A* global RDF schema *for* S *is an RDF schema* R *such that* $R_i \subseteq R, 1 \leq i \leq n$.

2.2 Projecting RDF Schemas

This section defines the operator of projection on RDF schemas. Projecting RDF schemas is based on a given set of RDF classes and involves the extraction of a part of an RDF schema that includes at least those classes. Before we present the projection operator in detail, we give some definitions which are useful to the discussion that will follow. All subsequent definitions refer to an RDF schema $R = (C, L, P, SC, SP)$.

Definition 4. *The* extended domain *of a property* $(c, s, p) \in P$*, denoted by* $\mathcal{D}^+((c, s, p))$*, is the set of classes* $\{c, c_1, \ldots c_n\}$*, where* $\{c_1, \ldots c_n\}$ *are all descendants of* c*.*

Using the *extended domain* of a property we refer to all classes which can be applied to a property as a set. For example, in the RDF schema of Figure 2 we have $\mathcal{D}^+(MoviePerson,\ Movie, participates) = \{Movieperson, Actor\}$. Similarly, we define the *extended range* of a property to refer to all the classes from which a property can take values as a set.

Definition 5. *The* extended range *of a property* $(e, c, p) \in P$*, denoted by* $\mathcal{R}^+((e, c, p))$*, is the set of classes* $\{c, c_1, \ldots c_n\}$*, where* $\{c_1, \ldots c_n\}$ *are all descendants of* c*.*

Below we define the nearest common ancestor of a set of classes. Intuitively, it is the class which is the lower-level ancestor of all classes in the set. For example, $nca\{G, H, F\} = B$ in R_G of Figure 5.

Definition 6. *The* nearest common ancestor *of a set of classes* $C_s \subseteq C$*, where* C_s *consists of more than two classes, denoted by* $nca(C_s)$*, is the class* z *such that for all* $x \in C_s$ $x \preceq_C^+ z$ *holds and there is no class* $y \in C$ *such that* $x \preceq_C^+ y \preceq_C^+ z$*. The* nearest common ancestor *of one class is the class itself.*

We now define the projection operator for RDF schemas. Intuitively, a projection on an RDF schema R, given a set of classes C_s, results in a subset of R that has all classes from C_s, their involved properties and some other classes, the role of which will be clarified shortly.

Consider the RDF schema R in Figure 4. Projecting R with $C_s = \{C, D, B\}$ results in an RDF schema which includes classes C, D, B and the involved property $(D, B, p2)$. Class A (and its involved property (A, B, p_1)) is also part of the result although $A \notin C_s$. In general, classes like A (which are actually nearest common ancestors for classes that are included in C_s) are used to resolve the issue of having more than one classes as domain (or range) for a property. Another example of projection is presented in Figure 5. The formal definition for the projection operator follows.

Definition 7. *Let* $R = (C, L, P, SC, SP)$ *be an RDF schema and* $C_s \subseteq C$ *a set of classes. The* projection Π *on* R*, given* C_s*, denoted as* $\Pi_{C_s}(R)$*, is the RDF schema* $R' = (C', L', P', SC', SP')$*, where:*

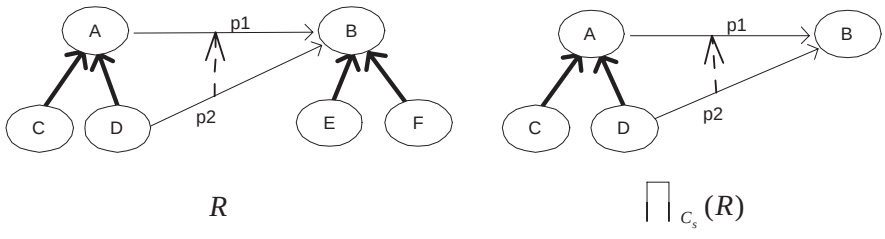

Fig. 4. Projecting R with $C_s = \{C, D, B\}$.

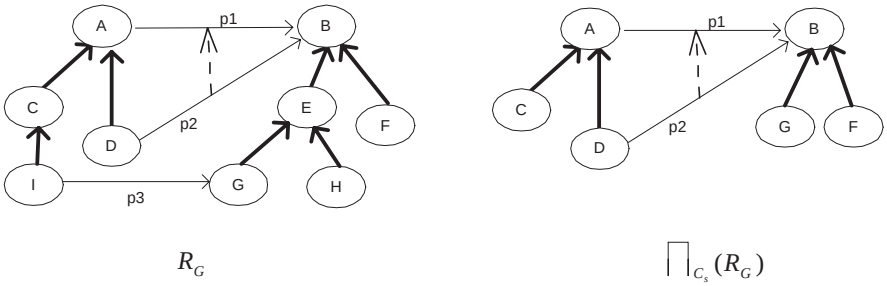

Fig. 5. Projecting R_G with $C_s = \{C, D, G, F\}$.

1. $(c'_1, c'_2, p') \in P'$ *if* $\exists (c_1, c_2, p) \in P$, *with* $C_s \cap \mathcal{D}^+((c_1, c_2, p)) \neq \emptyset$ *and* $C_s \cap \mathcal{R}^+((c_1, c_2, p)) \neq \emptyset$, *where*
   (a) c'_1 *is the* $nca(C_s \cap \mathcal{D}^+((c_1, c_2, p)))$,
   (b) c'_2 *is the* $nca(C_s \cap \mathcal{R}^+((c_1, c_2, p)))$, *and*
   (c) $p' = p$.
2. $(c'_1, c'_2) \in SC'$ *if* $c'_1 \preceq^+_C c'_2$.
3. $((c'_1, c'_2, p'_1), (c'_3, c'_4, p'_2)) \in SP'$ *if* $(c'_1, c'_2, p'_1) \preceq^+_P (c'_3, c'_4, p'_2)$.
4. $C' = C_s \cup C_a$, *where* $C_a = (\cup_i c'_i) \cup (\cup_i s'_i)$, *for all* $(c'_i, s'_i, p'_i) \in P'$.
5. $L' = \{l \in L |\ \exists c \in C_s$ *and* $(c, l, p) \in P\}$.

3 Set-like RDF Schema Operators

We define three binary operators applied on RDF schema graphs. The operators can be included in any RDF query language and support manipulation of RDF schemas as full-fledged objects, under union, intersection and difference semantics. In all presented examples, the RDF schemas are subsets of R_G illustrated in Figure 5.

3.1 Union

The *union* operator merges two RDF schemas and results in a new RDF schema that contains all elements from both schemas, without violating the isA hierarchies for the involved classes and properties. The union operator for two RDF schemas R_1 and R_2 is defined as a projection on the global schema R_G, given the (set) union of class sets of R_1 and R_2, respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

Definition 8. *Let* $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ *and* $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ *be two RDF schemas with* $R_1, R_2 \subseteq R_G$ *and* $C = C_1 \cup C_2$. *The* union *of* R_1 *and* R_2, *denoted by* $R_1 \cup R_2$, *is the RDF schema* $R = \Pi_C(R_G)$.

Figure 6 shows an example of union operation.

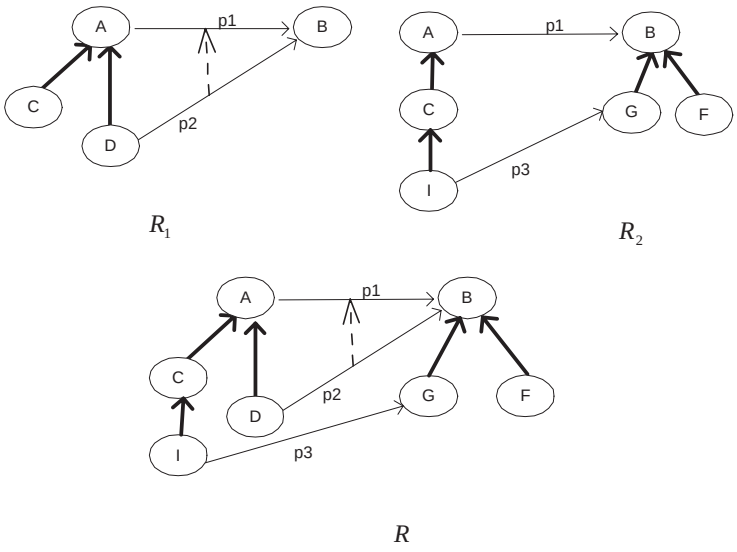

Fig. 6. An example of union operation: $R = R_1 \cup R_2$.

3.2 Intersection

The *intersection* operator results in a new RDF schema that contains only common elements from both schemas, keeping the isA hierarchies for the involved classes and properties. The intersection operator for two RDF schemas R_1 and R_2 is defined as a projection on the global schema R_G, given the (set) intersection of class sets of R_1 and R_2, respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

Definition 9. *Let* $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ *and* $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ *be two RDF schemas with* $R_1, R_2 \subseteq R_G$ *and* $C = C_1 \cup C_2$*. The* intersection *of* R_1 *and* R_2*, denoted by* $R_1 \cap R_2$*, is the RDF schema* $R = \Pi_C(R_G)$*.*

Figure 7 shows an example of intersection operation.

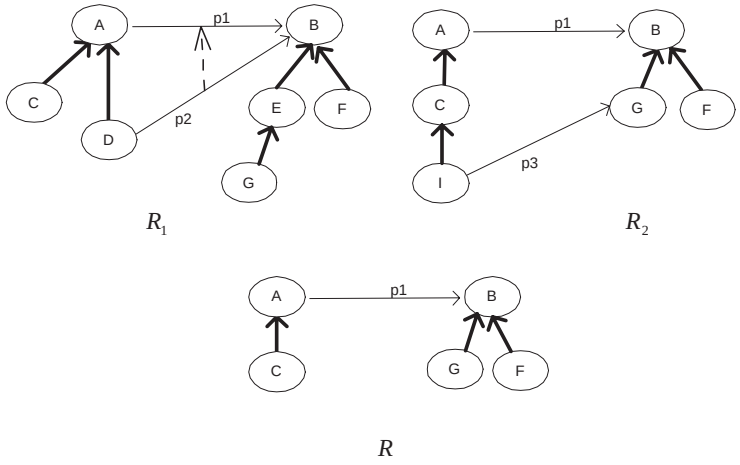

Fig. 7. An example of intersection operation: $R = R_1 \cap R_2$.

3.3 Difference

The *difference* operator results in a new RDF schema that contains elements of one schema which are not present in another one, keeping the isA hierarchies for the involved classes and properties. The difference operator for two RDF schemas R_1 and R_2 is defined as a projection on the global schema R_G, given the (set) difference of class sets of R_1 and R_2, respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

Definition 10. *Let* $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ *and* $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ *be two RDF schemas with* $R_1, R_2 \subseteq R_G$ *and* $C = C_1 - C_2$*. The* difference *of the two RDF schemas, denoted by* $R_1 - R_2$*, is the RDF schema* $R = \Pi_C(R_G)$*.*

Figure 8 shows an example of difference operation.

4 The RDFSculpt Prototype

The RDFSculpt is a prototype system for RDF schema management and implements the operators suggested in this paper. As shown in Figure 9, the system

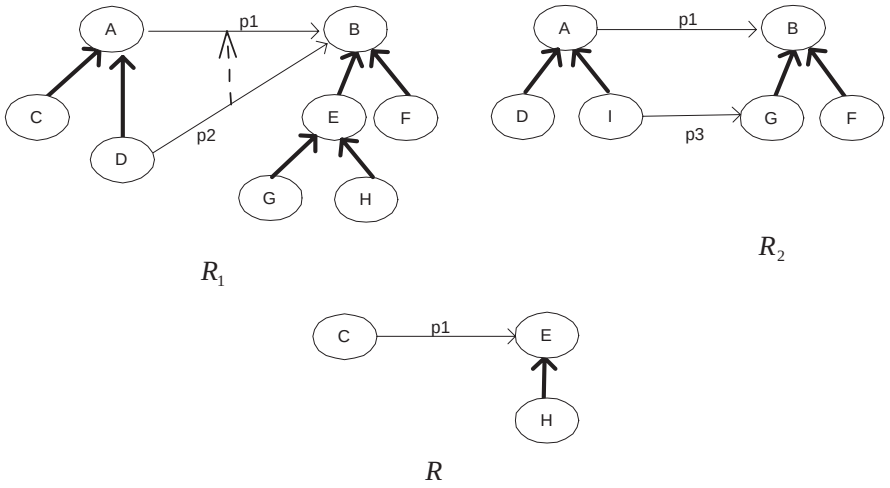

Fig. 8. An example of difference operation: $R = R_1 - R_2$.

is built on top of the ICS-FORTH RDFSuite[4]. To this extend, it exploits all RDF management and query APIs offered by RDFSuite. Users can issue queries on RDF schemas and produce new, integrated ones, using projection, union, intersection and difference operators. The RDFSculpt assists the user in queries formulation, offering her query-by-example capabilities. Results are visualized using RDFSViz[5]. Figure 10 shows some screen shots of the system.

4.1 Query Processing in RDFSculpt

We next describe in detail how RDFSculpt executes a projection operator on an RDF schema.

Algorithm projection(C_s)
/* D, $\mathcal{D}^+$, R, $\mathcal{R}^+$ denote property domain, extended domain, range and extended range, respectively */

1 for each class c in C_s do
2 P =properties that have class c in their $\mathcal{D}^+$;
3 for each property p in P do
4 R = classes that are in the $\mathcal{R}^+$ of p;
5 $R = R \cap C$;
6 endfor
7 for each p in P do

[4] http://www.ics.forth.gr/isl/RDF/index.html
[5] http://www.dfki.uni-kl.de/frodo/RDFSViz/

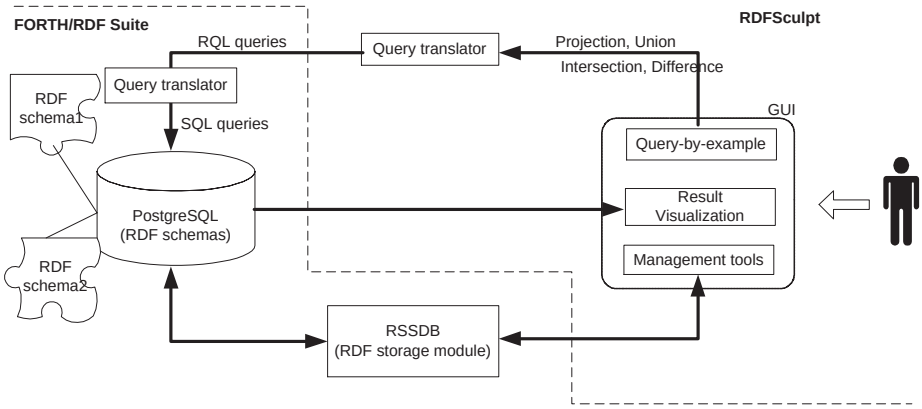

Fig. 9. The architecture of RDFSculpt

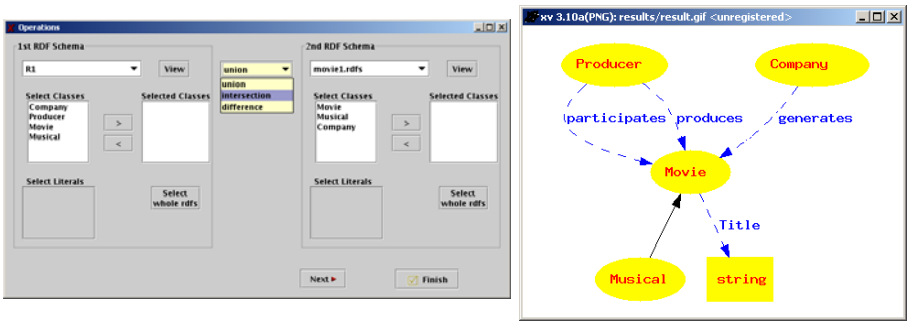

Fig. 10. Screen shots of the RDFSculpt prototype system.

8 /∗ Working in the domain of p ∗/
9 if there is only one class in $\mathcal{D}^+$ of p, then have this class as D of p
10 else
11 let R' be the classes in $\mathcal{D}^+$ of p;
12 if all classes in R' are in the same path of the isA hierarchy
13 then have the higher-level one as the domain of p
14 else
15 find the nearest common ancestor of R', have it as D of p,
16 and add it in C_s;
17 endif
18 endif
19 /∗ Working in the range of p ∗/
20 if there is only one class in $\mathcal{R}^+$ of p, then have this class as R of p
21 else
22 let R' be the classes in $\mathcal{R}^+$ of p;

23 if all classes in R' are in the same path of the isA hierarchy
24 then have the higher-level one as the range of p
25 else
26 find the nearest common ancestor of R', have it as R of p,
27 and add it in C_s;
28 endif
29 endif
30 endfor
31 endfor

For example, consider the projection shown in Figure 4, where $C_s = \{B, C, D\}$. For class $c = D$ (line 1) we get $P = \{p_1, p_2\}$ (line 2). For property p_1 (line 3) we get $R = \{B, E, F\}$ (line 4), and after the intersection with C_s we get $R = \{B\}$ (line 5). Working in the domain of p_1, we have $\mathcal{D}^+ = \{C, D\}$. Therefore, there is more than one classes in $\mathcal{D}^+$ (line 10) and those classes ($\{C, D\}$) are not in the same path (line 14). Consequently, we find the nearest common ancestor of C and D (line 15), which is class A, and add it in C_s (line 16).

Projection is implemented by posing a set of appropriate RQL queries to the RDF storage module. Below we show some examples of RQL queries used to implement projection (given the set of classes $C_s = \{B, C, D\}$) for some steps of the previous algorithm. Details about the RQL language can be found in [5]. For convenience, we note here that the prefix $ denotes a class variable, the prefix @ a property variable and the expression $\{; B\}$ denotes a filtering condition of schema classes, taking into account the rdfs:subClassOf links. For instance, the path expression $\{; D\}@P$ denotes that the domain of $@P$ is denoted to be class D or any of its superclasses. Furthermore, $nca(B, C)$ is a function of RQL which finds the nearest common ancestor of two nodes.

- In line 2, in order to find the properties of a specific domain, let it be B, we use the RQL query:
  `select` $@P$ `from` $\{; B\}@P\{\$C\}$
- In line 4, in order to find the classes in $\mathcal{R}^+$ of a specific property, let it be p, we use the RQL query:
  `select` $\$C$ `from` $p\{\$C\}$
- In line 12, in order to check if classes in $\mathcal{D}^+$ are in the same path, we make use of the RQL queries:
  `domain(`p`)`
  `superClassOf(`B`)`
- In line 15, in order to find the nearest common ancestor of R', we make use of the RQL query:
  `nca(`B, C`)`
- In line 23, in order to check if classes in $\mathcal{R}^+$ are in the same path, we make use of the RQL queries:
  `range(`p`)`
  `superClassOf(`B`)`

– In line 26, in order to find the nearest common ancestor of R', we make use of the RQL query:
`nca(`B, C`)`

The union, intersection and difference operators are implemented as projections, with C_s being the (set) union, (set) intersection and (set) difference of class sets in the involved RDF schemas, respectively.

We should note here that the existence of blank nodes in an RDF schema does not affect our approach. A blank node does not have a URI. However, it should have a unique identifier to distinguish itself from other blank nodes in the RDF schema. In that case, blank nodes can be treated as classes and included in the set of classes C_s (see Definition 7).

5 Conclusion

This paper introduced a set of operators to manage RDF schemas. They are applied on RDF schema graphs and produce new, integrated RDF schema graphs. Such integration is based on set-like semantics. Specifically, we formalized the notion of RDF schema subsets, and we defined a unary operator (projection) to extract parts (subsets) of RDF schemas. Based on projection, we defined three binary operators: union, intersection and difference. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. Finally, we described RDFSculpt, a prototype system for managing RDF schemas that implements our framework.

We are currently working towards the following directions. We are first studying the algebraic properties of the presented operators to show formally that they obey all known laws of set theory. The other research direction involves extending our framework to manage RDF schemas under the assumption that the global RDF schema is not given, but it should be constructed from the available RDF schemas. Furthermore, we are planning to layer our operations to deal with other RDF vocabularies and not only RDF schema graphs.

References

1. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.
2. Philip A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, USA, 2003.
3. Jeen Broedstra, Arjohn Kampman, and Frank van Harmelen. Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Chia, Sardinia, Italy, 2002.
4. Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A Comparison of RDF Query Languages. In *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, Hiroshima, Japan, 2004.

5. Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th International World Wide Web Conference (WWW'02)*, Honolulu, Hawaii, USA, 2002.
6. Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, and Dimitris Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Chia, Sardinia, Italy, 2002.
7. Aimilia Magkanaraki, Val Tannen, Vassilis Christophides, and Dimitris Plexousakis. Viewing the Semantic Web through RVL Lenses. *Journal of Web Semantics*, 1:4, 2004.
8. Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *Proceedings of the Special Interest Group on Management of Data (SIGMOD) Conference (ACM SIGMOD 2003)*, 2003.
9. Prasenjit Mitra and Gio Wiederhold. An Algebra for Semantic Interoperability of Information Sources. In *Proceedings of the IEEE Symposium on BioInformatics and Bioengineering (BIBE'01)*, Bethesda, MD, Nov. 2001.
10. Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.
11. W3C Recommendation. XML Schema Part 2: Datatypes Second Edition, 2004. http://www.w3.org/TR/xmlschema-2/.
12. Andy Seaborne. RDQL - A Query Language for RDF. W3C member submission, January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
13. Michael Sintek and Stefan Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the Deductive Databases and Knowledge Management Workshop (DDLP 2001)*, Tokyo, Japan, 2001.
14. W3C. Resource Description Framework (RDF) Schema Specification 1.0, 2001. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.
15. Gio Wiederhold. An Algebra for Ontology Composition. In *Proceedings of the Monterey Workshop on Formal Methods*, U.S. Naval Postgraduate School, Monterey CA, 1994.