# On Explicit Provenance Management in RDF/S Graphs

P. Pediaditis
*ICS-FORTH*
*University of Crete*
*Greece*
*pped@ics.forth.gr*

G. Flouris
*ICS-FORTH*
*Greece*
*fgeo@ics.forth.gr*

I. Fundulaki
*ICS-FORTH*
*Greece*
*fundul@ics.forth.gr*

V. Christophides
*ICS-FORTH*
*University of Crete*
*Greece*
*christop@ics.forth.gr*

## Abstract

The notion of RDF Named Graphs has been proposed in order to assign provenance information to data described using RDF triples. In this paper, we argue that named graphs alone cannot capture provenance information in the presence of RDFS reasoning and updates. In order to address this problem, we introduce the notion of *RDF/S Graphsets*: a graphset is associated with a set of RDF named graphs and contain the triples that are *jointly owned* by the named graphs that constitute the graphset. We formalize the notions of RDF named graphs and RDF/S graphsets and propose query and update languages that can be used to handle provenance information for RDF/S graphs taking into account RDFS semantics.

## 1 Introduction

An increasing number of scientific communities (such as Bioinformatics [21, 22] and Astronomy [13]) rely on common terminologies and reference models related to their subject of investigation in order to share and interpret scientific data world-wide. Scientists, acting as curators, rely on a wide variety of resources to build conceptual models of increasing formality (e.g., taxonomies, reference models, ontologies) related to their subject of study. These models are usually developed and maintained manually and co-evolve along with experimental evidence produced by scientific communities worldwide. To enforce sharing and reusability, these knowledge representation artefacts are nowadays published using Semantic Web (SW) languages such as RDF or OWL, and essentially form a special kind of curated databases [3]. The popularity of the RDF data model [6] and RDF Schema language (RDFS) [1] among scientific communities is due to the flexible and extensible representation of both schema-relaxable and schema-less information under the form of *triples*.

An RDF triple, $(subject, property, object)$, asserts the fact that *subject* is associated with *object* through *property*. A collection of (data and schema) triples forms an *RDF/S graph* whose nodes represent either information resources universally identified by a *Universal Resource Identifier (URI)* or *literals*, while edges are properties, eventually defined by one or more associated RDFS vocabularies (comparable to relational and XML schemas). In addition, *RDF Named Graphs* have been proposed in [5, 28] to capture *explicit* provenance information by allowing users to refer to specific parts of RDF/S graphs in order to decide *"how credible is"*, or *"how evolves"* a piece of information. Intuitively, an RDF named graph is a collection of triples associated with a URI which can be referenced by other graphs as a normal resource; this way, one can assign explicit provenance information to this collection of triples.

RDFS is used to add semantics to RDF triples, by imposing *inference rules* [12] (mainly subsumption relationships) which can be used to entail new *implicit* triples (i.e., facts) which are not explicitly asserted. There are two different ways in which such implicit knowledge can be viewed and this affects the assignment of provenance information to such triples, as well as the semantics of the update operations. Under the *coherence* semantics [8], implicit knowledge is not depending on the explicit one but has value on its own; therefore, there is no need for explicit "support" of some triple. Under this viewpoint, implicit triples are "first-class citizens", i.e., considered of equal value as explicit ones. On the other hand, under the *foundational* semantics [8], implicit knowledge is only valid as long as the supporting explicit knowledge is there. Therefore, each implicit triple depends on the existence of the explicit triple(s) that imply it. In this work, we assume coherence semantics.

Currently, there is no adequate support for querying and updating RDF/S graphs that takes into account both RDF named graphs and RDFS inference. In particular, existing declarative query and update languages for RDF have been extended either with named graphs support (such as Sparql [19] and Sparql Update [25]), or with RDFS inference support [18, 20], but not with both.

In this paper, we introduce *RDF/S graphsets* in order to cope with RDFS reasoning issues while querying and updating logical modules of RDF/S graphs. An *RDF/S graphset* is defined using a *set of RDF named graphs*, and is itself associated with a URI and with a *set of triples* whose *ownership is shared by the named graphs that constitute the graphset*. The main objective behind the introduction of this construct is *i)* to preserve provenance information that would otherwise be lost in the presence of *updates* and *ii)* to record *joint ownership* of facts, something that is not possible with the use of named graphs only.

## 1.1 Problem Statement

We will use, for illustration purposes, an example taken from a bioinformatics application. The RDFS schema of our biological example (see Figure 1) captures information related to *diseases*, *receptors* and *ligands*, as well as the *relationships* between them, and is contributed by several curated databases, *each one represented with one named graph*. For illustration purposes, we use the name of the curated database as the URI of its corresponding named graph.

Figure 1 shows the graph obtained from the triples of sources $S_1$, $S_2$ and $S_3$. The building blocks of an RDFS vocabulary are *classes* and *properties* (binary relations between classes). Since RDF/S graphs can be seen as a kind of *labeled directed graphs*, we use the following graphical notation: classes are represented with boxes, and their instances are presented as ovals and contain their URI reference. To distinguish between individual resources and classes, we prefix a URI with the "&" symbol. RDFS built-in properties [1] subclassOf, type and subpropertyOf are represented by dashed, dotted and dotted-dashed arrows respectively. If a triple $(s,\ p,\ o)$ belongs to a named graph whose URI is $n$ we write $s \xrightarrow{p\ (n)} o$. For instance, the triple $(\&dopamine\_receptor\_D2$,type,-$Neurotransmitter\ Receptor)$ is provided by source $S_1$, whereas the triple $(Neurotransmitter\ Receptor$,sc,-$Cell\ Receptor)$ is provided by source $S_2$.

Not surprisingly, a great part of the information captured by an RDF/S graph can be inferred by the transitivity of class (and property) subsumption relationships stated in the associated RDFS schemas. For instance, although not explicitly asserted, from the graph of Figure 1 we can infer the triple $(\&dopamine\_receptor\_D2$,type,-$Cell\ Receptor)$, because class *Neurotransmitter Receptor* is a subclass of class *Cell Receptor*. However, this triple does not belong to any of the considered so far named graphs. In terms of provenance, we view the origin of this triple as *composite* (i.e., being *shared* by two or more different sources). In our example, we need to combine triples from sources $S_1$ and $S_2$ to derive triple $(\&dopamine\_receptor\_D2$,type,$Cell\ Receptor)$. Shared origin (or ownership) cannot be captured by RDF named graphs alone.

Unfortunately, shared ownership cannot be captured by a *set-theoretic-based union* of the involved named graphs either. Such a union would contain all triples of both named graphs, and, as a consequence, all triples computed by applying the RDFS inference rules on this set of triples. The contents of the union are totally determined by the contents of its operands.

This has two undesirable (and related) consequences. The first is that one cannot explicitly assert triples to belong to a union, since the result of the union is determined by the operands; thus, an explicit triple cannot be asserted to be of shared origin, but should belong to some individual named graph.

The second consequence is related to updates, which are common practice in the context of curated databases. Recall that, under coherence semantics, implicit triples are of equal value as explicit ones and should not be deleted when their support is lost; thus, when deleting a triple $t$, we want to retain the implicit triples that were inferred when $t$ was asserted in order to *preserve as much information as possible*. For instance, consider that the experimental evidence that $\&dopamine\_receptor\_D2$ is an instance of class *Neurotransmitter Receptor* from source $S_1$, was erroneous and this triple is deleted. Nevertheless, we wish to retain that $\&dopamine\_receptor\_D2$ is an instance of class *Cell Receptor*. In this case, we need to associate this triple with *a set of named graphs* (namely, $\{S_1, S_2\}$) to record that these named graphs *share the ownership* of the triple.

Note that the above problems are not specific to the union operator, but would appear in any operator-based formalization of shared ownership. What we need here is a *first-class construct* that would capture shared ownership independently, but without losing the connection with the sources (named graphs) that compose the structure. This is the purpose of the RDF/S *graphsets* machinery that is introduced here.

As a side remark, we can note that the later re-addition of the information that $\&dopamine\_receptor\_D2$ is an instance of class *Neurotransmitter Receptor* by source $S_1$ would result to the restoration of the original RDF/S graph. This means that our model allows the identification of the data being deleted and subsequently added, unlike standard provenance models (e.g., [2]), where successive deletions and additions of the same data result to

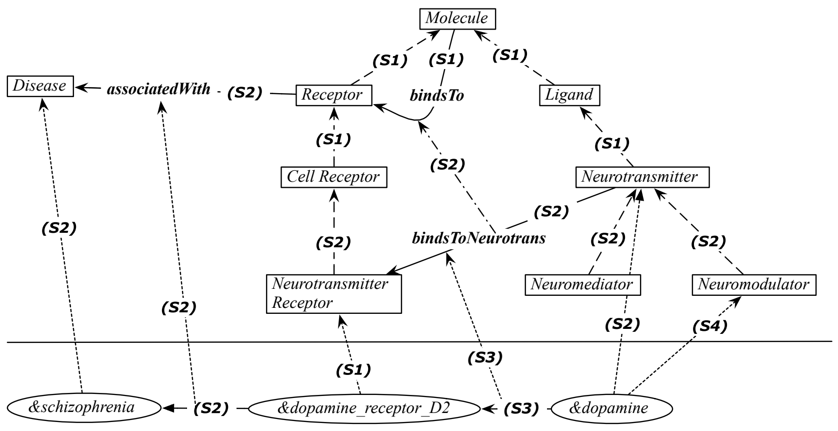


Figure 1: Collaborative Neurobiology Ontology

loss of provenance information due to the generation of a new identifier in each addition. On the contrary, in our context, we consider that two triples are identical when they carry the same *information*; this policy is supported (and imposed) by the fact that the constituents of triples (i.e., the resources) are uniquely identified by their URI, so triples with the same content are identical.

The main contributions of our work are: *i)* the formalization of the notion of RDF/S graphsets to record and reason about provenance information for RDF/S graphs and *ii)* the elaboration of the semantics of query and update languages for RDF/S graphs in the presence of RDFS inference.

## 2 Preliminaries

As already mentioned, in the RDF data model [6], the universe of discourse is a set of *resources*. A resource is essentially anything that can have a URI. Resources are described using *binary predicates* which are used to form descriptions (*triples*) of the form $(subject, predicate, object)$: a subject denotes the described resource, a predicate denotes a resource's property, and an object the corresponding property's value. The predicate is also a resource, while an object can be a resource or a literal value. We consider two disjoint and infinite sets $\mathbf{U}$, $\mathbf{L}$, denoting the URIs and literals respectively.

**Definition 1** *An RDF triple* $(subject,predicate,object)$ *is any element of the set* $\mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$.

The RDF Schema (RDFS) language [1] provides a built-in vocabulary for asserting user-defined schemas in the RDF data model. For instance, the RDFS names *Resource* [res], *Class* [class] and *Property* [prop] could be used as objects of triples describing *class* and *property* types. Furthermore, one can assert *instance of* relationships of resources with the RDFS predicate rdf:type [type], while *subsumption* relationships among classes and properties are expressed with the RDFS subclassOf [sc] and subpropertyOf [sp] predicates respectively. In addition, RDFS domain [domain] and range [range] predicates allow one to specify the domain and range to which properties can apply. In the rest of this paper, we consider two disjoint and infinite sets of URIs of classes ($\mathbf{C} \subset \mathbf{U}$) and property types ($\mathbf{P} \subset \mathbf{U}$).

It should be finally stressed that RDFS schemas are essentially descriptive and not prescriptive, designed to represent data. We believe that this flexibility in representing schema-relaxable (or schema-less) information, is the main reason for RDF and RDFS popularity. Using the uniform formalism of RDF triples, we are able to represent in a flexible way both schema and instances in the form of RDF/S graphs. It should be noted that RDF/S graphs are not classical *directed labeled graphs*, because, for example, an RDFS predicate (e.g., subpropertyOf) may relate other predicates (e.g., $bindsTo$ and $bindsToNeuroTrans$). Thus, the resulting structure is not a graph in the strict mathematical sense. An RDF/S graph can be assigned a URI and a collection of such graphs forms an *RDF Dataset* as defined in [19].

To capture the fact that a triple belongs to a particular RDF/S graph, we extend the notion of triple as follows:

**Definition 2** *An RDF quadruple* $(subject, predicate, object,graph)$ *is any element of the set* $\mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L}) \times \mathbf{U}$. *We denote by* $\mathcal{D}$ *the set of quadruples.*

Using this definition, we can define the notion of an RDF Dataset featuring several graphs as follows:

**Definition 3** *An RDF Dataset $d$ is a finite set of quadruples in $\mathcal{D}$ ($d \subseteq \mathcal{D}$).*

## 3 RDF Named Graphs

Intuitively, an RDF named graph is defined by a *set of triples* to which we have explicitly assigned an identifier (URI). We denote with $\mathbf{N} \subset \mathbf{U}$ the set of named graph URIs.

**Definition 4** *A named graph $g_n$ identified by a URI $n \in \mathbf{N}$, is a set of quadruples in $d$ of the form $(s, p, o, g)$ such that $g = n$.*

**RDFS Inference for Named Graphs:** The RDFS specification [12] relies on a set of *inference rules* which, when applied to a set of triples, entail knowledge which was not explicitly specified. We extend those inference rules for sets of quadruples. The result shown in Table 1 is a straightforward extension of the RDFS inference rules discussed in [11]. For instance, rule $I_n^{(2)}$ defines the transitivity of the sc RDFS predicate: if a class $C_1$ is a subclass of $C_2$ and $C_2$ a subclass of $C_3$ in named graph $n_1$, then we infer that $C_1$ is a subclass of $C_3$ in $n_1$. The remaining rules are defined in a similar manner.

| | | |
|---|---|---|
| Reflexivity of sc | $I_n^{(1)}$ : | $\dfrac{(C, \mathsf{type}, \mathsf{class}, n_1)}{(C, \mathsf{sc}, C, n_1)}$ |
| Transitivity of sc | $I_n^{(2)}$ : | $\dfrac{(C_1, \mathsf{sc}, C_2, n_1), (C_2, \mathsf{sc}, C_3, n_1)}{(C_1, \mathsf{sc}, C_3, n_1)}$ |
| Reflexivity of sp | $I_n^{(3)}$ : | $\dfrac{(P, \mathsf{type}, \mathsf{prop}, n_1)}{(P, \mathsf{sp}, P, n_1)}$ |
| Transitivity of sp | $I_n^{(4)}$ : | $\dfrac{(P_1, \mathsf{sp}, P_2, n_1), (P_2, \mathsf{sp}, P_3, n_1)}{(P_1, \mathsf{sp}, P_3, n_1)}$ |
| Transitivity of class instantiation | $I_n^{(5)}$ : | $\dfrac{(x, \mathsf{type}, C_1, n_1), (C_1, \mathsf{sc}, C_2, n_1)}{(x, \mathsf{type}, C_2, n_1)}$ |
| Transitivity of property instantiation | $I_n^{(6)}$ : | $\dfrac{(P_1, \mathsf{sp}, P_2, n_1), (x_1, P_1, x_2, n_1)}{(x_1, P_2, x_2, n_1)}$ |

Table 1: Inference Rules for RDF Named Graphs

The *closure* of an RDF named graph, as well as the employed inference rules, are as usual abstracted by a *consequence operator*, $Cn$. More formally, for a named graph $g_n$ the result of $Cn(g_n)$ contains *all the implicit and explicit* quadruples obtained by applying the rules in Table 1 until no more rules can be applied. Note that these inference rules *do not span across multiple named graphs*. We say that a named graph $g_n$ *entails* a quadruple $t = (s, p, o, n)$ iff $t$ *belongs to* the closure of $g_n$:

$$g_n \vdash t \Leftrightarrow t \in Cn(g_n)$$

In some cases, we may want to restrict entailment in order to use only *some* of the rules $I_n^1, \ldots, I_n^6$. The employed rules in such a case will be specified as a subscript of $\vdash$; for example, the symbol $g_n \vdash_{\{I_n^1, I_n^2\}} t$ means that $t$ is entailed by $g_n$ using only $I_n^1, I_n^2$. We say that two named graphs $g_n^{(1)}$ and $g_n^{(2)}$ are *identical*, denoted by $g_n^{(1)} = g_n^{(2)}$ iff they are identified by the same name.

## 4 RDF/S GraphSets

Intuitively, an RDF/S *graphset* is a set of quadruples defined either *extensionally* (by assigning them a graphset identifier), or *intentionally* (i.e., *jointly entailed* by a set of associated named graphs using the inference rules of Table 2 – see Section 5.1). The identifier of an RDF/S graphset is obtained via skolemization on the URIs (names) of its associated named graphs. Without loss of generality, we consider singletons of named graphs to be graphsets identified by the URI of the only named graph in the set (i.e., the identifier of $\{n\}$ is $n$). We denote with $\mathbf{I} \subset \mathbf{U}$ the set of graphset identifiers; obviously, $\mathbf{N} \subset \mathbf{I}$.

**Definition 5** *An RDF/S graphset $g_s$, identified by an identifier $i \in \mathbf{I}$ and associated with a set of named graphs $S$, is a set of quadruples $(s, p, o, g)$ in $d$ that (1) either are assigned identifier $i$ (2) or are* jointly entailed *by the named graphs in $S$, but not by any subset thereof. Thus: $\forall\ t = (s, p, o, g) \in g_s$, it holds that either $g = i$ or $\exists\ T_1, T_2, \ldots T_{|S|} \subseteq d$, such that for all $g_n^{(j)} \in S$, $g_n^{(j)} \vdash T_j$ and $\cup_{j=1,\ldots,|S|} T_j \vdash t$ and there does not exist quadruples in a subset $S'$ of $S$ that entail $t$.*

The above definition does now allow the construction of graphsets by composition. Note that none of the existing approaches combine intentional and extensional assignment of triples to graphsets (or named graphs). In [24] named graphs are defined intentionally through Sparql [25] views and do not support the explicit assignment of triples to named graphs, whereas in [5] a purely extensional definition is followed. The notion of graphsets introduced in this paper allows us to capture both the intentional and extensional aspects of RDF datasets that are useful to record and reason about provenance information in the presence of updates.

**Example 1.** *Consider the named graphs*

$$\begin{array}{lcl} g_n^{(1)} & = & \{(r, \mathsf{type}, A, n^{(1)}), (A, \mathsf{sc}, B, n^{(1)}), (C, \mathsf{sc}, D, n^{(1)})\} \\ g_n^{(2)} & = & \emptyset \\ g_n^{(3)} & = & \{(B, \mathsf{sc}, C, n^{(3)})\} \end{array}$$

*shown in Figure 2(a), and graphset $g_s$ associated with set $S = \{g_n^{(1)}, g_n^{(2)}, g_n^{(3)}\}$ of named graphs and identified by $i$. The set of quadruples directly associated with the graphset is $\{(r, \mathsf{type}, D, i)\}$. The set of quadruples jointly entailed by the named graphs in $N$ is empty since there does not exist a quadruple $t$ that is jointly entailed by quadruples belonging in all of the named graphs in $S$.*

*Suppose that named graph $g_n^{(2)}$ is now*

$$g_n^{(2)} = \{(A, \mathsf{sc}, B, n^{(2)})\})$$

*(see Figure 2(b)). Then, the set of jointly entailed quadruples for graphset $g_s$ becomes:*

$$\{(r, \mathsf{type}, C, i), (A, \mathsf{sc}, D, i)\}$$

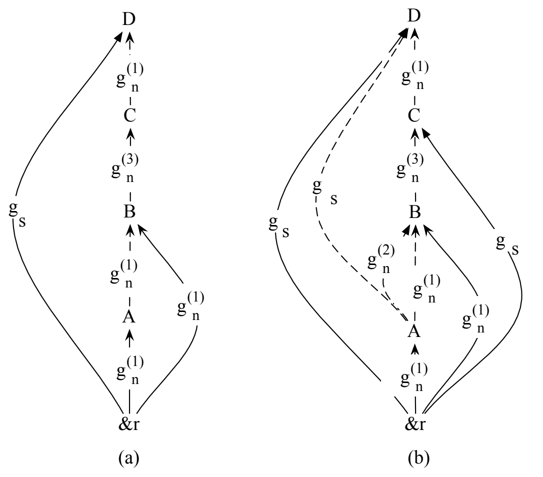


Figure 2: Graphset Example

In a similar manner as for *RDF named graphs* we define a consequence operator that abstracts a set of inference rules which compute the *closure* of a graphset. The inference rules given in Table 1 can be applied for graphsets as well where $n$ is the graphset identifier. We overload the notation here, and write $Cn(g_s)$ to refer to the closure of a graphset $g_s$. Also, we say that two graphsets are *identical*, denoted by $g_s^{(1)} = g_s^{(2)}$, iff they have the same identifier. It is straightforward to see that two graphsets associated with the same set of named graphs are identical via skolemization. Entailment for RDF/S graphsets is defined as follows: a graphset $g_s^{(1)}$ *entails* graphset $g_s^{(2)}$ iff the closure of $g_s^{(2)}$ is a subset of the closure of $g_s^{(1)}$ modulo the graphset identifiers.

Consider a graphset $g_s^{(1)}$ with an associated set $S$ of named graphs; we say that named graph $g_n$ is a *constituent of* $g_s^{(1)}$, denoted by $g_n \lessdot g_s^{(1)}$, iff $g_n \in S$.

Finally, note that graphsets *can be materialized* and subsequently treated as RDF named graphs by assigning them a *user defined URI*. The quadruples of materialized graphsets stem from both the intentional and extensional definition and they behave as yet another source of triples: the connection with their constituent named graphs is lost. This is useful for distributed SW applications requiring to exchange graphsets from one RDF/S processing system to another.

## 5 Reasoning for RDF Datasets

In this section we discuss *inference*, *validity* and *redundancy elimination* for RDF datasets. The validity constraints as well as redundancy elimination (in the style of [26]) are defined *independently* of the notion of graphsets introduced in this paper.

### 5.1 Inference

The RDFS inference mechanism can (and should) be extended to infer facts *across* graphsets. The rules in Table 2 span across multiple graphsets and are a straightforward extension of those in Table 1. The rules in Table 2 record the graphset that the implicit quadruple belongs to, based on those implying it.

| | | |
|---|---|---|
| Reflexivity of sc | $I_g^{(1)}$ : | $\dfrac{(C, \mathsf{type}, \mathsf{class}, i^{(1)})}{(C, \mathsf{sc}, C, i^{(1)})}$ |
| Transitivity of sc | $I_g^{(2)}$ : | $\dfrac{(C_1, \mathsf{sc}, C_2, i^{(1)}), (C_2, \mathsf{sc}, C_3, i^{(2)})}{(C_1, \mathsf{sc}, C_3, i^{(1,2)})}$ |
| Reflexivity of sp | $I_g^{(3)}$ : | $\dfrac{(P, \mathsf{type}, \mathsf{prop}, i^{(1)})}{(P, \mathsf{sp}, P, i^{(1)})}$ |
| Transitivity of sp | $I_g^{(4)}$ : | $\dfrac{(P_1, \mathsf{sp}, P_2, i^{(1)}), (P_2, \mathsf{sp}, P_3, i^{(2)})}{(P_1, \mathsf{sp}, P_3, i^{(1,2)})}$ |
| Transitivity of class instantiation | $I_g^{(5)}$ : | $\dfrac{(x, \mathsf{type}, C_1, i^{(1)}), (C_1, \mathsf{sc}, C_2, i^{(2)})}{(x, \mathsf{type}, C_2, i^{(1,2)})}$ |
| Transitivity of property instantiation | $I_g^{(6)}$ : | $\dfrac{(P_1, \mathsf{sp}, P_2, i^{(1)}), (x_1, P_1, x_2, i^{(2)})}{(x_1, P_2, x_2, i^{(1,2)})}$ |

Table 2: RDFS Inference Rules with Graphsets

In Table 2, $i^{(1)}$ and $i^{(2)}$ are graphset identifiers and we denote with $i^{(1,2)}$ the identifier of the graphset whose associated named graphs are the associated named graphs of $i^{(1)}$ and $i^{(2)}$. Take, for instance, rule $I_g^{(2)}$: if

$(A, \mathsf{sc}, B, i^{(1)})$ with $i^{(1)}$ the identifier for graphset $g_s^{(1)}$ and $(B, \mathsf{sc}, C, i^{(2)})$ with $i^{(2)}$ the identifier for graphset $g_s^{(2)}$, then quadruple $(A, \mathsf{sc}, C, i^{(1,2)})$ belongs to the graphset whose associated named graphs are those of $g_s^{(1)}, g_s^{(2)}$ (and has identifier $i^{(1,2)}$). Moreover, we overload the closure operator $Cn$ in order to capture the *closure* of an RDF Dataset, computed using the inference rules of Table 2.

## 5.2 Validity and Redundancy Elimination

The notion of validity has been described in various fragments of SW languages ([16, 26]), and is used to overrule certain triple combinations. In the context of graphsets, the validity constraints are applied (and defined) at the level of the RDF dataset, but the graphset-related part of the quadrable is not considered. The main validity requirement that we will use in this paper is the fact that a property instance's subject and object should be correctly classified under the domain and range of the property respectively; other constraints include the disjointness between class and property URIs and the acyclicity of [sc] and [sp]. For a full list of the related validity constraints, see [17]. Similarly, the detection and removal of redundancies is straightforward using the rules of Table 2.

In the sequel, we assume that queries and updates are performed upon valid and redundant-free RDF datasets. In effect, this means that invalidities and redundancies are detected (and removed) at update time rather than at query time. This choice was made because we believe that in real scale SW systems, query performance should prevail over update performance. Redundant-free RDF datasets were chosen because they offer a number of advantages in the case of transaction management for concurrent updates and queries.

## 6 Querying and Updating RDF Datasets

## 6.1 Querying RDF Datasets

In this section, we discuss the semantics of our query language, which is an extension of RQL [14]. We consider $\mathcal{V}, \mathcal{GV}$ to be two sets of variables for resources and graphsets respectively; $\mathcal{V}$, $\mathcal{GV}$, $\mathbf{U}$ and $\mathbf{L}$ are mutually disjoint sets. We rely on tableau queries to formalize the semantics of our query language: in our context, a query is of the form $(H, B, C)$ where $H$ (head) is a *q-pattern*, $B$ (body) is a conjunction of *q-patterns* and $C$ (constraints) is a conjunction of *atomic predicates*. A *q-pattern* is a quadruple from $(\mathbf{U} \cup \mathcal{V}) \times (\mathbf{U} \cup \mathcal{V}) \times (\mathbf{U} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathcal{GV})$, whereas each atomic predicate (from $C$) has the form:

1. $v \; op \; c$ for $v \in \mathcal{V}$, $op$ is one of $\{=, <, >, <=, >=\}$ and $c \in \mathbf{L} \cup \mathbf{U} \cup \mathcal{V}$
2. $v \; op' \; v'$ for $v, v' \in \mathcal{GV}$, $op' \in \{=, \lessdot\}$
3. $i = \{n_1, n_2, \ldots, n_k\}$ where $i \in \mathcal{GV}$ and $n_i \in \mathbf{N}$

According to the above definition, one can express constraints on resources *(1)*, on graphsets *(2)*, as well as to specify that a graphset considered in the query is associated with a given set of named graphs *(3)*. In this paper, we focus on atomic predicates involving resources which use the equality ($=$) operator. In addition, we require that all variables that appear in the head of the query ($H$) appear in the query's body ($B$). This restriction is imposed in order to have computationally desirable properties.

We denote variables with $?x$, $?y$, ... for resources and $?i_1$, $?i_2$, ... for graphset identifiers. To define the semantics of queries, we use the notion of *valuation* (mapping) in the same spirit as in [11] as follows: a *valuation* $\nu$ from $\mathcal{V} \cup \mathcal{GV}$ to $\mathbf{U} \cup \mathbf{L} \cup \mathbf{I}$ is a partial function $\nu : (\mathcal{V} \cup \mathcal{GV}) \rightarrow \mathbf{U} \cup \mathbf{L} \cup \mathbf{I}$. The domain of $\nu$ ($dom(\nu)$) is the subset of $\mathcal{V} \cup \mathcal{GV}$ where $\nu$ is defined. For $\nu$ a valuation, $?x$ a variable, $\nu(?x)$ denotes the resource, literal, or graphset to which $?x$ is mapped through $\nu$.

To define the semantics of a *q-pattern* we must define first the *semantics of property $p$ over an RDF Dataset $d$*, denoted by $[[p]]_d$. Given an RDF Dataset $d$, $[[p]]_d$ is defined for the properties $\mathsf{type}$, $\mathsf{sc}$, $\mathsf{sp}$ and $p$ as follows:

$$\begin{aligned}
[[\mathsf{type}]]_d &= \{(x,\ y,\ i) \mid d \vdash_{\{I_g^{(2)}, I_g^{(5)}\}} (x, \mathsf{type}, y, i)\} \\
[[\mathsf{sc}]]_d &= \{(x,\ y,\ i) \mid d \vdash_{\{I_g^{(1)}, I_g^{(2)}\}} (x, \mathsf{sc}, y, i)\} \\
[[\mathsf{sp}]]_d &= \{(x,\ y,\ i) \mid d \vdash_{\{I_g^{(3)}, I_g^{(4)}\}} (x, \mathsf{sp}, y, i)\} \\
[[p]]_d &= \{(x,\ y,\ i) \mid d \vdash_{\{I_g^{(4)}, I_g^{(6)}\}} (x, p, y, i)\}
\end{aligned}$$

We write $\langle p \rangle_d$ to denote the semantics of property $p$ when no inference rule is used. We can now define the semantics of a *q-pattern*. Consider an RDF dataset $d$ and $t$ = $(?X, exp, ?Y, ?i)$ a *q-pattern*, where $exp$ is one of $\mathsf{sc}$, $\mathsf{sp}$, $\mathsf{type}$, $\mathsf{domain}$, $\mathsf{range}$ or $p$. Then the evaluation of $t$ over $d$ is defined as follows:

$$\begin{aligned}
[[t]]_d \quad = \quad & \{\nu \mid dom(\nu) = \{?X, ?Y, ?i\} \; and \\
& (\nu(?X),\ \nu(?Y),\ \nu(?i)) \in [[exp]]_d\}.
\end{aligned}$$

In Table 3 we give the semantics of some *q-patterns* when URIs, literals and graphset identifiers are considered (in Table 3, $a$ and $b$ are constant URIs or literals and $i$ is a graphset identifier).

Finally, given a valuation $\nu$ we say that $\nu$ satisfies an atomic predicate $C$, denoted by $\nu \vdash C$, per the following conditions:

$$\begin{array}{lll}
\nu \vdash (?x = c) & iff & \nu(?x) = c, c \in \mathbf{U} \cup \mathbf{L}, \\
 & & \quad ?x \in dom(\nu) \\
\nu \vdash (?x = ?y) & iff & \nu(?x) = \nu(?y), \\
 & & \quad ?x, ?y \in dom(\nu) \\
\nu \vdash (?i = ?i') & iff & \nu(?i) = \nu(?i'), \\
 & & \quad ?i, ?i' \in dom(\nu) \\
\nu \vdash (?i \lessdot ?i') & iff & \mu(\nu(?i)) \subseteq \mu(\nu(?i')), \\
 & & \quad ?i, ?i' \in dom(\nu) \\
\nu \vdash (?i = \{n_1, \ldots n_k\}) & iff & \mathtt{sid}(\{n_1, \ldots, n_k\}) = \nu(?i), \\
 & & \quad ?i \in dom(\nu)
\end{array}$$

$$
\begin{array}{lcl}
[[(a, exp, ?y, ?i)]]_d & = & \{\nu \mid dom(\nu) = \{?y, ?i\} \; and \; (a,\ \nu(?y),\ \nu(?i)) \in [[exp]]_d\} \\
[[(?x, exp, a, ?i)]]_d & = & \{\nu \mid dom(\nu) = \{?x, ?i\} \; and \; (\nu(?x),\ a,\ \nu(?i)) \in [[exp]]_d\} \\
[[(?x, exp, ?y, i)]]_d & = & \{\nu \mid dom(\nu) = \{?x, ?y\} \; and \; (\nu(?x),\ \nu(?y),\ i) \in [[exp]]_d\} \\
[[(a, exp, b, ?i)]]_d & = & \{\nu \mid dom(\nu) = \{?i\} \; and \; (a,\ b,\ \nu(?i)) \in [[exp]]_d\} \\
[[(a, exp, b, i)]]_d & = & \{\nu \mid dom(\nu) = \emptyset \; and \; (a,\ b,\ i) \in [[exp]]_d\}
\end{array}
$$

Table 3: Semantics of *q-patterns*

where $\mu$ is a function that returns for a graphset identifier the set of identifiers of its associated named graphs and sid is the skolem function that computes the graphset identifier based on the graphset's constituents named graphs.

As in [18], the semantics of the conjunction of q-patterns is defined as follows:

$$[[P_1, P_2]]_d = [[P_1]]_d \bowtie [[P_2]]_d$$

where

$$[[P_1]]_d \bowtie [[P_2]]_d = \{\nu_1 \cup \nu_2 \mid \nu_1 \in [[P_1]]_d, \nu_2 \in [[P_2]]_d, \\ \nu_1, \nu_2 \ are\ compatible\ mappings\}$$

We say that two mappings are compatible if they map the same variable to the same value (i.e., for all $x \in dom(\nu_1) \cap dom(\nu_2)$, it holds that $\nu_1(x) = \nu_2(x)$). Similarly, we can define the semantics of optional patterns (like in Sparql [19]).

## 6.2 Updating RDF Datasets

RUL [15] extends the RQL language and is used for updating RDF graphs. RUL supports fine-grained updates at the (class and property) instance level, set-oriented updates with a deterministic semantics and takes benefit of the expressive power of RQL for restricting variables' range to nodes and arcs of RDF graphs. Here, we present an extension of RUL for supporting updates for RDF datasets focusing on *instance updates*.

The semantics of each RUL update is specified by its corresponding *effects* and *side-effects*. The effect of an insert or delete is defined over the graphset that is specified in the operation. The side-effects ensure that the resulting RDF dataset continues to be valid and non-redundant as discussed in [29]. Update semantics adhere to the principle of *minimal change* [7], per which a minimal number of insertions and deletions should be performed in order to restore a valid and non-redundant state of an RDF dataset. The effects and side-effects of insertions and deletions are determined by the kind of triple involved, i.e., whether it is a *class instance* or *property instance* insertion or deletion.

### 6.2.1 INSERT Operation

A primitive insert operation is of the form: insert$(s, p, o, i)$ where $s, p \in \mathbf{U}$, $o \in \mathbf{U} \cup \mathbf{L}$, $i \in \mathbf{I}$.

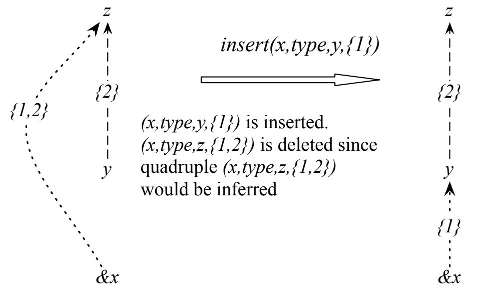

z
{2}
{1,2}
y
&x
insert(x,type,y,{1})
(x,type,y,{1}) is inserted.
(x,type,z,{1,2}) is deleted since
quadruple (x,type,z,{1,2})
would be inferred
z
{2}
y
{1}
&x

Figure 3: Class Instance Insertion

```
   Data: insert$(x, p, y, i)$, RDF dataset $d$
   Result: Updated RDF dataset $d$
1  if ($\exists\ (x,\ y,\ i) \in [[p]]_d$) then return $d$;
2  if ($p =$ type) then
3      if ($y \notin \mathbf{C}$) then
4          return $d$;
5      forall (($x,\ z,\ i'$) $\in \langle$type$\rangle_d$ s.t. $\exists(y,\ z,\ i'') \in$
       $[[$sc$]]_d$ and $i' = \{i, i''\}$) do
6          $d = d \setminus \{(x, \text{type}, z, i')\}$;
7      end
8      $d = d \cup \{(x, p, y, i)\}$;
9      return $d$;
10
11 else if ($\nexists\ (p,\ X,\ i) \in \langle$domain$\rangle_d$, $(x,\ X,\ j) \in$
   $[[$type$]]_d$ or $\nexists\ (p,\ Y,\ k) \in \langle$range$\rangle_d$, $(y,\ Y,\ l) \in$
   $[[$type$]]_d$) then
12     return $d$;
13
14 forall (($x,\ y,\ i'$) $\in \langle p\rangle_d$ s.t. $\exists(p,\ q,\ i'') \in \langle$sp$\rangle_d$ and
   $i' = \{i, i''\}$) do
15     $d = d \setminus \{(x, q, y, i')\}$;
16 end
17 $d = d \cup \{(x, p, y, i)\}$;
18 return $d$;
```

**Algorithm 1**: Class and Property Instance Insertion Algorithm

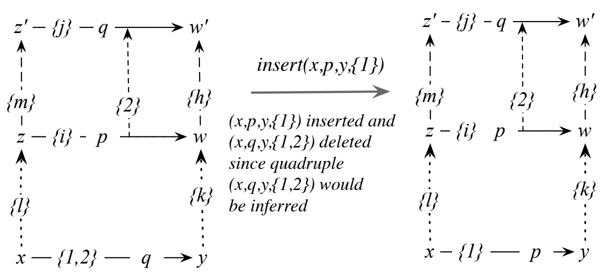


Figure 4: Property Instance Insertion

A formal description of the insertion of a triple to a graphset (i.e., a quadruple, say $(x, p, y, i)$) along with its side-effects can be found in Algorithm 1. At line `1` we examine if the quadruple already belongs to the semantics of property $p$. If not, and if the triple to be inserted is of the form $(x, \mathsf{type}, y, i)$ then, we ensure that $y$ is a class (lines `3`–`4`). If it is, then we remove all class instantiation quadruples from the RDF dataset which can be entailed through the quadruple to be inserted and the class subsumption relationships (lines `5`–`7`). Finally, the quadruple is inserted (line `8`). An example of a class instance insertion is shown in Figure 3.

If the quadruple to be inserted is of the form $(x, p, y, i)$ where $p \neq \mathsf{type}$ we must make sure that the domain and range validity constraints hold, i.e., that $x, y$ are instances of the domain and range of property $p$ respectively (lines `11`–`13`). If so, we remove all quadruples that will be redundant when the quadruple is inserted (lines `14`–`16`). Finally, the quadruple is added to the RDF dataset $d$ (line `17`). Figure 4 demonstrates an example of a property instance insertion.

### 6.2.2 DELETE Operation

A primitive delete operation is of the form: $\mathsf{delete}(s, p, o, i)$ where $s, p \in \mathbf{U}$, $o \in \mathbf{U} \cup \mathbf{L}$, $i \in \mathbf{I}$.

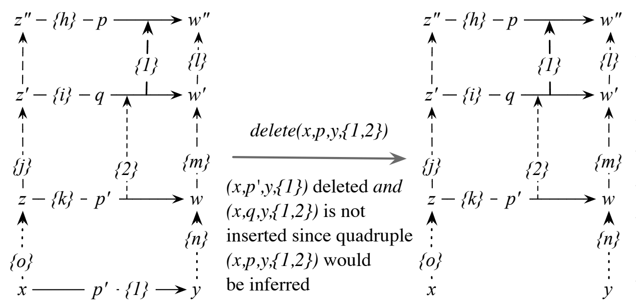


Figure 5: Property Instance Deletion

A formal description of the deletion of a quadruple

```
Data: delete($x, p, y, i$), RDF dataset $d$
Result: Updated RDF dataset $d$
1  if ($\nexists\ (x,\ y,\ i) \in [[p]]_d$) then return $d$;
2  if ($p = \mathsf{type}$) then
3      forall ($(x,\ y',\ i') \in [[\mathsf{type}]]_d, (y',\ y,\ i'') \in [[\mathsf{sc}]]_d$ s.t. $i = \{i', i''\}$) do
4          forall $(y',\ z,\ k) \in \langle \mathsf{sc} \rangle_d$ s.t. $y' != z$ do
5              if $\nexists (z,\ y,\ h) \in [[\mathsf{sc}]]_d$ then
                   $d = d \cup \{(x, \mathsf{type}, z, \{i', k\})\}$
6              end
7              $d = d \setminus \{(x, \mathsf{type}, y', i')\}$
8      end
9      forall $(x,\ o,\ h) \in \langle q \rangle_d$, s.t. $(q,\ c,\ i) \in \langle \mathsf{domain} \rangle_d$ do
10         if $\nexists\ (x,\ c,\ j) \in [[\mathsf{type}]]_d$ then
11             $d = d \setminus \{(x, q, o, h)\}$ ;
12             forall $q'$ s.t. $\exists\ (q,\ q',\ h'') \in [[\mathsf{sp}]]_d$ do
13                 if $\exists\ (x,\ e,\ k) \in [[\mathsf{type}]]_d$ s.t. $\exists\ (q,\ e,\ k') \in \langle \mathsf{domain} \rangle_d$ then
                       $d = d \cup \{(x, q', o, \{h, h''\})\}$
14             end
15
16     end
17     forall $(o,\ x,\ h) \in \langle q \rangle_d$, s.t. $(q,\ c,\ i) \in \langle \mathsf{range} \rangle_d$ do
18         if $\nexists\ (x,\ c,\ j) \in [[\mathsf{type}]]_d$ then
19             $d = d \setminus \{(o, q, x, h)\}$ ;
20             forall $q'$ s.t. $\exists\ (q,\ q',\ h'') \in [[\mathsf{sp}]]_d$ do
21                 if $\exists\ (x,\ e,\ k) \in [[\mathsf{type}]]_d$ s.t. $\exists\ (q,\ e,\ k') \in \langle \mathsf{range} \rangle_d$ then
                       $d = d \cup \{(o, q', x, \{h, h''\})\}$
22             end
23
24     end
25     return $d$;
26 else
27     forall $(x,\ y,\ i') \in [[p']]_d, (p',\ p,\ i'') \in [[\mathsf{sp}]]_d$ s.t. $i = \{i', i''\}$) do
28         forall $(p',\ q,\ k) \in \langle \mathsf{sp} \rangle_d$ s.t. $p' != q$ do if $\nexists (q,\ p,\ h) \in [[\mathsf{sp}]]_d$ then
29             $d = d \cup \{(x, q, y, \{i', k\})\}$
30
31     end
32     forall $(x,\ y,\ i') \in \langle p' \rangle_d, (y',\ y,\ i'') \in [[\mathsf{sp}]]_d$ s.t. $i = \{i', i''\}$ do $d = d \setminus \{(x, p', y, i')\}$ return $d$;
33
```
**Algorithm 2**: Class and Property Instance Deletion Algorithm

$(x, p, y, i)$ is given in Algorithm 2. As with the insertion of quadruples we differentiate between deletion of an instantiation link (i.e., a quadruple of the form $(x, \mathsf{type}, y, i)$ – lines 2–25) and a property edge (lines 26–33).

In the first case, we must remove all the quadruples that would cause the implication of the quadruple to be deleted (line 7 – see Figure 6 for an example), but, before that, we must make sure that the implications of the about-to-be-deleted quadruples which do not imply the deleted quadruple are retained (lines 4–6). In order to ensure that the RDF dataset is still valid after the updates, we must remove all properties originating from (or reaching) $x$ whose domain (or range) is a class that $x$ is no longer an instance of (lines 9–25). Figure 7 shows an example of class instance deletion that involves also property deletion.

In the case of deleting a property edge, a similar procedure is followed (see Figure 5): first, we explicitly add all quadruples that should be maintained (lines 26–31) and then remove the desired quadruple (lines 32–33).

## 7 Related Work

There are three kinds of provenance information [27]: *why provenance* (which refers to the source data that had some influence on the existence of the target data), *where provenance* (which refers to the locations in the source data from which the target data was extracted [4]) and *how provenance* (which refers to how source and target data are related and constrained via mappings [10]). To the best of our knowledge, this is the first work that examines the problem of *why provenance* for the RDF data model while considering RDFS inference and updates.

In [5], the use of named graphs as the means to store and manage explicit provenance information has also been considered, but there is no in depth discussion on how to manage provenance in the presence of queries and updates. There exist some works describing declarative languages for querying and updating RDF triples in the presence of named graphs (such as Sparql [19] and Sparql Update [25]), but these do not consider RDFS inference. On the other hand, two recent works that support RDFS inference [18, 20], do not support named graphs.

On the other side of the spectrum, a significant amount of work on the issue has been done for relational and tree-structured databases [2, 4, 10, 9]. In [2] authors discuss explicit provenance recording under copy-paste semantics where all operations are a sequence of delete-insert-copy-paste operations. In that work, new identifiers are introduced in the case in which the same object is deleted and then re-inserted, whereas in our case we are able to recognize the corresponding triple, and consequently preserve provenance information. In [10], fine-grained *where* and *how* provenance for relational databases is captured; however, updates are not considered in that work. Finally, in [9] authors consider a colored algebra to annotate columns and rows of relational tables at a coarse grained level which bares similarities to our named graph based approach.

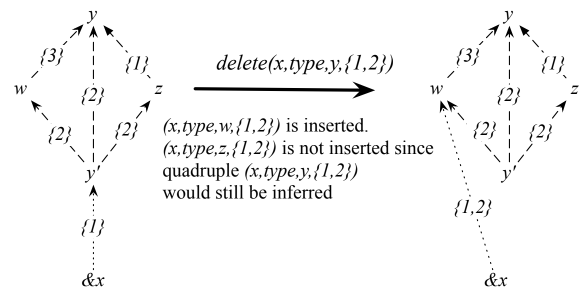

y
{3}
{1}
w
{2}
z
{2}
{2}
y'
{1}
&x
delete(x,type,y,{1,2})
(x,type,w,{1,2}) is inserted.
(x,type,z,{1,2}) is not inserted since quadruple (x,type,y,{1,2}) would still be inferred
y
{3}
{1}
w
{2}
z
{2}
{2}
y'
{1,2}
&x

Figure 6: Class Instance Deletion (1)

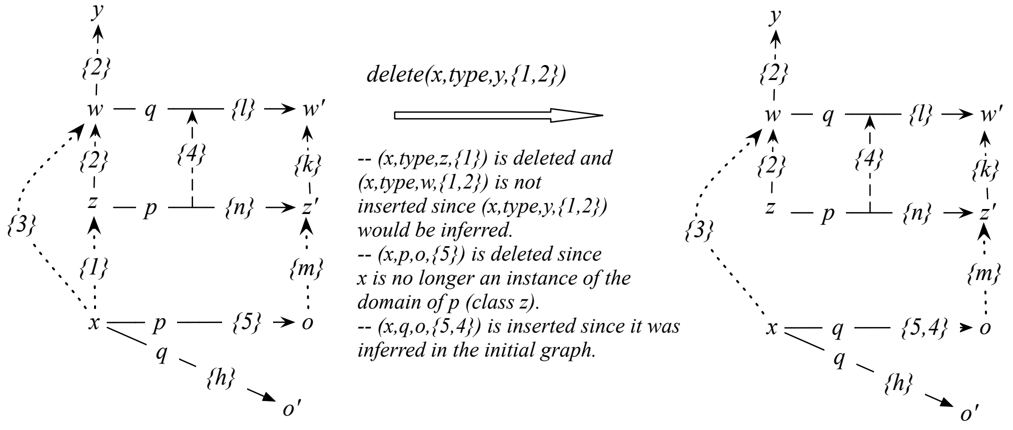
## 8 Conclusion

This paper addresses the problem of managing provenance information in RDF datasets. We follow the idea presented in [5, 28], where named graphs have been proposed in order to assign provenance information to a collection of RDF triples. One of the main arguments of our paper is that named graphs are not sufficient for most applications, because they don't allow the explicit assignment of "joint entailment" information to triples, a feature that is necessary in order to support updates without losing provenance information. For this purpose, we formalize the notion of graphsets as a generalization of named graphs; this is the first contribution of this paper. The interested reader can find a more detailed description in [17].

In order to be able to manage provenance information in RDF datasets, we extended existing query (RQL [14]) and update (RUL [15]) languages to support queries and updates of triples with provenance information (graphsets), taking into account the RDFS inference semantics. To our knowledge, this is the first effort to formally define the semantics of query and update languages that support both RDFS inference and provenance. These languages have been recently implemented and a demo can be found at [23].

## 9 Acknowledgments

This work was partially supported by the EU projects CASPAR (FP6-2005-IST-033572) and KP-Lab (FP6-2004-IST-4).



-- (x,type,z,{1}) is deleted and (x,type,w,{1,2}) is not inserted since (x,type,y,{1,2}) would be inferred.
-- (x,p,o,{5}) is deleted since x is no longer an instance of the domain of p (class z).
-- (x,q,o,{5,4}) is inserted since it was inferred in the initial graph.

Figure 7: Class Instance Deletion (2)

## References

[1] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. www.w3.org/TR/2004/REC-rdf-schema-20040210, 2004.

[2] P. Buneman, A. P. Chapman, and J. Cheney. Provenance Management in Curated Databases. In *SIGMOD*, 2006.

[3] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS*, 2008.

[4] P. Buneman, J. Cheney, and S. Vansummeren. On the Expressiveness of Implicit Provenance in Query and Update Languages. In *ICDT*, 2007.

[5] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, Provenance and Trust. In *WWW*, 2005.

[6] B. McBride F. Manola, E. Miller. RDF Primer. www.w3.org/TR/rdf-primer, February 2004.

[7] P. Gardenfors. Belief Revision: An Introduction. *Belief Revision*, (29):1–28, 1992.

[8] P. Gardenfors. The dynamics of belief systems: Foundations versus coherence theories. *Revue Internationale de Philosophie*, 44:24–46, 1992.

[9] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. In *ICDE*, 2006.

[10] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[11] C. Gutierrez, C. A. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *PODS*, 2004.

[12] P. Hayes. RDF Semantics. www.w3.org/TR/rdf-mt, February 2004. W3C Recommendation.

[13] The UMD Astronomy Information and Knowledge Group. Astonomy Ontology in OWL. archive.astro.umd.edu.

[14] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. Rql: a declarative query language for rdf. pages 592–603. ACM Press, 2002.

[15] M. Magiridou, S. Sahtouris, V. Christophides, and M. Koubarakis. RUL: A Declarative Update Language for RDF. In *ISWC*, 2005.

[16] S. Munoz, J. Perez, and C. Gutierrez. Minimal deductive systems for RDF. In *ESWC*, 2007.

[17] P. Pediaditis. Querying and Updating RDF/S Named Graphs. Master's thesis, Computer Science Department, University of Crete, 2008.

[18] J. Perez, M. Arenas, and C. Gutierrez. nSPARQL: A Navigational Language for RDF. In *ISWC*, 2008.

[19] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. www.w3.org/TR/rdf-sparql-query, January 2008.

[20] PSPARQL. psparql.inrialpes.fr.

[21] Gene Ontology. www.geneontology.org.

[22] UniProtRDF. dev.isb-sib.ch/projects/uniprot-rdf.

[23] RQL, RUL demo. athena.ics.forth.gr:3026/RULdemo/named_graph_demo/.

[24] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the Web. In *WWW*, 2008.

[25] A. Seaborne and G. Manjunath. SPARQL/Update: A language for updating RDF graphs. jena.hpl.hp.com/~afs/SPARQL-Update.html, April 2008.

[26] G. Serfiotis, I. Koffina, V. Christophides, and V. Tannen. Containment and Minimization of RDF/S Query Patterns. In *ISWC*, 2005.

[27] Wang-Chiew Tan. Provenance in databases: Past, current, and future. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2007.

[28] E. Watkins and D. Nicole. Named Graphs as a Mechanism for Reasoning About Provenance. In *Frontiers of WWW Research and Development - APWeb*, 2006.

[29] D. Zeginis, Y. Tzitzikas, and V. Christophides. On the foundations of computing deltas between rdf models. In *ISWC/ASWC*, 2007.