Querying for Meta Knowledge

Bernhard Schueler
Sergej Sizov
Steffen Staab

Nr. 8/2008

Arbeitsberichte aus dem
Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Kontaktdaten der Verfasser

Bernhard Schueler, Sergej Sizov, Steffen Staab
Institut für Informatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: bernie@uni-koblenz.de, sizov@uni-koblenz.de, staab@uni-koblenz.de

Querying for Meta Knowledge

Bernhard Schueler, Sergej Sizov, Steffen Staab
ISWeb — Information Systems and Semantic Web
University of Koblenz-Landau, Germany
{bernie,sizov,staab}@uni-koblenz.de

ABSTRACT

The Semantic Web is based on accessing and reusing RDF data from many different sources, which one may assign different levels of authority and credibility. Existing Semantic Web query languages, like SPARQL, have targeted the retrieval, combination and reuse of facts, but have so far ignored all aspects of meta knowledge, such as origins, authorship, recency or certainty of data, to name but a few.

In this paper, we present an original, generic, formalized and implemented approach for managing many dimensions of meta knowledge, like source, authorship, certainty and others. The approach re-uses existing RDF modeling possibilities in order to represent meta knowledge. Then, it extends SPARQL query processing in such a way that given a SPARQL query for data, one may request meta knowledge without modifying the query proper. Thus, our approach achieves highly flexible and automatically coordinated querying for data and meta knowledge, while completely separating the two areas of concern.

1. INTRODUCTION

Integrating and re-using Semantic Web data becomes more and more fruitful and worthwhile in order to answer questions and deliver results. Typically, engines like Swoogle provide points of access for RDF data, crawlers may fetch relevant RDF data, and query languages like SPARQL with their corresponding query engines allow for selecting and re-using data in the appropriate format. With the arrival of more and more data in the Semantic Web and more sophisticated processing through query and reasoning engines, one now, however, encounters challenging questions linked to meta knowledge about the data like:

- Where is this data from?
- Who provided the data?
- When was this data provided?
- Was the provider certain about the truth of this data?
- Was the data believed by others, too?

For instance, when querying the Semantic Web with the help of SPARQL for the affiliation of a person named of “James Hendler”, one finds (at least) two answers, i.e. “University of Maryland” and “Rensselaer Polytechnic Institute”. Without further indication as to where, by whom, when, etc. such information was given, it is impossible to decide which of the two affiliations is still valid.

The problem might be remedied in several ways. First, an ideosyncratic solution by the search engine, such as returning the corresponding RDF files or links to sources of knowledge extraction (say `http://www.cs.umd.edu/survey.pdf` and `http://www.rpi.edu/report.doc`), might help in this special case. However, an ideosyncratic solution may not be appropriate in a second case in which the ‘when’ was more relevant than the ‘where’ or in a third case where such a piece of information had to be aggregated from several resources. Second, the person or system requesting the meta knowledge might manually extend the SPARQL query formalizing the request for the affiliation in order to return the where, the who and the when. Such a modification will, however, be very tedious, as it will include a number of additional optional statements, and expressing it manually will be error prone. Also, it will not help in delivering meta knowledge that arises from joining several statements, e.g. meta knowledge about uncertainty that was based on several meta knowledge statements with different values of uncertainty.

Therefore, querying Semantic Web data requires a principled, generic approach to the treatment of meta knowledge that is able to adapt to many dimensions of meta knowledge and that is open to accommodate to new dimensions when the need arises. Such a principled, original framework is given in this paper. We start to explain our approach with a discussion of important design choices in section 2. We model meta knowledge in existing RDF structures by embedding a slightly more expressive language, which we call RDF$^+$, into RDF . We define the abstract syntax of RDF$^+$, its semantics and its embedding in RDF in Section 4. In Section 5, we extend the SPARQL syntax and semantics to work on data and meta knowledge of RDF$^+$. The extension allows the user to extend a given conventional SPARQL query by a keyword for meta knowledge triggering the construction of meta knowledge by the query processor. Section 6 summarizes the overall use and processing of SPARQL queries with meta knowledge. Section 7 reports on initial graceful results for meta knowledge processing from a theoretic point of view and Section 8 provides pointers to the prototype implementation of the system.

2. SCENARIO

In our sample application scenario, we assume that the user utilizes knowledge which has been initially extracted from Web pages of Computer Science departments and stored in form of RDF triples in his personal “active space" [17], backed by a local RDF repository. Example 2.1 shows the relevant facts that may have been obtained from departments of different universities. For better readability, we use for our examples in this paper the RDF triple language TriG [1] with Named Graphs [2] in a simplified form that abstracts from default namespaces.

Example 2.1. *Extracted Knowledge and SPARQL query*

```
G1 { JamesHendler researchTopic SemanticWeb .
     JamesHendler affiliatedWith RensselaerPI }

G2 { JamesHendler researchTopic Robotics .
     JamesHendler affiliatedWith UnivMaryland .
     RudiStuder researchTopic SemanticWeb .
     RudiStuder affiliatedWith UnivKarlsruhe }
```

The extracted knowledge comes from different sources, at different timepoints, and with different degrees of extraction confidence. This information is also captured and stored into the same RDF repository as shown in example 2.2, using the notion of Named RDF Graphs [2, 5].

Example 2.2. *Associated meta knowledge*

```
G3 { G1 mk:source <www.rpi.edu/report.doc> .
     G1 mk:certainty "0.9" .
     G1 mk:timestamp "5/5/2007" }

G4 { G2 mk:source <www.cs.umd.edu/survey.pdf> .
     G2 mk:certainty "0.6" .
     G2 mk:timestamp "6/6/2001" }
```

In our scenario, the sample user aims to explore the knowledge and meta knowledge using the RDF query language SPARQL. We assume that he aims to find experts in the domain of Semantic Web and their affiliations. The corresponding SPARQL query is shown in example 2.3. In addition, the user wants to exploit meta knowledge from example 2.2 for obtaining results with best certainty and for analyzing contradictive answers (e.g. different affiliations for the same person “James Hendler” in example 2.1).

Example 2.3. *Extracted Knowledge and SPARQL query*

```
CONSTRUCT {?x worksAt ?z}
FROM NAMED G1
FROM NAMED G2
WHERE { GRAPH ?g {?x affiliatedWith ?z .
                  ?x researchTopic SemanticWeb}
}
```

3. DESIGN CHOICES

This section summarizes and shortly motivates the design choices for our meta knowledge framework.

Reification. Establishing relationships between knowledge and meta knowledge requires appropriate reification mechanisms for supporting statements about statements. Our general objective is to execute queries on original data (i.e. without meta knowledge) directly, without complex transformations. For compliance with existing applications that access the repository in a common way (e.g. using SPARQL queries), we do not modify existing user data. This requirement does not allow us to use mechanisms like RDF reification, which decompose existing triples and fully change the representational model. In our framework described in section 4, we adopt the notion of Named RDF Graphs for meta knowledge representation [2, 5].

Storage mechanisms. Following the overall philosophy of RDF, we do not separate meta knowledge from “normal” user knowledge in the repository. Following this paradigm, a user or developer has unlimited access to all contents of the triple store and can manipulate meta knowledge directly. In other words, the user can directly access meta knowledge (e.g. using suitable SPARQL queries). Beyond explicitly designed queries for meta knowledge access, in Section 5 we describe the extension of SPARQL that allows us to access meta knowledge about the result set automatically without user intervention.

Dimensions of Meta Knowledge. An important point for the application design is the definition of relevant meta knowledge properties and their suitable interpretation for arbitrary complex query patterns. In general, these properties are application dependent and must be carefully chosen by the system administrator. In our scenario (sections 2 and 6) we discuss common and widely used properties, such as timestamp, source, and (un)certainty, and show ways of defining and utilizing them in our framework.

Syntax extensions. Seamlessly integrated access to meta knowledge requires corresponding extensions of existing querying mechanisms. These can be realized at different levels, for instance at the level of query languages (e.g. SPARQL) or at the level of application-specific interfaces (e.g. Sesame API). In Section 5 we describe our SPARQL extension for constructing query results with associated meta knowledge. It is system-independent and not related to some particular implementation of the RDF repository. Furthermore, it fully supports the existing SPARQL syntax and semantics. Compliance with existing established standards makes the integration with existing applications and interfaces substantially easier.

4. SYNTAX AND SEMANTICS FOR RDF WITH META KNOWLEDGE

In the course of representing and reasoning with meta knowledge we embed a language with meta knowledge reasoning, i.e. RDF$^+$, in a language without such specific facilities, i.e. in RDF. This embedding implies that we may consider an RDF snippet in its literal sense *and* we may possibly interpret it as making a meta knowledge statement. Embedding meta knowledge in RDF is not the most expressive means to deal with all needs of meta knowledge processing, but it retains upward compatibility with existing usage of the language and corresponding tools and methods, which is a major concern for Semantic Web approaches.

Though we denote meta knowledge in RDF, we must distinguish the notation of RDF with only *implicit* notation of meta knowledge, but no semantic consequences specifically due to this meta knowledge, from a formally extended model of RDF with *explicit* notation of meta knowledge. The following definition of RDF$^+$ helps us to draw this line very clearly and concisely. The abstract syntax for this embedded language, RDF$^+$, is given in Section 4.1 and its semantics in Section 4.2. Eventually in this section, we show how to embed RDF$^+$ in RDF with named graphs.

4.1 An Abstract Syntax for RDF$^+$

The abstract syntax of RDF$^+$ is based on the same building blocks as RDF:

- U are Uniform Resource Identifiers (URIs).
- L are all RDF literals.
- $G \subseteq U$ is the set of graph names.
- $P \subseteq U$ is the set of properties.

In addition, we must be able to refer to statements directly without use of reification. For this purpose, we introduce statement identifiers:

- Θ is a set of statement identifiers, which is disjoint from U and L.

Now, we may define RDF$^+$ literal statements that are placed in named graphs and have, in addition to RDF, a globally unique statement identity.

Definition 4.1 (RDF$^+$ Literal Statements).
The set of all RDF$^+$ literal statements, $\mathfrak{S}$, is defined as quintuples by:
$\mathfrak{S} := \{(g, s, p, o, \theta) \mid g \in G, s \in U, p \in P, o \in U \cup L, \theta \in \Theta\}$.

Thereby, θ and (g, s, p, o) are keys such that there exists a bijection f_1 with $f_1(g, s, p, o) = \theta \wedge f_4(\theta) := f^{-1}(\theta) = (g, s, p, o)$. Moreover, we define the overloaded function f_5 to return the complete quintuple given either θ or (g, s, p, o), i.e. $f_5(\theta) := (g, s, p, o, \theta) =: f_5(g, s, p, o)$, when $f_1(g, s, p, o) = \theta$.

The reader may note that we assume that f_1 is fixed and given before any statement is defined. Furthermore, this definition of literal statements and the rest of this paper abstracts from RDF blank nodes in order to keep the formalization more concise. However, we do not see any conceptual problem in extending our treatments to blank notes, too.

The two statements of Graph G1 of Example 2.1 may now be represented in RDF$^+$ in the following way.

Example 4.1.

$\mathfrak{S} \supseteq K \supseteq \{$
(G1, JamesHendler, researchTopic, SemanticWeb, θ_1),
(G1, JamesHendler, affiliatedWith, RensselaerPI, θ_2) }

Thereby, the exact form of statement identifiers in Θ is up to the implementation, as they are only used for internal processing.

Having represented the literal interpretation of RDF statements in RDF$^+$, we may now address the representation of selected RDF statements as RDF$^+$ meta knowledge. This is done using a structure of RDF$^+$ meta knowledge statements, $\mathfrak{M}$, that is separate from the set of RDF$^+$ literal statements:

Definition 4.2 (RDF$^+$ Meta Knowledge Statements).
Let $\Pi \subseteq P$ be the set of meta knowledge properties. Let Ω_π, with $\pi \in \Pi$, be sets providing possible value ranges for the meta knowledge properties $\pi \in \Pi$.

Then, the set of all RDF$^+$ meta knowledge statements, $\mathfrak{M}$, is defined by: $\mathfrak{M} := \{(\theta, \pi, \omega) \mid \theta \in \Theta, \pi \in \Pi, \omega \in \Omega_\pi)\}$.

The following example partially demonstrates the target representation of the first two meta knowledge statements of graph *G3* from Example 2.2.

Example 4.2.

$\mathfrak{M} \supseteq M \supseteq \{$
(θ_1, mk:source, {<www.rpi.edu/report.doc>}),
(θ_1, mk:certainty, 0.9)}

Together we may now define a RDF$^+$ theory.

Definition 4.3 (RDF$^+$ Theory).
A RDF$^+$ theory of literal statements and associated meta knowledge statements is a pair (K, M) referring to a set of literal statements $K \subseteq \mathfrak{S}$ and a set of meta knowledge statements $M \subseteq \mathfrak{M}$.

A (partial) example for such a theory is given by the pair (K, M) with definitions for K and M as given in examples 4.1 and 4.2, respectively.

4.2 A Semantics for RDF$^+$

We now have an abstract syntax for representing RDF triples like *JamesHendler researchTopic SemanticWeb* as part of *G1* and meta knowledge statements like *the source of the statement that James Hendler's research topic is Semantic Web is found in the document <www.rpi.edu/report.doc>*. However, such an abstract syntax may remain remarkably ambiguous if it cannot be linked to a formal semantics. Assume two meta knowledge statements:
(θ_1, mk:source, {<www.rpi.edu/draftReport.doc>}) and
(θ_1, mk:source,{<www.rpi.edu/finalReport.doc>})
for the same literal statement identified by θ_1, the question may arise whether this means a disjunction, i.e. one of the two documents has provided the fact, or a conjunction, i.e. both documents

have provided the fact, or a collective reading, i.e. the two documents together gave rise to the fact, or whether this situation constitutes invalid meta knowledge.

In order to prevent such ambiguities we introduce a generic semantic framework for meta knowledge in RDF$^+$. However, the framework must also be able to reproduce the literal interpretations found in RDF. For the latter purpose, we first define a 'standard' model for a RDF$^+$ theory.

Definition 4.4 (Standard Interpretation and Model). *A standard interpretation $I_s : \mathfrak{S} \rightarrow \{\top, \bot\}$ for a structure (K, M) assigns truth values to all statements*[1] *in K.*

A standard interpretation is a standard model if and only if it makes all statements in K become true.

For instance, any standard model I_s for (K, M) in example 4.1 would include *(G1, JamesHendler, researchTopic, SemanticWeb, θ_1)* in its set of literal statements evaluating to $\top$.

In order to address the level of meta knowledge we foresee an additional model layer that provides a different interpretation to each meta knowledge property.

Definition 4.5 (Π-Interpretation and Model). *A Π-interpretation $I_\pi : \mathfrak{S} \rightharpoonup \Omega_\pi$ for a property $\pi \in \Pi$ is a partial function mapping statements into the allowed value range of π.*

A Π-interpretation I_π is a Π-model for (K, M) if and only if for all meta knowledge statements $(\theta, \pi, \omega) \in M$ where $f_1(\theta) = (g, s, p, o)$ the value of the interpretation coincides with ω, i.e. $I_\pi((g, s, p, o, \theta)) = \omega$.

As an example, consider the certainty interpretation $I_{certainty}$ of the literal statement *(G1, JamesHendler, researchTopic, SemanticWeb, θ_1)* from Examples 4.1 and 4.2. A model I would map this literal statement using $I_{certainty}$ onto *0.9*.

The literal and the meta knowledge interpretations may now be combined to define what an overall, unambiguous model is:

Definition 4.6 (meta knowledge Interpretation and Model). *A meta knowledge interpretation $\mathfrak{I}$ is a set including a standard interpretation I_s and the Π-interpretations I_π for all meta knowledge properties $\pi \in \Pi$.*

A meta knowledge interpretation $\mathfrak{I}$ is a model for a theory (K, M) if and only if all its interpretations $I \in \mathfrak{I}$ are a standard model or Π-models for (K, M).

4.3 Mapping between RDF and RDF$^+$

The mapping between RDF and RDF$^+$ needs to be defined in two directions. First, one must be able to map from RDF as given in the examples from Section 2 to RDF$^+$. Second, one must be able to map from RDF$^+$ to RDF. Because RDF$^+$ is more fine-grained than RDF the first direction will be easy. For the second a compromise on the granularity of the representation has to be made.

[1]Note that because f_1 is fixed there are no two tuples $(g, s, p, o, \theta_1), (g, s, p, o, \theta_2)$, where $\theta_1 \neq \theta_2$. This implies that the standard interpretation is independent of the identifiers θ_1, θ_2.

4.3.1 From RDF to RDF$^+$

The examples of Section 2 reify groups of statements, i.e. the ones found in *G1* and *G2*, in order to associate meta knowledge, such as given in *G3* and *G4*. In order to allow for an interpretation of the meta knowledge as defined in the preceding section, we map RDF into RDF$^+$. For all RDF statements, including statements in graphs *G1* and *G2* of Example 2.1, the mapping performed is close to an identity mapping. One only needs to add statement identifiers. The result for *G1* in RDF$^+$ is:

Example 4.3.

```
K ⊇ { (G1, JamesHendler, researchTopic, SemanticWeb, θ1),
(G1, JamesHendler, affiliatedWith, RensselaerPI, θ2) },
with
θ1 := f1(G1, JamesHendler, researchTopic, SemanticWeb) and
θ2 := f1(G1, JamesHendler, affiliatedWith, RensselaerPI)
```

The same mapping – close to the identity mapping – is performed for meta knowledge statements like statements of graph *G3*, resulting in their representation as literal statements:

Example 4.4.

```
K ⊇ {
(G3, G1, mk:source, <www.rpi.edu/report.doc>, θ3),
(G3, G1, mk:certainty, "0.9", θ4), ...}
```

Note that this step is necessary in order to achieve upward and – limited – downward compatibility between RDF$^+$ and RDF.

The interpretation of statements, like the ones found in *G3*, also require an interpretation as meta knowledge. This is achieved by mapping RDF statements with designated properties from Π like *mk:source* and *mk:certainty* to the additional meta knowledge layer:

Example 4.5.

```
M ⊇ {
(θ1, mk:certainty, "0.9"),
(θ1, mk:source, {<www.rpi.edu/report.doc>}), ...}
```

The mapping of predicates of these meta knowledge statements from RDF to RDF$^+$ is obvious, they are mapped to itself. Objects are mapped to the corresponding elements of the value ranges Ω_π. For the subjects, however, there arise modeling choices. For instance, if *mk:certainty* were interpreted using probability theory, one might assign a distributive or a collective reading. In the distributive reading, each fact in *G1* receives the probability value of 0.9 and, eventually, the distributive reading will assign a joint probability of close to 0 for a large number of *n* stochastically independent facts, i.e. the joint probability 0.9^n. In the collective reading, the collection of facts in *G1* as a whole will receive the probability value 0.9. Therefore, the collective reading will assign an individual certainty close to 1 for each individual fact, when the number of facts is high and each fact is independent from the others, i.e. the individual probability would be $\sqrt[n]{0.9}$. *A priori*, none of the two (and more) modeling choices is better than the other, but they constitute different modeling targets.

The mapping from RDF to RDF$^+$ for the *distributive reading* of a meta property π is easy to achieve.

Definition 4.7 (Distributive Embedding).
Given an RDF statement "G {S P O}" and an RDF meta knowledge statement "H {G π ω}", a distributive embedding of RDF$^+$ in RDF adds the meta knowledge statement

$$\{(\theta, \pi, \omega) \mid \theta = f_1(G, s, p, o) \wedge f_5(\theta) \in K\}$$

to M.

This means that such a meta knowledge statement is applied individually to all statements in the graph to which it refers in RDF, as indicated in the example above. For certain π there might be several RDF meta knowledge statements *H {G π ω_i}* which attach different values ω_i to a graph G via a single meta knowledge property π. In that case a set-valued range Ω_π has to be used in order to be consistent with Definition 4.5.

4.3.2 From RDF$^+$ to RDF

The serialization of RDF$^+$ data in the knowledge base K is straightforward. Each quintuple (g, s, p, o, θ) is realized as a corresponding triple in a named graph and the tuple identifier θ is discarded.

Example 4.6.

```
(G5, JamesHendler, researchTopic, SemanticWeb, θ)
                    is mapped to
G5 {JamesHendler researchTopic SemanticWeb }
```

For meta knowledge statements the situation is more challenging, because literal statements with different statement identifiers may belong to only one named graph. Their corresponding meta knowledge statements may differ, but the realization of the meta knowledge statements in RDF does not allow for retaining these fine-grained distinctions – unless one chooses to change the modeling approach drastically, e.g. by assigning each literal statement to a named graph of its own, which seems undesirable (cf. discussion in Section 3).

We have preferred to pursue a more conventional modeling strategy for RDF with named graphs. Therefore, we weaken the association between meta knowledge statements and their corresponding literal statements when mapping to RDF. I.e. we group sets of meta knowledge property values into one complex value.

Definition 4.8 (Generating Grouped meta knowledge).
Given an RDF$^+$ theory (K, M), RDF meta knowledge is generated by grouping RDF$^+$ meta knowledge statements as follows:

Add the triple $(g\ \pi\ \omega')$ to the RDF graph $g' := hashGraph(g)$ for each

$$\omega' := \omega_1 \vee_\pi \ldots \vee_\pi \omega_n\,,$$

where $(\theta, \pi, \omega_i) \in M \wedge (g, S, P, O, \theta) \in K$. Further, hashGraph *is a function mapping existing graph names onto graph names suitable for associating meta knowledge and $\vee_\pi$ is an operation defined on Ω_π.*

If ω' is set-valued then a set of triples is added to g' in order to represent ω'. The suitability of *hashGraph* may be application specific. A general strategy may map graph names g to graph names prefixed by <http://metaknowledge.semanticweb.org> in a deterministic manner. Operations on meta knowledge properties are discussed in section 5.2.

In the following example the grouping of meta knowledge values is illustrated.

Example 4.7.

```
K:={
(G5, JamesHendler, researchTopic, SemanticWeb, θ1),
(G5, JamesHendler, affiliatedWith, UnivMaryland, θ2) },
M:={
(θ1, mk:source, {<www.rpi.edu/report.doc>}),
(θ2, mk:source, {<www.cs.umd.edu/survey.pdf>}) }
                    is mapped to
G5 { JamesHendler researchTopic SemanticWeb .
     JamesHendler affiliatedWith UnivMaryland }
G6 { G5 mk:source <www.rpi.edu/report.doc>,
                              <www.cs.umd.edu/survey.pdf>.}
```

In Example 4.7, the resulting grouped value is the set consisting of the two documents <*report.doc*> and <*survey.pdf*> which is represented by two triples. For specific meta knowledge properties, an additional function may be necessary to provide a mechanism for representing grouped values in an appropriate RDF data structure.

5. SPARQL FOR RDF AND META KNOWLEDGE

In this section we first introduce a small extension to standard SPARQL syntax [16] and then define how SPARQL can be applied to an RDF$^+$ knowledge base. The objective of our considerations is the derivation of meta knowledge about query results.

5.1 SPARQL Syntax Revisited

When using SPARQL to query RDF$^+$ we propose only two modifications to obtain meta knowledge. First, we introduce one additional expression "WITH META *MetaList*". This expression includes the named graphs specified in *MetaList* for treatment as meta knowledge. This statement is optional. When it is present the SPARQL processor may digest the RDF$^+$ meta knowledge statements derivable from the RDF named graphs appearing in the *MetaList*. The SPARQL processor will then use this meta knowledge to compute and output all the meta knowledge statements deriveable by successful matches of RDF$^+$ literal statements with the WHERE pattern.

In order to determine which literal statements should be considered we introduce a second modification. We do not process FROM expressions with our meta knowledge framework, but only FROM NAMED. The reason is that FROM *g* expressions replicate all RDF triples of *g* into the default triple space of the query. Thereby, they remove the links between the RDF statements of *g* and possible meta knowledge. Hence, FROM expressions are not relevant for our treatment of meta knowledge, but of course they may still be processed using the standard SPARQL semantics.

Thus, SPARQL queries on RDF$^+$ have one of the two following overall forms:

Definition 5.1 (SPARQL SELECT Query).
The structure of a SPARQL SELECT query has the following form:

```
SELECT SelectExpression
(WITH META MetaList)?
(FROM NAMED GraphName)+
WHERE P
```

Definition 5.2 (SPARQL CONSTRUCT Query).
The structure of a SPARQL CONSTRUCT query has the following form:

```
CONSTRUCT ConstructExpression
(WITH META MetaList)?
(FROM NAMED GraphName)+
WHERE P
```

In these definitions, P refers to a graph pattern that explains how RDF$^+$ literal statements from named graphs specified using FROM NAMED statements are matched. Matches bind variables that are used for providing results according to the *SelectExpression* or the *ConstructExpression*.

5.2 SPARQL Semantics Revisited

In this subsection we define the semantics of SPARQL queries evaluated on an RDF$^+$ theory. For our definitions we use two building blocks: algebraic semantics of SPARQL [12, 14] and the *how-provenance* calculated via annotated relations (cf. [8]).

The algebraic semantics of SPARQL queries are given based on set-theoretic operations for sets of variable assignments (cf. [12, 14]). Thereby, a variable assignment is a partial function $\mu : V \to U \cup L$, where V is the set of variables given in a SPARQL query. A set of variable assignments can be represented by a relation ϕ over the domain $(U \cup L)^{|V|}$, where the variables V are the attributes and assignments are the tuples of this relation. Such a set of assignments may be assigned information about the so called *how-provenance* [8], i.e. the assignments may be annotated with formulae describing the individual derivation tree used to assign the variables. The how-provenance annotation may be represented by a function $\Phi : (U \cup L)^{|V|} \to \mathfrak{F}$, where $(U \cup L)^{|V|}$ is the set of all tuples of the length $|V|$ over the domain $U \cup L$ and $\mathfrak{F}$ is the set of formulae annotating variable assignments. The set of formulae $\mathfrak{F}$ is given by all Boolean formulas constructed over the set of literal statements $\mathfrak{S}$ and including a bottom element $\perp$ and a top element $\top$. The formulae constitute an algebra $(\mathfrak{F}, \wedge, \vee, \neg, \perp, \top)$. The special element $\perp$ is used as annotation of variable assignments which are not in the relation ϕ. The special element $\top$ may be omitted, but it allows for simplification of complex formulas.

Assume the following SPARQL query to be evaluated on the RDF$^+$ knowledge base K:

Example 5.1.

```
SELECT ?g ?x ?y
FROM NAMED G1
FROM NAMED G2
WHERE {
    GRAPH ?g {?x researchTopic ?y}
}
```

Example 5.2.

```
K = {
(G1, JamesHendler,researchTopic,SemanticWeb, θ1),
(G1, JamesHendler,affiliatedWith,RensselaerPI, θ2),
(G2, JamesHendler,researchTopic,Robotics, θ3),
(G2, JamesHendler,affiliatedWith,UnivMaryland, θ4),
(G2, RudiStuder,researchTopic,SemanticWeb θ5),
(G2, RudiStuder,affiliatedWith,UnivKarlsruhe θ6) }
```

For the query of example 5.1, we may find the following variable assignments using standard SPARQL processing and we may indicate, which atomic formulae, i.e. RDF$^+$ quintuples in this simple example, led to these variable assignments. This indication is given by the statement identifiers representing their statements.

Example 5.3.

$\Phi =$

?g	?x	?y	$\mathfrak{F}$
G1	JamesHendler	SemanticWeb	θ_1
G2	JamesHendler	Robotics	θ_3
G2	RudiStuder	SemanticWeb	θ_5

This simple example of how a set of variable bindings has been produced is generalized to SPARQL queries of arbitrary complexity by a recursive definition of simultaneous query evaluation and computation of the annotations. The first step in evaluating a graph pattern is to find matches for the triple pattern contained in the query. Because the RDF$^+$ knowledge base K consists of quintuples, we need to adapt the SPARQL evaluation procedures. The statement identifiers do not need to be matched, as they depend functionally on graph name, subject, predicate and object. Therefore, we consider matching of quadruple patterns $(\gamma, \alpha, \beta, \delta)$. As a simplification of our formalization we assume that the keyword GRAPH together with a URI or a graph variable is used in any given SPARQL query. If it is not used, we may expand a given SPARQL query to include it.

Definition 5.3 (Basic Quadruple Pattern Matching).
Let K be a knowledge base of RDF$^+$ literal statements and μ be a variable assignment.

The evaluation of the SPARQL query ”GRAPH γ $\{\alpha\,\beta\,\delta\}$” *over K, denoted by* $[\![\text{GRAPH } \gamma\ \{\alpha\,\beta\,\delta\}]\!]_K$ *is defined by the annotated relation* Φ, $\mathrm{dom}(\Phi) = \{\mu \,|\, \mathrm{dom}(\mu) = \mathrm{vars}(\text{GRAPH } \gamma\ \{\alpha\,\beta\,\delta\})\}$,

$$\Phi(\mu) = \begin{cases} \theta & \textit{if } \mathrm{r}(\mu, (\gamma, \alpha, \beta, \delta)) = (g, s, p, o) \wedge \\ & (g, s, p, o, \theta) \in K \wedge f_1(g, s, p, o) = \theta, \\ \perp & \textit{else} \end{cases}$$

where $\mathrm{vars}(P)$ *denotes the variables contained in a pattern P and* $\mathrm{r}(\mu, (\gamma, \alpha, \beta, \delta))$ *is the quadruple obtained by replacing the variables in* $(\gamma, \alpha, \beta, \delta)$ *according to μ.*

An example for this definition is given by evaluating the query from Example 5.1 on the dataset of Example 5.2 delivering the result as indicated in example 5.3.

Basic quadruple pattern matching is not directly applicable, if an expression "GRAPH γ" appears outside a complex triple pattern. In such a case, we first need to distribute the expression "GRAPH γ" appropriately to atomic triple patterns in order to prescribe atomic SPARQL expressions accessible by basic quadruple pattern matching. Because named graphs cannot be nested, this distribution is always possible and unambiguous. In the following we use the function quads(P) to denote the query resulting from this transformation. In example 5.4 this transformation is demonstrated on a conjunction of two triple patterns.

Example 5.4.

```
P1 =
GRAPH ?src {
    { ?x researchTopic ?y .}
    { ?x affiliatedWith ?z .} }
quads(P1) =
GRAPH ?src { ?x researchTopic ?y .}
GRAPH ?src { ?x affiliatedWith ?z .}
```

Now we define the evaluation of complex graph patterns by operations on sets of variable assignments similar to [12, 14].

Definition 5.4 (Complex graph pattern matching). *Let P_1, P_2 be complex graph patterns. The evaluation of graph patterns over K, denoted by $[\![\cdot]\!]_K$, is defined recursively:*

1. $[\![\text{GRAPH } \gamma \ \{\alpha\,\beta\,\delta\}]\!]_K$ *is given by definition 5.3,*
2. $[\![\text{GRAPH g P}_1]\!]_K = [\![quads(P_1)]\!]_K$,
3. (a) $[\![P_1 \text{ AND } P_2]\!]_K = [\![P_1]\!]_K \bowtie [\![P_2]\!]_K$,
   (b) $[\![P_1 \text{ OPT } P_2]\!]_K = [\![P_1]\!]_K \rightarrowtail\!\!\bowtie [\![P_2]\!]_K$,
   (c) $[\![P_1 \text{ UNION } P_2]\!]_K = [\![P_1]\!]_K \cup [\![P_2]\!]_K$,
4. $[\![P_1 \text{ FILTER } C]\!]_K = \sigma_c([\![P_1]\!]_K)$,

The definition uses the operation AND. In standard SPARQL the operation AND is denoted by the absence of an operator. Like [12, 14] we still use the explicit term AND in order to facilitate referencing to this operator.

The recursion in the SPARQL query evaluation defined here is indeed identical to [12, 14]. Only the basic pattern matching has been changed slightly. Basic pattern matching now considers quadruples and it annotates variable assignments from basic matches with atomic statements from $\mathfrak{S}$ and variable assignments from complex matches with Boolean formulae $F \in \mathfrak{F}$ over S.

As an example, consider the query from Example 5.5 evaluated on the knowledge base from Example 5.2.

Example 5.5.

```
SELECT ?h1 ?h2 ?x ?y
FROM NAMED G1
FROM NAMED G2
WHERE {
    {GRAPH ?h1 {?x affiliatedWith ?y}} AND
    {GRAPH ?h2 {?x researchTopic SemanticWeb}}
    FILTER  {?x=JamesHendler}
}
```

Let P be the graph pattern contained in the WHERE clause of the query. Then the evaluation of P is defined by an algebraic expression:

$$\begin{aligned}
[\![P]\!]_K &= [\![\{P_1 \text{ AND } P_2\} \text{ FILTER } \{?\text{x} = \text{JamesHendler}\}]\!]_K \\
&= \sigma_{?\text{x=JamesHendler}}([\![P_1 \text{ AND } P_2]\!]_K) \\
&= \sigma_{?\text{x=JamesHendler}}([\![P_1]\!]_K \bowtie [\![P_2]\!]_K) \\
&= \sigma_{?\text{x=JamesHendler}}(\Phi_1 \bowtie \Phi_2)
\end{aligned}$$

where Φ_1 and Φ_2 are relations representing variable assignments and their annotations. In this example and in the preceding definition we have used algebraic operations on sets of annotated bindings. However, we have not yet explained how these operations are used to construct formulas representing the how-provenance. The following definition will specify how complex formulae from $\mathfrak{F}$, which serve as annotations for results of matching complex graph pattern, will be derived.

Definition 5.5 (Algebra of Annotated Relations). *Let Φ, Φ_1 and Φ_2 be sets of annotated variable assignments. We define $\bowtie$, $\cup$, $\setminus$ and σ, $\rightarrowtail\!\!\bowtie$ via operations on the annotations of the assignments as following:*

- $(\Phi_1 \bowtie \Phi_2)(\mu) = \Phi_1(\mu_1) \wedge \Phi_2(\mu_2)$, *where* $\forall x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2) : \mu_1(x) = \mu_2(x)$ *and* $\mu = \mu_1 \cup \mu_2$,
- $(\Phi_1 \cup \Phi_2)(\mu) = \Phi_1(\mu) \vee \Phi_2(\mu)$,
- $(\Phi_1 \setminus \Phi_2)(\mu) = \Phi_1(\mu) \wedge \neg\left(\bigvee_{\mu_i, \Phi_2(\mu_i) \neq \bot} \Phi_2(\mu_i)\right)$, *where* $\forall x \in \text{dom}(\mu_i) \cap \text{dom}(\mu) : \mu_i(x) = \mu(x)$.
- $(\sigma_c(\Phi))(\mu) = \Phi(\mu) \wedge f_c(\mu)$, *where $f_c(\mu)$ denotes a function mapping μ to either $\top$ or $\bot$ according the condition c.*
- $(\Phi_1 \rightarrowtail\!\!\bowtie \Phi_2)(\mu) = (\Phi_1 \bowtie \Phi_2)(\mu) \vee (\Phi_1 \setminus \Phi_2)(\mu)$.

Let us now continue the evaluation of the query specified in Example 5.5. In order to evaluate the expression $\sigma_{?\text{x=JamesHendler}}(\Phi_1 \bowtie \Phi_2)$ we need to determine Φ_1 and Φ_2 using definition 5.3. The intermediate result is shown in example 5.6. To evaluate the conjunction of two quadruple patterns the operation $\bowtie$ is applied, the result is shown in example 5.7. The annotation $\theta_1 \wedge \theta_2$ of the first row represents that this assignment has been derived from the conjunction of the two literal statements θ_1 and θ_2 (see example 5.2). Application of the σ-operation to the intermediate results gives the annotated relation shown in example 5.8.

Example 5.6.

$\Phi_1 =$

?h1	?x	?y	A_1
G1	JamesHendler	RensselaerPI	θ_2
G2	JamesHendler	UnivMaryland	θ_4
G2	RudiStuder	UnivKarlsruhe	θ_6

$\Phi_2 =$

?h2	?y	A_2
G1	JamesHendler	θ_1
G2	RudiStuder	θ_5

Example 5.7.

$\Phi_1 \bowtie \Phi_2 =$

?h1	?h2	?x	?y	A_3
G1	G1	JamesHendler	RensselaerPI	$\theta_1 \wedge \theta_2$
G1	G2	JamesHendler	UnivMaryland	$\theta_1 \wedge \theta_4$
G2	G2	RudiStuder	UnivKarlsruhe	$\theta_5 \wedge \theta_6$

Example 5.8.

$\sigma_{?x=\text{JamesHendler}}(\Phi_1 \bowtie \Phi_2) =$

?h1	?h2	?x	?y	A_4
G1	G1	JamesHendler	RensselaerPI	$(\theta_1 \wedge \theta_2) \wedge \top$
G1	G2	JamesHendler	UnivMaryland	$(\theta_1 \wedge \theta_4) \wedge \top$

The annotations $\Phi(\mu)$ can now be used to assign truth values for μ. I_s (see definition 4.4) assigns truth values to all atomic statements $s_i \in K \subseteq \mathfrak{S}$. We extend the interpretation I_s to capture all the Boolean formulae over statements $\mathfrak{S}$.

Definition 5.6 (Standard Interpretation of Formulae). *Let $F, F_1, F_2 \in \mathfrak{F}$ be Boolean formulae over $\mathfrak{S}$, let $F_a \in \mathfrak{S}$ be an atomic formula. We define the standard interpretation of formulae I_s^f as follows:*

- $I_s^f(F_a) := I_s(F_a)$;
- $I_s^f(\neg F) := \bot$ *if* $I_s^f(F) = \top$; $I_s^f(\neg F) := \top$ *if* $I_s^f(F) = \bot$;
- $I_s^f(F_1 \wedge F_2)$ *is* $\top$ *if* $I_s^f(F_1) = I_s^f(F_2) = \top$, *otherwise* $\bot$
- $I_s^f(F_1 \vee F_2)$ *is* $\top$ *if* $I_s^f(F_1) = \top$ *or* $I_s^f(F_2) = \top$, *otherwise* $\bot$.

For instance, I_s^f returns $\top$ for the assignment shown in the first row of $\Phi_1 \bowtie \Phi_2$ from example 5.7, because the statements θ_1 and θ_2 are in the knowledge base.

Analogously to I_s^f, we can extend a Π-interpretation I_π over RDF$^+$ statements to a Π-interpretation I_π^f over formulae. Remember that meta knowledge interpretations allow for only one ω per $\theta \in \Theta$ and $\pi \in \Pi$ (Definition 4.5). In order to make use of the how-provenance represented by the annotations we require that for each meta knowledge property π an algebra $(\Omega_\pi, \wedge_\pi, \vee_\pi, \neg_\pi, \top_\pi, \bot_\pi)$ with three operations $\wedge_\pi, \vee_\pi, \neg_\pi$ and two special elements $\top_\pi, \bot_\pi \in \Omega_\pi$ is defined. The definition of the algebras can be supplied by a modeler according to the intended semantics of the different meta knowledge properties.

Definition 5.7 (Π-Interpretation of Formulae). *Let $F, F_1, F_2 \in \mathfrak{F}$ be Boolean formulae over $\mathfrak{S}$, let $F_a \in \mathfrak{S}$ be an atomic formula. We define the interpretation I_π^f as follows:*

- $I_\pi^f(F_a) := I_\pi(F_a)$;
- $I_\pi^f(\neg F)$ *is* $\neg_\pi I_\pi^f(F)$;
- $I_\pi^f(F_1 \wedge F_2)$ *is* $I_\pi^f(F_1) \wedge_\pi I_\pi^f(F_2)$;
- $I_\pi^f(F_1 \vee F_2)$ *is* $I_\pi^f(F_1) \vee_\pi I_\pi^f(F_2)$;

For illustration we consider in Example 5.9 the definition of fuzzy logic operations to calculate a possibility measure on variable assignments, operations defined on timestamps which calculate the time of the last modification, and set operations defined for source documents that construct the combined provenance.

Example 5.9.

$$I^f_{certainty}(x_1 \wedge x_2) = \min(I^f_{certainty}(x_1), I^f_{certainty}(x_2))$$
$$I^f_{certainty}(x_1 \vee x_2) = \max(I^f_{certainty}(x_1), I^f_{certainty}(x_2))$$
$$I^f_{certainty}(\neg x_1) = 1 - I^f_{certainty}(x_1)$$
$$\Omega_{certainty} = [0, 1]$$

$$I^f_{time}(x_1 \wedge x_2) = \max(I^f_{time}(x_1), I^f_{time}(x_2))$$
$$I^f_{time}(x_1 \vee x_2) = \min(I^f_{time}(x_1), I^f_{time}(x_2))$$
$$I^f_{time}(\neg x_1) = 0$$
$$\Omega_{time} = [0, \infty)$$

$$I^f_{source}(x_1 \wedge x_2) = I^f_{source}(x_1) \cup I^f_{source}(x_2)$$
$$I^f_{source}(x_1 \vee x_2) = I^f_{source}(x_1) \cup I^f_{source}(x_2)$$
$$I^f_{source}(\neg x_1) = \{\}$$
$$\Omega_{source} = 2^D, \text{ D the set of document URIs}$$

Query forms. In standard SPARQL query forms, such as SELECT and CONSTRUCT, allow to specify how resulting variable bindings or RDF graphs, respectively, are formed based on the solutions from graph pattern matching [16]. Modifiers, e.g. for projection and ordering, can be applied. The evaluation of SPARQL queries on RDF$^+$ data differs in that meta knowledge is attached to the results.

The evaluation of SELECT queries on an RDF$^+$ dataset is based on $\text{project}_X([\![P]\!]_K)$, where X denotes the set of variables specified in the *SelectExpression* and project is defined as following:

Definition 5.8 (Projection). *Let Φ be a set of annotated variable assignments and X be a set of variables, then*

$$(\text{project}_X(\Phi))(\mu) = \begin{cases} \bigvee_{\forall x \in X: \mu(x)=\nu(x),\, \Phi(\nu)\neq\bot} \Phi(\nu), & \text{if } \mu \text{ is a partial function defined only on } X, \\ \bot, & \text{else} \end{cases}$$

If X forms a proper subset of the variables used in the graph pattern then the annotations of all bindings ν are aggregated. This aggregation is analog to the generation of grouped meta knowledge described in Definition 4.8. As an example consider the query shown in Example 5.10, which is a slight modification of the query from Example 5.5, applied to the data shown in Example 5.2. For the result see Example 5.11. In contrast to Example 5.7 there is only one row for *JamesHendler*.

Example 5.10.

```
SELECT ?x
WITH META G3, G4
FROM NAMED G1
FROM NAMED G2
WHERE {
    {GRAPH ?h1 {?x affiliatedWith ?y}} AND
    {GRAPH ?h2 {?x researchTopic "SemanticWeb"}}
}
```

Example 5.11.

$\text{PROJECT}_{\{?x\}}(\Phi_1 \bowtie \Phi_2) =$

?x	A_5
JamesHendler	$(\theta_1 \wedge \theta_2) \vee (\theta_1 \wedge \theta_4)$
RudiStuder	$\theta_5 \wedge \theta_6$

The result of a SELECT query is a set of extended bindings. Such an extended binding contains values for the specified variables and values for each meta knowledge property $\pi \in \Pi$ which can be regarded as additional variables. For each binding μ these variables π are bound to $I^f_\pi(\text{PROJECT}_X([\![P]\!]_K)(\mu_i))$, see Example 5.12. For this result the meta knowledge from Example 5.13 has been used. For instance $I^f_{certainty}((\theta_1 \wedge \theta_2) \vee (\theta_1 \wedge \theta_4)) = 0.9$. If no meta knowledge statement (θ, π, ω) exists for a particular RDF$^+$ literal statement $f_5(\theta)$ and a particular meta knowledge property π then $\perp_\pi$ serves as default value. For the result of a SELECT query all bindings from $\text{PROJECT}_X([\![P]\!]_K)$ are extended in this way.

Example 5.12.

?x	*certainty*	*time*
JamesHendler	0.9	5/5/2007
RudiStuder	0.7	8/8/2003

Example 5.13.

```
M = {
(θ1, mk:certainty, 0.9),
(θ1, mk:time, "5/5/2007"),
(θ2, mk:certainty, 0.9),
(θ2, mk:time, "5/5/2007"),
(θ3, mk:certainty, 0.6),
(θ3, mk:time, "6/6/2001"),
(θ4, mk:certainty, 0.6),
(θ4, mk:time, "6/6/2001"),
(θ5, mk:certainty, 0.6),
(θ5, mk:time, "6/6/2001"),
(θ6, mk:certainty, 0.6),
(θ6, mk:time, "6/6/2001")}
```

Analogously to standard evaluation, the evaluation of a CONSTRUCT query on an RDF$^+$ dataset results in a single RDF$^+$ graph which is built using the graph template specified in the *ConstructExpression* (see Definition 5.2). This is in line with the fact that the graph template consists of a conjunction of triple patterns and thus quadruple patterns cannot be stated.[2] Similar to the evaluation of SELECT queries the evaluation of CONSTRUCT queries is based on $\text{PROJECT}_X([\![P]\!]_K)$, where X denotes the set of variables specified in the *ConstructExpression*. The RDF$^+$ graph is constructed as described in the following:

[2]Standard SPARQL does not allow for giving this graph a name. In order to associate meta knowledge, multiple named graphs as outputs are convenient. In order to remain standard compliant, the SPARQL engine may however also return data and meta knowledge in two different batches distinguished by some implementation-specific mechanism.

Let t_j denote triple pattern j specified in the *ConstructExpression*, P denote the graph pattern specified in the WHERE-clause, $(s_{i,j}, p_{i,j}, o_{i,j})$ denote the triple obtained by replacing the variables in t_j according to a mapping μ_i and $\hat{g}$ denote a new graph name. Then, for each binding $\mu_i \in \text{PROJECT}_X([\![P]\!]_K)$ and for each t_j the quintuple $(\hat{g}, s_{i,j}, p_{i,j}, o_{i,j}, \theta_{i,j})$ is added to $\mathfrak{S}$, where $\theta_{i,j}$ is the statement identifier $f_1(\hat{g}, s_{i,j}, p_{i,j}, o_{i,j})$. Further $(\theta_{i,j}, \pi, \omega_{i,j})$ is added to $\mathfrak{M}$, where $\omega_{i,j} = I^f_\pi(\text{PROJECT}_X([\![P]\!]_K)(\mu_i))$.

Each new quintuple inherits the meta knowledge properties π associated with the binding which has been used to create that quintuple. The value of $\omega_{i,j}$ is determined by applying I^f_π to the formula which annotates the binding. Note that since $\text{PROJECT}_X([\![P]\!]_K)$ and the interpretations I^f_π are functions and further the graph template in *ConstructExpression* is a set of triples the meta knowledge properties $(\theta_{i,j}, \pi, \omega_{i,j})$ are unique for a given $\theta_{i,j}$.

As an example for a CONSTRUCT statement consider Example 5.14. Meta knowledge for some of the RDF$^+$ statements presented in example 5.2 is specified in example 5.13. For graph pattern P contained in this query the result of $\text{PROJECT}_X([\![P]\!]_K)$ is identical to the annotated relation shown in Example 5.7 except for the first two columns. Based on the single triple pattern `?x worksAt ?y` contained in the graph template and the two bindings contained in $\text{PROJECT}_X([\![P]\!]_K)$ two quintuples are constructed and added to the RDF$^+$ literal statements K_{res} as shown in Example 5.15. M_{res} contains the corresponding meta knowledge statements resulting from $I^f_\pi(\text{PROJECT}_X([\![P]\!]_K)(\mu_i))$.

Example 5.14.

```
CONSTRUCT {?x worksAt ?y}
WITH META G3, G4
FROM NAMED G1
FROM NAMED G2
WHERE {
    {GRAPH ?h1 {?x affiliatedWith ?y}} AND
    {GRAPH ?h2 {?x researchTopic SemanticWeb}}
}
```

Example 5.15.

```
Kres = {
(Gnew, JamesHendler, worksAt, RensselaerPI, θnew1)
(Gnew, JamesHendler, worksAt, UnivMaryland, θnew2)}
(Gnew, RudiStuder, worksAt, UnivKarlsruhe, θnew3)}
Mres = {
(θnew1, mk:certainty, 0.9),
(θnew1, mk:time, "5/5/2007"),
(θnew2, mk:certainty, 0.6),
(θnew2, mk:time, "6/6/2001")
(θnew3, mk:certainty, 0.6),
(θnew3, mk:time, "6/6/2001") }
```

6. TASKS AND BENEFITS

This section summarizes the discussed steps of meta knowledge representation and utilization for the sample scenario that was introduced in section 2.

6.1 Tasks for the administrator

In order to represent and utilize meta knowledge, the system administrator has to make some design choices. In particular, the application-specific meta knowledge properties must be defined. In our sample scenario, we consider three meta knowledge properties: source, certainty, and timestamp. In the next step, the administrator defines the intended semantics of these properties in order to facilitate query processing with complex expressions and pattern combinations. Using the notion from Section 5.1, we assume that corresponding definitions for meta knowledge properties are defined according to previously discussed Example 5.9.

Finally, data and available associated meta knowledge are represented in RDF using named graphs [2, 5], and imported into our RDF$^+$-based repository.

6.2 Processing performed by the System

We assume that the administrator manages the small sample knowledge base introduced in section 2. The knowledge base is transformed into the RDF$^+$ quintuples shown in Example 5.2 as discussed in section 4. Associated meta knowledge is transformed into further RDF$^+$ literal statements and RDF$^+$ meta knowledge statements. For the properties *mk:time* and *mk:certainty* the latter are shown in Example 5.13.

Following our sample scenario, the query from Example 2.3 can be reformulated as the query from Example 5.14 which retrieves names of Semantic Web experts together with their affiliations. Internally, the query processor evaluates this query using graph patterns as discussed in 5.1. If P denotes the graph pattern from this query then all matches for all variables in P are given by $[\![P]\!]_K$. The resulting set of annotated variable assignments is shown in Example 5.7. It contains possible variable assignments, and the how-provenance (A_3) that explains how these source statements have been used.

By combining this information with definitions for meta knowledge properties and available meta knowledge statements, the query processor constructs the result shown in Example 5.15. This result is then serialized in RDF.

6.3 Benefits for the user/developer

The user or application developer can access the knowledge stored in the RDF$^+$-based repository in different ways. On one hand, the repository does not change the existing SPARQL semantics and thus fully supports common SPARQL queries. This is an important advantage for compatibility with existing applications and interfaces. On the other hand, the repository supports the advanced SPARQL syntax with metaknowledge support (section 5.1). Thus, the user obtains additional access to valuable meta knowledge that can be used for relevance ranking, conflict resolution, or other applications in connection with retrieved knowledge.

In our application scenario, the user may realize that the query answer is potentially contradictive (James Hendler is affiliated with Rensselaer PI and University of Maryland). By inspecting the associated meta knowledge, he would realize that the second fact was generated by mistake. In fact, it is based on outdated information (knowledge from the document *survey.pdf* with timestamp *6/6/2001*) that was wrongly combined with knowledge from a more recent source (namely document *report.doc* with timestamp *5/5/2007*). It turns out that the affiliation of James Hendler has actually changed from U Maryland to Rensselaer PI, and the erroneous tuple can be safely excluded from further processing.

7. COMPLEXITY

In this section we analyze how the construction of the annotations influences the complexity of the decision problem related to SPARQL. The decision problem associated with the evaluation of a SPARQL query can be stated as following [12]: *Given an RDF dataset D, a graph pattern P and a mapping μ, determine whether μ is in the result of P applied to D.* For this decision problem, which we denote by EVAL, an analysis of the complexity is presented in [12, 13]. In the context of RDF$^+$ datasets and annotated variable assignments we have a slightly different decision problem: *Given an RDF$^+$ dataset D^+, an RDF$^+$ graph pattern P^+, a variable assignment μ and an annotation α determine whether α is the correct annotation of μ.* We denote this problem by EVAL$^+$. An annotation is correct iff the formula is *equivalent* (in the logical sence) to the formula obtained by evaluating P^+ as defined in section 5.

With the following two theorems we show that for patterns which do not use the OPTIONAL operator EVAL$^+$ has the same complexity as EVAL. The RDF counterparts of both theorems have been established by [12, 13]. Like [12, 13] we restrict P^+ to graph pattern which do not contain blank nodes. In the first theorem we consider graph pattern which use only AND and FILTER operations. Bindings obtained by such pattern are annotated with formulae which do not contain any other operator besides $\wedge$ according to definition 5.4.

THEOREM 7.1. EVAL$^+$ *can be solved in time $O(|P^+| \cdot |D^+|)$ for graph pattern expressions constructed by using only AND and FILTER operators and for annotation formulas using only the operation $\wedge$.*

Proof: The proof consists of two parts: In the first part we construct a correct annotation $\hat{\alpha}$ for μ and in the second we check whether α and $\hat{\alpha}$ are equivalent. In the following let p_i denote quadruple pattern i from P$^+$ and r(μ, p_i) denote the quadruple obtained by replacing the variables in the quadruple pattern p_i according to μ.

In order to construct $\hat{\alpha}$ we start by evaluating $[\![p_i]\!]$ for all quadruple pattern using definition 5.3. In order to achieve this D^+ needs to be searched for all quadruples r(μ, p_i). This can be performed in $O(|P^+| \cdot |D^+|)$. Then, we construct the algebraic expression ψ by evaluating $[\![P^+]\!]$ using definition 5.4. This can be performed by traversing P^+. The correct annotation $\hat{\alpha}$ of μ in the result of evaluating P^+ on D^+ is defined as $\psi(\mu)$, see definition 5.5. We can construct $\hat{\alpha} = \psi(\mu)$ in a depth-first traversal of ψ as following:

Let ψ_1, ψ_2 be algebraic expressions part of ψ. For each join operation in ψ the annotation of $(\psi_1 \bowtie \psi_2)(\mu)$ is given by $\psi_1(\mu) \wedge \psi_2(\mu)$. For each select operation $\sigma_c(\psi)(\mu)$ is given by $\psi(\mu)$ if condition c is fulfilled otherwise it is given by $\perp$. Since traversing P^+ and ψ each has time complexity $O(|P^+|)$ the evaluation of $\hat{\alpha} = \psi(\mu)$ remains in $O(|P^+| \cdot |D^+|)$.

Now we determine whether for a given annotation α holds $\alpha \equiv \hat{\alpha}$. We transform both formulas into a normal form in $O(|P^+| \cdot \log |P^+|)$ using associativity, commutativity and idempotency of $\wedge$. First, we remove all brackets then establish an order among atomic formulas (identifiers and $\top, \bot$) and finally remove duplicates. For α and $\hat{\alpha}$ normalized this way syntactic equality implys logical equivalence. Assuming $|P^+| < |D^+|$ the overall complexity remains in $O(|P^+| \cdot |D^+|)$. ■

Theorem 7.2. Eval$^+$ *is NP-complete for graph pattern expressions constructed by using only AND, FILTER and UNION operators.*

Proof: The proof consists of two parts: In the first part we show that Eval$^+$ is contained in NP and in the second we establish NP-hardness of Eval$^+$. As above, let p_i denote quadruple pattern i from P^+ and $\mathrm{r}(\mu, p_i)$ denote the quadruple obtained by replacing the variables in the quadruple pattern p_i according to μ.

The first part consists of two steps as well: first we construct a correct annotation $\hat{\alpha}$ for μ and then we check whether α and $\hat{\alpha}$ are equivalent. In order to construct $\hat{\alpha}$ we start by evaluating $[\![p_i]\!]$ for all quadruple pattern using definition 5.3. In order to achieve this D^+ needs to be searched for all quadruples $\mathrm{r}(\mu, p_i)$. Then, we construct the algebraic expression ψ by evaluating $[\![P^+]\!]$ using definition 5.4. This can be performed by traversing P^+. The correct annotation $\hat{\alpha}$ of μ in the result of evaluating P^+ on D^+ is defined as $\psi(\mu)$, see definition 5.5. We can construct $\hat{\alpha} = \psi(\mu)$ in a depth-first traversal of ψ as following:

Let ψ_1, ψ_2 be algebraic expressions part of ψ. For each join operation $(\psi_1 \bowtie \psi_2)(\mu)$ is given by $\psi_1(\mu) \land \psi_2(\mu)$. For each union operation $(\psi_1 \cup \psi_2)(\mu)$ is given by $\psi_1(\mu) \lor \psi_2(\mu)$. For each select operation $\sigma_c(\psi)(\mu)$ is given by $\psi(\mu)$ if condition c is fulfilled otherwise it is given by $\bot$.

Time complexity for evaluating $\psi(\mu)$ is $O(|P^+| \cdot |D^+|)$. The evaluation of $\alpha \equiv \hat{\alpha}$ is more difficult if UNION operations are contained in P^+. But it is subsumed by checking equivalence of Boolean formulae which is a NP-complete problem. Thus, the decision problem Eval$^+$ is contained in NP.

We can deduce that Eval$^+$ is an NP-complete problem if Eval, which has been shown to be NP-complete [12, 13], can be reduced to it. Eval can be reduced to Eval$^+$ if the evaluation defined in section 5 results in an annotation $\alpha \equiv \bot$ exactly for such bindings which are not in the result of standard SPARQL evaluation. For graph pattern which do not contain the OPTIONAL operator this can be shown by induction over the structure of algebraic expressions using definition 5.4 and its standard SPARQL counterpart from [12]. ■

8. IMPLEMENTATION

The framework described in this paper has been implemented and is available as an initial prototype. The prototype is available as an open source implementation at
`http://isweb.uni-koblenz.de/Research/MetaKnowledge`
together with example queries using artificial data from the LeHigh benchmark[3].

9. EXPERIMENTS

In order to evaluate the overhead produced by the evaluation of meta knowledge properties for results of SPARQL queries we carried out two experiments based on the well-known LUBM benchmark [9] and our implementation of the meta knowledge framework. Our main aim is to find out whether the evaluation of SPARQL queries remains feasible if provenance and meta knowledge is provided for query results and thus to support the theoretical results of section 7.

A key question is how to separate the additional effort for the evaluation of provenance and meta knowledge from standard SPARQL processing. Triples describing meta knowledge receive an *additional* meta knowledge interpretation according to section 4. At the same time they are also treated as ordinary RDF triples (and thus can be queried using standard SPARQL). Thus, if we add meta knowledge to a knowledge base this increases its overall size which also increases the workload for standard query processing. In order to account for this we compare query evaluations performed on the same knowledge base which includes meta knowledge. Our implementation is built on top of Sesame[4] 2.0 (beta 6) using query rewriting. The triple store is used to store both, knowledge and meta knowledge. We expect that a native implementation of the meta knowledge framework can achieve an increased performance.

Query evaluation. For the evaluation of SPARQL on RDF$^+$ we defined the results of SELECT queries to be set-valued, see section 5.2. For standard SPARQL [16], however, a SELECT query returns a *solution sequence* which may contain duplicate elements. All 14 queries of the LUBM benchmark are SELECT queries. To allow for a consistent comparison of extended evaluation on one side and standard evaluation on the other we added the keyword DISTINCT to the queries for standard evaluation. This tells the query processor to eliminate duplicates. Since the evaluation of individual meta knowledge properties can be arbitrarily complex we compare the following three kinds of query evaluation:

- standard evaluation with additional duplicate elimination (SD), as performed by Sesame,
- evaluation of provenance formulae for each query result (PF) and
- evaluation of four basic meta knowledge properties (M4), namely *agent*, *confidence*, *creation time* and *source*, see Example 5.9 (we evaluate *agent* analogously to *source*).

Only the last type of query processing actually makes use of the additional triples.

Data. We added artificial meta knowledge to the LUBM data. Amount and granularity of the additional meta knowledge are key properties of the resulting dataset. We created two datasets containing a different percentage of meta knowledge triples. In the first dataset (MK29) 10 triples each are assigned to a named graph and artificial values of the four meta knowledge properties are attached to it. As a consequence the dataset contains 29 percent of meta knowledge which might be a reasonable scenario in a real world scenario. In the second dataset (MK400) each single triple is assigned to a different graph and the four meta knowledge properties are associated with it. This way the knowledge base contains four times as many meta knowledge triples as knowledge triples.

[3]available at `http://swat.cse.lehigh.edu/projects/lubm/`

[4]`www.openrdf.org`

	SD	PF	M4
MK29	313	894	1382
MK400	196	104	425

Table 1: Random query sequence experiments: average processing time (ms).

The data set MK29 was created based on LUBM OWL data for 10 Universities. We added the meta knowledge by putting groups of 10 consecutive original triples into one named graph and associating random values of the meta knowledge properties *agent*, *confidence*, *creation time* and *source* with this graph name, see Example 9.1.

Example 9.1.

```
<graph>
  <uri>
     http://www.x-media-project.org/ontologies/someGraph#0-0_30
  </uri>
  <triple>
    <uri>http://www.Department0.University0.edu/
                                           FullProfessor1</uri>
    <uri>
      http://www.lehigh.edu/%7Ezhp2/2004/0401/
                        univ-bench.owl#doctoralDegreeFrom</uri>
    <uri>http://www.University882.edu</uri>
  </triple>
  <triple>
    <uri>http://www.University882.edu</uri>
    <uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</uri>
    <uri>http://www.lehigh.edu/%7Ezhp2/2004/0401/
                                univ-bench.owl#University</uri>
  </triple>
...
<graph>
  <uri>http://www.metaknowledge.semanticweb.org0-0_30</uri>
  <triple>
    <uri>http://www.x-media-project.org/ontologies/
                                         someGraph#0-0_30</uri>
    <uri>http://www.x-media.org/ontologies/metaknow#source</uri>
    <uri>http://www.x-media-project.com/ex#source30</uri>
  </triple>
  <triple>
    <uri>http://www.x-media-project.org/ontologies/
                                         someGraph#0-0_30</uri>
    <uri>http://www.x-media.org/ontologies/metaknow
                                       #confidence_degree</uri>
    <typedLiteral datatype='http://www.w3.org/2001/
                          XMLSchema#double'>0.7</typedLiteral>
  </triple>
...
```

The resulting dataset consists of 1.8 million triples of which 1.3 million triples were created by the LUBM generator to which we added 0.5 million meta knowledge triples. It contains 0.3 million (additional) graph URIs.

The data set MK400 was created based on LUBM OWL data for three Universities. The resulting dataset consists of 1.7 million triples of which 0.35 million are orignal LUBM data and 1.4 million are additional metadata. Note that the overall sizes of the two datasets are comparable. As indicated above the overall size of the knowledge base influences the workload for query processing. In fact, main memory consumption appeared to be a key factor. By choosing similar sizes for the two datasets we reduce the influence of factors related to the size of the knowledge base and concentrate on how the different evaluations influence the processing time.

Random Query Sequence. We conducted two experiments with each of the two datasets. One experiment aims at simulating behavior of a query engine in a real life scenario: first a dataset is loaded and then a sequence of queries is submitted to the query engine. We measured the average processing time of these queries. The query sequence was created based on 8 of the 14 queries from the LUBM benchmark. Query 2 was not included since we were not able to obtain results for this query with this version of Sesame, this dataset and an average machine. Five more queries are discarded since they require OWL inferencing or hierarchy information to obtain complete results and Sesame 2 was not able to obtain bindings using plain SPARQL processing. Since no meta knowledge needs to be calculated if the result set is empty using these queries would bias the evaluation in favor of the meta knowledge processing. The authors of the benchmark identified three main characteristics of queries with respect to plain SPARQL processing: *input size*, *selectivity* and *complexity*. The remaining 8 queries still cover different settings for these features. Experiments with single queries will be presented below.

The query sequence consisted of a random shuffle of 20 copies of each of the remaining queries. For each query in the sequence we measured the time which elapses during issuing the query, obtaining the result and traversing the result sequentially. This measure is similar to the *query responce time* defined in [9]. The only difference is that in [9] each query is performed ten times after the knowledge base has been loaded to measure the caching performance of the query engine. For each run we determined the average of the execution times of the queries in the sequence in question. The results are summarized in Table 1.

Evaluation of provenance formulae (PF) given dataset MK29 almost tripled the average query execution time. On average, the evaluation took about half a second longer than standard evaluation (SD). The evaluation of four meta knowledge properties (M4) adds again half a second to the evaluation of provenance. The overall overhead to obtain meta knowledge is about a second given a non-trivial dataset. We consider this to be an indication for the feasibility of our approach. Since these numbers are average values the question remains whether reasonable processing times are achieved for all individual queries as well. This will be analyzed below.

For the dataset MK400 the computations were faster for all three kinds of evaluations. This can be explained by the fact that this dataset contains a smaller number of knowledge triples and therefore the result sets for some of the queries contain fewer bindings as we will see below. Surprisingly, evaluation of provenance formulas needed less time than standard evaluation. A possible explanation is optimization performed by the qurery processor of Sesame 2. Since our implementation uses query rewriting different queries are evaluated for the different kinds of evaluations. Optimization techniques might be easier to apply to some of them. Why does the evaluation of MK29 not show similar charactерisitcs? Here a possible explanation is the larger number of bindings involved in the evaluation. Main memory consumption was quite large in both cases. Possibly there was no space left for caching of (intermediary) results given dataset MK29. The calculation of the four meta knowledge properties did result in an increase in processing time as expected.

Single LUBM Queries. We also measured processing times for single queries. As stated above we measured the time which

Query		*Q1*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q10*	*Q14*	*Av.*
# bindings		5	10	411	24019	3	1874	4	75547	
processing time (ms)	SD	127	163	211	357	135	1619	127	977	465
	PF	324	347	386	995	337	691	325	2320	716
	M4	340	375	426	2187	346	878	326	10501	1922

Table 2: Processing time of query evaluation with and without provenance and meta knowledge for individual queries and the MK29 dataset. Processing times are average values of 10 runs each.

Query		*Q1*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q10*	*Q14*	*Av.*
# bindings		5	10	411	6390	3	1874	4	19868	
processing time (ms)	SD	137	189	204	217	131	1596	124	379	372
	PF	367	350	367	517	316	676	300	769	458
	M4	346	359	440	859	321	1083	306	1953	708

Table 3: Processing time of query evaluation with and without provenance and meta knowledge for individual queries and the MK400 dataset. Processing times are average values of 10 runs each.

elapsed during during issuing the query, obtaining the result and traversing the result sequentially. In contrast to the previous experiment and to the definition of the *query response time* from [9] the application was restarted before each single query execution. That way each query was evaluated against a newly loaded knowledge base since we want to measure the effort it takes to evaluate the queries and not the caching strategy of the query engine. This procedure was repeated 10 times for each query and each method of query evaluation. The average values from these runs are summarized in tables 2 and 3. The standard deviations estimated from these 10 runs are less than or equal 10 percent for all queries and methods evaluated on the two datasets.

On average the calculation of provenance formulas (PF) increases processing time by factor 1.5. In absolut numbers the average increase is about 0.2 seconds. The largest increase (1.3 seconds for the MK29 dataset) can be observed for query 14 which also gives the largest result set. We attribute this to the main memory consumption of our implementation. For query 8 there even is a decrease in processing time. As for the case of a random query sequence evaluated on MK400 a possible explanation are optimizations of the query processor. The additional querying for meta knowledge might guide the optimization of the query execution.

Calculation of the four basic provenance properties (MD4) causes an average increase of factor 4.1 (1.5 seconds) for the MK29 dataset and factor 1.9 (0.3 seconds) for the MK400 dataset compared to standard evaluation. We ascribe the larger increase for the runs based on dataset MK29 to the larger number of results for queries 6 and 14 and the non-optimized memory consumption of our implementation. These results are in line with the results from the experiments using a random query sequence shown in table 1. For query 8 the processing time decreases for query evaluation including meta knowledge as well which might be explained by query optimization of the underlying triple store as stated above.

Conclusions. At first we consider the experiments with individual queries. From the estimated standard deviations of the 10 runs for each query, kind of evaluation and dataset we conclude that the measurements are reliable enough to draw two general conclusions: On the one hand we can observe a noticeable increase of processing time using our implementation and on the other hand the amount of this increase can be described by a small linear factor.

The results of the experiments using a random sequence of queries indicate that similar results also hold for real world scenarios. Here caching can be applied by a query processor. A few times evaluations of the same query even were repeated directly one after another in the sequence. The resulting average processing times are smaller but still comparable to the average values for the evaluations of single queries. A key insight is that the overall processing times remain feasible for evaluations on a dataset of up to 1.8 million triples and results of up to 75,000 bindings.

If we compare the processing times obtained for the two different datasets we might expect that for dataset MK400 processing times increase by a larger factor if meta knowledge is involved. MK400 contains a larger number of meta knowledge triples which need to be processed in order to evaluate meta knowledge – especially compared to the number of knowledge triples. However, at least with our implementation, the dominant characteristic of query evaluation appears to be the size of the result set.

10. RELATED WORK

The importance of better understanding the ways by which the result came about is fundamental to many Semantic Web applications and scenarios. The specification of the Semantic Web proof layer was discussed in [11, 15, 10]. Our approach is focused on a different language model (RDF) and provides fine-grained meta knowledge management for retrieval queries with SPARQL that is not directly comparable with proof traces for OWL reasoning.

In the area of database systems, meta knowledge is often represented using an extension of the relational data model, coined *annotated relations*. Its purpose is primarily the description of data origins (provenance) and the process by which it arrived as a query answer [6, 3, 4, 7]. Basically, our methodology follows the same idea. However, our approach is specially designed for RDF graph models and not directly comparable to metadata models for relational database systems. The same holds for the query language (SPARQL instead of SQL) and its semantics. An important difference to isolated database solutions is the serialization ability of RDF and thus seamless exchanging and utilization of meta knowledge from our framework across the Semantic Web.

11. CONCLUSION AND FUTURE WORK

In this paper, we presented an original, generic, formalized and implemented approach for the management of many dimensions of meta knowledge, like source, authorship, certainty, and others, for RDF repositories. Our method re-uses existing RDF modeling possibilities in order to represent meta knowledge. Then, it extends SPARQL query processing in such a way that given a SPARQL query for data, one may request meta knowledge without modifying the query proper. We achieve highly flexible and automatically coordinated querying for data and meta knowledge, while completely separating the two areas of concern. Our approach remains compatible to existing standards and query languages and can be easily integrated with existing applications and interfaces.

In the future, we will investigate the meta knowledge support for OWL-based knowledge bases with advanced reasoning capabilities. Due to the substantially higher complexity of inferencing and retrieval algorithms (e.g. reasoning in OWL-DL vs. RDF querying with SPARQL) and the distributed nature of knowledge sources in the Semantic Web, the notion of meta knowledge will require further, non-trivial justification. Another interesting research issue is the support for *nested* meta knowledge (i.e. construction of meta knowledge for the result with respect to additional information *about* meta knowledge of its origins).

Our long-term objective is the generic, efficient and effective infrastructure for meta knowledge management as an integral part of the proof layer of the Semantic Web.

12. REFERENCES

[1] Chris Bizer and Richard Cyganiak. The TriG Syntax. 2007. http://sites.wiwiss.fu-berlin.de/suhl/bizer/TriG/Spec/TriG-20070730/.

[2] Christian Bizer and Jeremy J. Carroll. Modelling Context using Named Graphs. In *W3C Semantic Web Interest Group Meeting*, Cannes, France, 2004.

[3] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Data Provenance: Some Basic Issues. *20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), New Delhi, India*, pages 87–93, 2000.

[4] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and Where: A Characterization of Data Provenance. *Proc. of ICDT*, pages 316–330, 2001.

[5] Jeremy J. Carroll and Patrick Stickler. TriX: RDF triples in XML. In *Proceedings of the Extreme Markup Languages 2004*, Montreal, Canada, 2004.

[6] Y. Cui and J. Widom. Practical Lineage Tracing in Data Warehouses. *Proc. of ICDE*, pages 367–378, 2000.

[7] Li Ding, Pranam Kolari, Tim Finin, Anupam Joshi, Yun Peng, and Yelena Yesha. On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework. In *Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security*, 2005.

[8] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance Semirings. In *PODS*, pages 31–40, 2007.

[9] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[10] D. McGuinness and P. Pinheiro da Silva. Explaining Answers from the Semantic Web: the Inference Web Approach. *J. Web Sem.*, 1(4):397–413, 2004.

[11] W. Murdock, D. McGuinness, P. Pinheiro da Silva, C. Welty, and D. Ferrucci. Explaining Conclusions from Diverse Knowledge Sources. *International Semantic Web Conference (ISWC), Athens, USA*, pages 861–872, 2006.

[12] Jorge Perez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. In *Proc. of ISWC*, pages 30–43, 2006.

[13] Jorge Perez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. arXiv:cs/0605124v1 [cs.DB], May 2006.

[14] Jorge Perez, Marcelo Arenas, and Claudio Gutierrez. Semantics of SPARQL. Technical Report TR/DCC-2006-17, Universidad de Chile, October 2006.

[15] P. Pinheiro da Silva, D. McGuinness, and R. Fikes. A Proof Markup Language for Semantic Web services. *Inf. Syst.*, 31(4-5):381–395, 2006.

[16] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Working draft, W3C, March 2007. http://www.w3.org/TR/rdf-sparql-query/.

[17] M. Schraefel, N. Shadbolt, N. Gibbins, S. Harris, and H. Glaser. CS AKTive Space: Representing Computer Science in the Semantic Web. *Proc. of WWW*, pages 384–392, 2004.

[18] Bernhard Schueler, Sergej Sizov, and Steffen Staab. Management of Meta Knowledge for RDF Repositories. In *Int. Conf. on Semantic Computing (ICSC)*, pages 543–550, Irvine, CA, September 2007.