Don't Like RDF Reification? Making Statements about Statements Using Singleton Property

Vinh Nguyen
Kno.e.sis Center
Wright State University
Dayton, Ohio, USA
vinh@knoesis.org

Olivier Bodenreider
National Library of Medicine,
National Institute of Health
Bethesda, Maryland, USA
olivier@nlm.nih.gov

Amit Sheth
Kno.e.sis Center
Wright State University
Dayton, Ohio, USA
amit@knoesis.org

ABSTRACT

Statements about RDF statements, or meta triples, provide additional information about individual triples, such as the source, the occurring time or place, or the certainty. Integrating such meta triples into semantic knowledge bases would enable the querying and reasoning mechanisms to be aware of provenance, time, location, or certainty of triples. However, an efficient RDF representation for such meta knowledge of triples remains challenging. The existing standard reification approach allows such meta knowledge of RDF triples to be expressed using RDF by two steps. The first step is representing the triple by a Statement instance which has subject, predicate, and object indicated separately in three different triples. The second step is creating assertions about that instance as if it is a statement. While reification is simple and intuitive, this approach does not have formal semantics and is not commonly used in practice as described in the RDF Primer.
In this paper, we propose a novel approach called *Singleton Property* for representing statements about statements and provide a formal semantics for it. We explain how this singleton property approach fits well with the existing syntax and formal semantics of RDF, and the syntax of SPARQL query language. We also demonstrate the use of singleton property in the representation and querying of meta knowledge in two examples of Semantic Web knowledge bases: YAGO2 and BKR. Our experiments on the BKR show that the singleton property approach gives a decent performance in terms of number of triples, query length and query execution time compared to existing approaches. This approach, which is also simple and intuitive, can be easily adopted for representing and querying statements about statements in other knowledge bases.

Categories and Subject Descriptors

H.1.m [**Information Systems**]: Models and Principles;

Table 1: Sample queries for different types of meta knowledge, each query example is assigned an identifier (P, T, S, and C) for references

Query type	Examples
Provenance	P1. Where is this fact from?
	P2. When was it created?
	P3. Who created this fact?
Temporal	T1. When did this event occur?
	T2. What is the time span of this event?
	T3. Which events were in the same year?
Spatial	S1. What is the location of this event?
	S2. Which events were at the same place?
Certainty	C1. What is the confidence of this fact?

Keywords

Semantic Web; Meta triples; RDF; SPARQL; Reification; RDF Singleton Property

1. INTRODUCTION

Representing and querying meta knowledge for triples including provenance, trust, certainty, time, and location have been emerging demands in creating and sharing Semantic Web knowledge bases [8, 15, 16, 17, 19]. Here we use the example from YAGO2 [8] for demonstrating the requirements of meta knowledge for triples and motivating our approach.

1.1 Motivating example

Resource Description Framework (RDF) [5, 12] has been well adopted for creating and sharing various knowledge bases. Knowledge bases that provide a comprehensive collection of facts (e.g., YAGO [20]) have been widely used by various applications. These facts are usually in the form of triples, or subject-predicate-object such as:

```
BobDylan     isMarriedTo    SaraLownds
BarackObama  isPresidentOf  TheUnitedStates.
```

Here we simplify the syntax of resource URIs for readability by eliminating their prefixes. While these facts are useful for finding spouses or political positions of a person, they do not provide sufficient information for answering many types of challenging questions involving meta knowledge. Such lists of query types and their examples are listed in Table 1.

Additional information about the triples must be provided in order to address those queries. Recent knowledge bases such as YAGO2 [8] provide such temporal and spatial information. For example, the information about the sources

Table 2: Reified statements and their meta knowledge assertions for the same fact `BobDylan isMarriedTo SaraLownds` occuring in two documents

Subject	Predicate	Object
BobDylan	isMarriedTo	SaraLownds
stmt#1	rdf:type	Statement
stmt#1	rdf:subject	BobDylan
stmt#1	rdf:predicate	isMarriedTo
stmt#1	rdf:object	SaraLownds
stmt#1	hasSource	wk:Bob_Dylan
stmt#1	extractedOn	2009-06-07
stmt#2	rdf:type	Statement
stmt#2	rdf:subject	BobDylan
stmt#2	rdf:predicate	isMarriedTo
stmt#2	rdf:object	SaraLownds
stmt#2	hasSource	wk:Sara_Dylan
stmt#2	extractedOn	2009-08-08

and dates of the fact `BobDylan isMarriedTo SaraLownds` would help find the answers for question P1 (where is the fact from?) and P2 (when was it created?) in Table 1.

We assume that this fact could be extracted from the wiki pages of Bob_Dylan on 2009-06-07 and Sara_Dylan on 2009-08-08. Using the reification approach, the whole reified statements and their assertions about sources and extraction dates are provided in Table 2.

The fact is represented as an instance of class Statement with three different properties for its subject, predicate and object. Since we need to represent two occurrences of the same statement in two different documents, we create two resources: `stmt#1` and `stmt#2` because if we create only one `stmt#1` for both occurrences, the association of each occurrence with its source and date of extraction together is not distinguishable. The meta information about the fact is represented by `hasSource` and `extractedOn` properties.

The lack of formal semantics connecting a statement and the resource describing it is one of the main drawbacks of using reification for describing triples. Since the resource `stmt#1` describing a statement is not associated with that statement, assertions created for this resource are not the same as assertions created for the original statement as explained in the RDF Primer [12]. Moreover, it is obvious that the reification approach requires four additional triples for representing one statement per document as a resource. This would increase the size of the data sets by at least four times, which is not a scalable approach. It would also make query patterns lengthy for finding when the couple was married or divorced.

1.2 Our approach

In this paper, we address the problem of representing and querying statements about statements, or meta knowledge of triples, by looking at it from a different perspective. Our motivation arises from the question as to whether or not a more efficient mechanism for describing a statement using RDF exists. A good design should provide a formal semantics, use existing syntax and be compatible with existing Semantic Web languages, tools, and methods. The proposed formal semantics should be compatible with the existing model-theoretic semantics in order to avoid conflicts

Table 3: Singleton properties and their meta knowledge assertions for the same fact `BobDylan isMarriedTo SaraLownds` occuring in two documents

N_o	Subject	Predicate	Object
T_1	BobDylan	isMarriedTo#1	SaraLownds
T_2	isMarriedTo#1	singletonPropertyOf	isMarriedTo
T_3	isMarriedTo#1	hasSource	wk:Bob_Dylan
T_4	isMarriedTo#1	extractedOn	2009-06-07
T_5	BobDylan	isMarriedTo#2	SaraLownds
T_6	isMarriedTo#2	singletonPropertyOf	isMarriedTo
T_7	isMarriedTo#2	hasSource	wk:Sara_Dylan
T_8	isMarriedTo#2	extractedOn	2009-08-08

in the RDF/RDFS interpretation. Using the existing RDF syntax would ensure the compatibility of meta triples and existing triple datasets. Such design would overcome the need to develop new or revise available tools and methods for making them work with new meta triples.

This paper proposes a novel approach called *Singleton Property* for representing statements about statements using RDF with regard to the three requirements above. Our approach is based on the intuition that the nature of every relationship is universally unique. The uniqueness of the relationship can be a key for any statement using the new type of property called *singleton property.* A singleton property is a property instance representing one specific relationship between two particular entities under one specific context. Singleton properties can be viewed as instances of generic properties whose extensions contain a set of entity pairs. Similar to the way we assign URIs for generic properties, we assign a URI for each singleton property. For example, for the same statement `BobDylan isMarriedTo SaraLownds`, we can create two singleton property instances describing the occurrences of this statement in two documents as provided in Table 3.

For each document (or context of the fact), we create one separate singleton property instance representing that fact (in T_1, T_5). Particularly we create two singleton properties `isMarriedTo#1` and `isMarriedTo#2` for the relationships extracted from the Wiki pages of Bob Dylan (`wk:Bob_Dylan`) and Sara Dylan (`wk:Sara_Dylan`), respectively. Meta knowledge about the fact from one document can be added as assertions for the singleton property from that document (in T_3, T_4, T_7, and T_8).

These two singleton properties and their generic property `isMarriedTo` (in T_2, T_6) are interconnected via the new property called `singletonPropertyOf`. Since the relationship between a generic property and a singleton property can be viewed as a special binary class and an instance, we define the `singletonPropertyOf` property as a sub property of `rdf:type`. We will provide a more detailed explanation on how we come up with the set of triples listed in Table 3 in the next section.

Our contributions in this paper include:

- A novel approach for representing and querying statements about statements using RDF/SPARQL syntax,
- A formal semantics for interpreting the singleton property terms in RDF/RDFS,
- Two real use cases in existing knowledge bases,

- And a comparison between the singleton property approach and the existing approaches based on three quantitative metrics: number of triples, query length and query execution time.

The remaining sections are organized as follows. Section 2 explains the approach in detail and justifies our design choices for the singleton property. Section 3 provides a formal semantics for the singleton property. The querying mechanism is described in Section 4. We also discuss issues related to how this singleton property could be adopted by existing Semantic Web technologies and standards by providing two use cases in Section 5. We finally report the experiments of the approach in Section 6, and the related work in Section 7. We discuss about the possible future work in Section 8 and conclusion in Section 9.

2. SINGLETON PROPERTY

In this section, we will explain our intuition for the proposed approach and justify our design choices for the singleton property. Here we explain our rationale accounting for the novel perspective that leads to our approach.

2.1 Singleton property as unique key for statement within a context

Back to the motivating example we used in Section 1.1, the reification process represents a triple as a resource, which is an instance of the Statement class [5]. Reifying a statement requires two steps. The first step is to find a resource that uniquely identifies a statement. The second is to create assertions for that statement via that resource. The first step involves finding which resource among the three elements of a triple could fundamentally distinguish statements.

In the Semantic Web, everyone can create any statement. It is possible that the same statements may be created in different datasets by different organizations. Therefore, we need to find a resource that can distinguish any two statements. Given that the statements may be the same, they may be associated with different contextual information when they are created. The information capturing the context when a statement is created could be helpful for identifying statements. Such contextual information of a statement could be described by various dimensions of meta knowledge, including the source recording that statement, the time or place that statement occurs, the certainty of the author about that statement, etc. We can conclude that a statement within a context is unique. Now the next question is, what can represent that uniqueness of a statement within a context? If the same statements are associated with different contexts, are they the same in nature? What remains the same? What becomes different?

From a philosophical point of view, we believe that the existence of two entities in the subject and the object of one statement is independent from the contexts creating that statement. Particularly, they already exist before the statement is created. For example, the existences of Bob Dylan and Sara Lownds do not depend on their marriage, and obviously they also exist before they marry each other. While creating a new statement, what we actually do is connect two existing entities and establish a new relationship between them. Therefore, the contextual information in establishing a new relationship can play the role of a key for any statement. We can manifest that key by creating a new property instance that represents the newly established relationship associated with a context and enforces it to be unique. We call it singleton property. The *singleton* concept is taken from set theory. A singleton set has only one element.

We define a *singleton property* as a unique property instance representing a newly established relationship between two existing entities in one particular context. For example, a new relationship is established for Bob Dylan and Sara Lownds according to two Wiki pages. We can consider each Wiki page as a context associated with the new relationship. Note that here we merely give examples of context and leave the questions of how exactly context is described and how to identify it for data publishers because those are subjective to them. As a result, we can create two singleton properties `isMarriedTo#1` and `isMarriedTo#2` to represent the new relationships associated with these two contexts. The statements asserting the new relationships become:

T_1: BobDylan isMarriedTo#1 SaraLownds, and
T_5: BobDylan isMarriedTo#2 SaraLownds.

Obviously, the number of such singleton properties would be as enormous as the number of facts and contexts in any real RDF datasets. We need to provide a mechanism to cluster them into groups for higher level abstraction. Such a mechanism allows us to group similar singleton properties into a more general one. We observe that, although statements are fundamentally distinguishable based on their context, they do share common characteristics in their nature which are captured by generic properties. The relationship between singleton and generic properties can be considered from two different perspectives: the singleton property is either a sub property, or an instance of the generic property.

Sub property. Singleton property can be considered as a specialization, or sub property of a generic property in one particular context. In this case, if we create one singleton property for each fact via `rdfs:subPropertyOf`, the number of singleton property nodes below the generic property in the property hierarchy would become enormous. For example, YAGO has 23,770 facts[1] using the property `isMarriedTo`. A schema with such a large amount of child nodes in the property hierarchy of `rdfs:subPropertyOf` is not desirable.

Property instance. In this view, while generic properties are properties whose extension contains a set of entity pairs, each singleton property is unique to one particular entity pair. Intuitively, we can consider singleton properties as instances of generic properties. In that sense, a singleton property is interconnected to its generic property via `rdf:type`. However, the property `rdf:type` as a generic property may also have its own instances. For example, since YAGO contains 9,019,948 facts using `rdf:type`, a triple like this may cause ambiguity: `type#1 rdf:type rdf:type`.

Considering the nature of the relationship from both perspectives, we invent a new property, `singletonPropertyOf` to connect singleton properties with their generic property. The extension of a generic property contains the set of singleton property instances created in all contexts. In the example described in Table 3, we use `singletonPropertyOf` in both T_2 and T_6.

T_2: isMarriedTo#1 singletonPropertyOf isMarriedTo
T_6: isMarriedTo#2 singletonPropertyOf isMarriedTo.

[1]http://www.mpi-inf.mpg.de/yago-naga/yago/statistics.html

We will provide further explanations of how singleton properties can be interpreted in RDF and RDFS in Section 3.

2.2 Asserting meta knowledge for triples

Here we demonstrate how to assert metadata for a statement, such as provenance, time, location, or certainty. Please note that we are not attempting to model complex contextual information involving these four dimensions for a statement because context modeling is out of the scope of this paper. We also note that meta properties used in the examples such as `hasSource, extractedOn, hasStart, hasEnd`, `tookPlaceAt`, and `hasScore`, are only for demonstration. While adopting this approach, one may want to use vocabularies of meta knowledge recommended by W3C such as PROV [10] or OWLTime [7] for enhancing the interoperability with other Semantic Web knowledge bases and applications.

Provenance. Provenance of a statement explains the origin of that statement [13, 18]. It includes many kinds of metadata for answering questions such as the ones listed in Table 1. For example, the triple T_1 and T_2 are extracted from the Wiki page of Bob Dylan and Sara Lownds. We can assert the provenance of two triples using the properties `hasSource` and `extractedOn` as follows:

T_3:	isMarriedTo#1	hasSource	wk:Bob_Dylan
T_4:	isMarriedTo#1	extractedOn	2009-06-07
T_7:	isMarriedTo#2	hasSource	wk:Sara_Dylan
T_8:	isMarriedTo#2	extractedOn	2009-08-08

Time. The validity of a statement may be associated with a specific time or a time span. For example, a person is born at one specific time, and a marriage between two persons may last for one period of time. Here we represent the time span of the marriage between Bob Dylan and Sara Lownds using `hasStart` and `hasEnd`:

isMarriedTo#1 hasStart 1965-11-22 .
isMarriedTo#1 hasEnd 1977-06-29 .

Location. A statement may be associated with a spatial dimension. For example, the Wiki page of Sara Lownds stated that the marriage of Bob Dylan and Sara Lownds took place at Mineola, Long Island. We assert this meta knowledge for `isMarriedTo#2` as follows:

isMarriedTo#2 tookPlaceAt Mineola .

Certainty. The certainty of a statement reflects the confidence of the authors while creating that statement. For example, if the confidence score of the tool extracting the statement T_2 is 0.7, we can represent it as follows:

isMarriedTo#2 hasScore 0.7 .

From the assertions created for provenance, time, location and certainty above, we observe that they share the same triple pattern, which is *singleton-property meta-property meta-value.* In our example, meta properties are `hasStart`, `hasEnd`, `hasSource`, `hasScore`, and `tookPlaceAt`. We can generalize this pattern for representing all dimensions of meta knowledge as follows.

Singleton Graph Pattern. In general, given a fact (s, p, o), let `p#i` be the singleton property representing this fact in one particular context, `mp#j` be the meta property, `mv#j` be the value of meta property, the set of triples forming a singleton graph pattern asserting meta knowledge for this fact is provided in Table 4. We will use this singleton graph pattern for querying meta knowledge in Section 4.

Table 4: Singleton graph pattern asserting meta knowledge for data triple (s,p,o)

Subject	Predicate	Object
p#i	singletonPropertyOf	p
s	*p#i*	*o*
p#i	*mp#j*	*mv#j*

2.3 Enforcing the singleton-ness of property instances

If the property `isMarriedTo#1` is asserted another triple such as `BarackObama isMarriedTo#1 MichelleObama`, this together with the existing assertion `isMarriedTo#1 hasStart 1965-11-22` would imply the marriage date of the Obamas is 1965-11-22, which is not true. In order to avoid this, we need to ensure the singleton property `isMarriedTo#1` occurs as a property in only one triple.

This constraint has to be enforced for all URIs of singleton property instances. Data publishers may combine their URI prefix, the generic property name and the timestamp when the instance is created into the URI of a singleton property to make it unique. However, there are still cases where two instances may share the same URI. Therefore, data publishers may employ the Universally Unique Identifier (UUID) [9], which is also supported by SPARQL and various programming languages, to ensure the singleton-ness of their property instances. The validation of this uniqueness constraint is straightforward, by counting the number of triple occurrences per singleton property. As the current RDF syntax does not allow blank nodes as properties, we do not represent singleton properties as blank nodes, although one advantage of using blank nodes in the property is providing the completeness for deduction rules [11].

3. MODEL-THEORETIC SEMANTICS

This section explains how the singleton property can fit well with the existing formal semantics. We reuse the model-theoretic semantics described in [6] with three levels of interpretation: simple, RDF and RDFS. For each interpretation we add additional criteria for supporting the singleton property. While we explain the new vocabulary elements in detail, elements without further explanation remain as they are in the original model-theoretic semantics described in [6].

Given a vocabulary V, the original simple interpretation $\mathcal{I}$ consists of:

- IR, a non-empty set of *resources*, alternatively called domain or universe of discourse of $\mathcal{I}$,
- IP, the set of *generic properties* of $\mathcal{I}$,
- I_{EXT}, a function assigning to each property a set of pairs from IR
  $I_{EXT} : IP \rightarrow 2^{IR \times IR}$ where $I_{EXT}(p)$ is called the *extension* of property p,
- I_S, a function, mapping URIs from V into the union set of IR and IP,
- I_L, a function from the typed literals from V into the set of resources IR,

Table 5: Singleton property approach representing facts and their temporal assertions

Subject	Predicate	Object
BobDylan	isMarriedTo	SaraLownds
BobDylan	isMarriedTo#1	SaraLownds
isMarriedTo#1	rdf:singletonPropertyOf	isMarriedTo
isMarriedTo#1	hasStart	1965-11-22
isMarriedTo#1	hasEnd	1977-06-29
BobDylan	isMarriedTo	CarolDennis
BobDylan	isMarriedTo#2	CarolDennis
isMarriedTo#2	rdf:singletonPropertyOf	isMarriedTo
isMarriedTo#2	hasStart	1986-06-##
isMarriedTo#2	hasEnd	1992-10-##

- LV, a subset of IR, called the set of *literal values*.

We define IPs as a set of singleton properties and $I_{S\_EXT}(p_s)$ as the function mapping a singleton property into a pair of resources.

Simple interpretation of vocabulary V is an original simple interpretation $\mathcal{I}$ of the vocabulary V $\cup$ V_{SIM} that satisfies the additional criteria:

- IPs, called the set of *singleton properties* of $\mathcal{I}$, as a subset of IR,
- $I_{S\_EXT}(p_s)$, the function mapping a singleton property to a pair of resources. $I_{S\_EXT} : IPs \rightarrow IR \times IR$.

Note that the mapping function I_{S_EXT} is not a one-to-one mapping; multiple singleton properties may be mapped to the same pair of entities.

RDF interpretation of a vocabulary V is a simple interpretation $\mathcal{I}$ of the vocabulary V $\cup$ V_{RDF} that satisfies the criteria from the original RDF interpretation and the following criteria:

- $x_s \in IPs$ if $\langle x_s,\, rdf\colon SingletonProperty^{\mathcal{I}}\rangle \in I_{EXT}$ $(rdf\colon type^{\mathcal{I}})$, a singleton property x_s is an instance of class SingletonProperty if they are interconnected by the property rdf:type.
- $x_s \in IPs$ if $\langle x_s, x^{\mathcal{I}}\rangle \in I_{EXT}(rdf\colon singletonPropertyOf^{\mathcal{I}})$, $x \in IP$. A singleton property x_s is an instance of a generic property x if they are interconnected by the property rdf:singletonPropertyOf, where x is called a *generic property.* Since the singletonPropertyOf is defined here, we use rdf:singletonPropertyOf from now on.
- if $x_s \in IPs$ then $\exists!\langle u,v\rangle : \langle u,v\rangle = I_{S\_EXT}({x_s}^{\mathcal{I}})$, and u,v $\in IR$. This enforces the singleton-ness for the property instances.

Given the set of triples with singleton properties and their temporal assertions in Table 5, let V_{EX} be the vocabulary consisting of all the names of subjects, predicates and objects in those triples, the RDF interpretation of the vocabulary V_{EX} is provided in Table 6.

In the RDFS interpretation, we will reuse the function $I_{CEXT} : IR \rightarrow 2^{IR}$ where $I_{CEXT}(y)$ is called (class) extension of y, $I_{CEXT}(y) = \{x \mid \forall x \in IR : \langle x,y\rangle \in I_{EXT}(rdf\colon type^{\mathcal{I}})\}$.

Table 6: RDF interpretation for the vocabulary V_{EX} from Table 5

$I_S =$	BobDylan	$\mapsto \alpha$
	SaraLownds	$\mapsto \beta$
	CarolDennis	$\mapsto \gamma$
	isMarriedTo	$\mapsto \delta$
	isMarriedTo#1	$\mapsto \theta$
	isMarriedTo#2	$\mapsto \lambda$
	hasStart	$\mapsto \sigma$
	hasEnd	$\mapsto \phi$
$IR =$	$\{\alpha,\beta,\gamma,\delta,\theta,\lambda\}$	
$IP =$	$\{\delta,\theta,\lambda,\sigma,\phi\}$	
$LV =$	{1965-11-22, 1977-06-29, 1986-06-##, 1992-10-##}	
$I_{EXT} =$	$\theta \mapsto \{\langle\alpha,\beta\rangle\}$	
	$\lambda \mapsto \{\langle\alpha,\gamma\rangle\}$	
	$\sigma \mapsto \{\langle\theta,$ 1965-11-22 $\rangle,$ $\langle\lambda,$ 1986-06-## $\rangle\}$	
	$\phi \mapsto \{\langle\theta,$ 1977-06-29$\rangle,$ $\langle\lambda,$ 1992-10-## $\rangle\}$	
	rdf:singletonPropertyOf $\mapsto \{\langle\theta,\delta\rangle,\langle\lambda,\delta\rangle\}$	
	$\delta \mapsto \{\langle\alpha,\beta\rangle,\langle\alpha,\gamma\rangle\}$	
$IPs =$	$\{\theta,\lambda\}$	
$I_{S\_EXT} =$	$\theta \mapsto \langle\alpha,\beta\rangle$	
	$\lambda \mapsto \langle\alpha,\gamma\rangle$	

RDFS interpretation of a vocabulary V is an RDF interpretation $\mathcal{I}$ of the vocabulary V $\cup$ V_{RDFS} that satisfies criteria from the original RDFS interpretation and the following criteria:

- $\langle rdf\colon SingletonProperty^{\mathcal{I}}, rdfs\colon Class^{\mathcal{I}}\rangle \in I_{EXT}$ $(rdf\colon type^{\mathcal{I}})$.
  rdf:SingletonProperty is defined as a class. The extension of rdf:SingletonProperty is the set IPs of all singleton properties, or
  $IPs = I_{CEXT}(rdf : SingletonProperty^{\mathcal{I}})$.
- $\langle rdf\colon SingletonProperty^{\mathcal{I}}, rdfs\colon Resource^{\mathcal{I}}\rangle \in I_{EXT}$ $(rdfs\colon subClassOf^{\mathcal{I}})$, this causes $IPs \subset IR$,
  every singleton property is an RDF resource.
- if $\langle x_s,\ x\rangle \in I_{EXT}(rdf\colon singletonPropertyOf^{\mathcal{I}})$, $x_s \in IPs$, and $x \in IP$, then $I_{S\_EXT}(x_s) \in I_{EXT}(x)$. $I_{EXT}(x)$ is called *property extension* of the generic property x. The set of singleton properties connected to that property via rdf:singletonPropertyOf is a sub set of the property extension of its generic property.
- Let $\langle x_s, x\rangle \in I_{EXT}(rdf\colon singletonPropertyOf^{\mathcal{I}})$, and $\langle x,y\rangle \in I_{EXT}(rdfs\colon domain)$, if $\langle u,v\rangle \in I_{S\_EXT}(x_s)$, then $u \in I_{CEXT}(y)$ where $I_{CEXT}(y)$ is the class extension of y. A singleton property shares domain with its generic property.
- Let $\langle x_s, x\rangle \in I_{EXT}(rdf\colon singletonPropertyOf^{\mathcal{I}})$, and $\langle x,y\rangle \in I_{EXT}(rdfs\colon range^{\mathcal{I}})$, if $\langle u,v\rangle = I_{S\_EXT}(x_s)$, then $v \in I_{CEXT}(y)$. A singleton property also shares range with its generic property.

4. QUERYING

In Section 2, we described the singleton graph pattern for representing meta knowledge. This section explains the principle for querying such meta knowledge based on that singleton graph pattern. We will use the example from Table 5 for demonstrating how we query meta knowledge.

Since all these triples in singleton graph patterns are represented in RDF, they can be queried using any RDF query language. Here we consider SPARQL as it is recommended for querying RDF by W3C [4].

In principle we can distinguish two basic types of query patterns: data vs. metadata. Both query patterns can be constructed as a graph pattern in the SPARQL queries.

Data query contains graph patterns created from a set of factual data triples in the form of (s, p, o) by replacing any of subject s, property p or object o with variables. For example, given two data triples of Table 5,

BobDylan	isMarriedTo	SaraLownds
BobDylan	isMarriedTo	CarolDennis

we can construct a data query asking for spouses of BobDylan as follows:

```
SELECT ?obj WHERE { BobDylan isMarriedTo ?obj}
```

This query type is commonly used in Semantic Web applications. Using the singleton property approach for representing meta knowledge of data triple, we can also easily query such meta knowledge.

Metadata query contains graph patterns created from a set of triples in the singleton graph pattern by replacing any subject, property and object of any triple with variables. One sample of a meta query pattern asking for the meta values of any meta property `mp` could be:

```
SELECT ?mp ?mv
WHERE {?pi rdf:singletonPropertyOf p .
s ?pi o . ?pi ?mp ?mv . }
```

The query instance asking for the dates of BobDylan's marriages is as follows.

```
SELECT ?startOrEnd ?dates
WHERE {?pi rdf:singletonPropertyOf isMarriedTo .
BobDylan ?pi ?o . ?pi ?startOrEnd ?dates . }
```

In practice, one may mix data patterns and metadata patterns into one query pattern for more complicated queries. The next section will provide more queries of different kinds of metadata in detail.

5. USING SINGLETON PROPERTY IN EXISTING KNOWLEGE BASES

5.1 BKR and Provenance

The Biomedical Knowledge Repository (BKR) is an extensive knowledge base that integrates biomedical knowledge from multiple sources while tracking their provenance using a unified provenance framework [15, 16]. A triple in the BKR may be extracted from PubMed articles and is associated with a confidence score from its extraction tool. Given a triple (s, p, o) extracted from PMID#1 with confidence score 0.3, from PMID#2 with confidence score 0.8, the current representation of provenance of a triple with PaCE [16] is provided in Table 7, note that the confidence score cannot be represented by this approach.

The basic idea of PaCE is to create one instance of subject, property, and object per context, and asserting the source of those instances. PaCE offered three flavors: minimalist (C1), intermediate (C2) and exhaustive (C3). The source of the triple is inferred from the common source of subject (C1), subject - property (C2), and subject - property - object (C3) instances. Among the three flavors, C1 was proven to be better than the reification approach in terms of number of triples and query performance in [16]. However, this approach is limited in supporting different dimensions of meta knowledge because it can only represent the source of a triple. Here we have at least two metadata associated with a triple, but it can only represent the source. For instance, if there exists another triple (s, p', o') with a different confidence score 0.2 extracted from the PMID#1, then this score cannot be represented correctly. Since (s, p, o) and (s, p', o') are from the same PMID#1, the instance s_PMID#1 representing both triples in the PMID#1 is used to assert the meta property hasScore: `s_PMID#1 hasScore 0.3`. This automatically infers the confidence score of (s, p', o') in PMID#1 is 0.3, which is incorrect because its score is 0.2.

Table 7: PaCE approach for (s, p, o) with meta knowledge (PMID#1, 0.3) and (PMID#2, 0.8)

Subject	Property	Object
s_PMID#1	rdf:type	s
p_PMID#1	rdf:type	p
o_PMID#1	rdf:type	o
s_PMID#1	p_PMID#1	o_PMID#1
s_PMID#1	derivedFrom	PMID#1
p_PMID#1	derivedFrom	PMID#1
o_PMID#1	derivedFrom	PMID#1
s_PMID#2	rdf:type	s
p_PMID#2	rdf:type	p
o_PMID#2	rdf:type	o
s_PMID#2	p_PMID#2	o_PMID#2
s_PMID#2	derivedFrom	PMID#2
p_PMID#2	derivedFrom	PMID#2
o_PMID#2	derivedFrom	PMID#2

Table 8: Singleton Property approach for (s, p, o) with meta knowledge (PMID#1, 0.3) and (PMID#2, 0.8)

Subject	Property	Object
p#1	rdf:singletonPropertyOf	p
s	p#1	o
p#1	derivedFrom	PMID#1
p#1	hasScore	0.3
p#2	rdf:singletonPropertyOf	p
s	p#2	o
p#2	derivedFrom	PMID#2
p#2	hasScore	0.8

Using the Singleton Property approach, we can represent the complete metadata information as provided in Table 8. Moreover, if we need to represent more meta knowledge dimensions for the triple (s, p, o), we can simply add assertions into the singleton properties `p#1` and `p#2`.

Provenance query. Since the BKR integrates data from multiple sources, it is common to ask about the provenance of a triple, such as the sources, the publication date, the confidence score, etc. For example, one may query the sources of a triple that has a high confidence score (above 0.7). This

Table 9: Overall statistics of the SP-YAGO2 dataset

Number of triples	292,166,376
Number of generic properties	83
Number of singleton properties	62,643,969

Table 10: Sample meta properties in SP-YAGO2 including temporal, spatial and provenance

Generic property	# of singleton properties
extractionSource	32,598,374
isLocatedIn	1,262,563
hasLongitude	393,717
hasLatitude	393,250
occursSince	553,116
occursUntil	337,116
wasBornOnDate	804,816

query cannot be supported by PaCE approach because the confidence score is not present. Using the Singleton Property approach, and adopting the metadata query discussed in Section 4, we can create a query like the following:

```
SELECT ?source ?score
WHERE {s ?pi o . ?pi rdf:singletonPropertyOf p .
?pi derivedFrom ?source . ?pi hasScore ?score .
FILTER (?score > 0.7) }
```

In the next section, we provide a more thorough comparison in the performance of the singleton property approach versus existing approaches.

5.2 YAGO2 and Temporal-Spatial Enhancement

While YAGO [20] provides an extensive collection of factual triples extracted from Wiki and other sources, YAGO2 [8] enhances this knowledge base with temporal and spatial information for those factual triples. This knowledge base becomes aware of times and places and, hence, is capable of answering more complex queries involving such metadata.

Here we reuse the example from [8] to demonstrate the requirements of representing meta knowledge in YAGO2. We put the set of facts from the example into Table 11. YAGO2 uses fact identifiers to represent the facts, and asserts the occurring time and place of the facts by using their fact identifiers as subjects of the meta assertions. It also provides a SPARQL-like query language which allows it to incorporate fact identifiers in the query pattern. Here we propose to replace the fact identifier by the singleton property in representing a statement and asserting its temporal and spatial information. This would enable the interoperability between this dataset with other RDF datasets and allow them to be queried using standard query language. This RDF representation is compatible with existing RDF datasets and interoperable with other Semantic Web applications that use existing standards such as SPARQL. We do not attempt to compare the query performance or expressiveness between SPARQL and SPARQL-like language used in YAGO2.

Table 11: YAGO2 uses fact ID for representing fact and asserting meta knowledge

Id	Subject	Predicate	Object
`#1`	GratefulDead	performed	TheClosingOfWinterland
`#2`	`#1`	occursIn	SanFrancisco
`#3`	`#1`	occursOn	1978-12-31

Table 12: Singleton property replaces fact ID in asserting meta knowledge

Subject	Predicate	Object
`performed#1`	singletonPropertyOf	performed
GratefulDead	**performed#1**	TheClosingOfWinterland
`performed#1`	occursIn	SanFrancisco
`performed#1`	occursOnDate	1978-12-31

The YAGO2 is available in the RDF Turtle format [2]. However, the link between the triple identifier and the triple itself doesn't exist. We created a new version of YAGO2 using the singleton property to link the triples and their identifiers. The fact identifiers in commented lines become the property of the fact. The statistics of the SP-YAGO2S version are provided in Table 9 and Table 10.

Temporal-spatial query in the YAGO can be specified in its query language SPARQL-like. For example, for finding concerts that took place near San Francisco, one may create a SPARQL-like query as follows:

```
?id: ?s        performed          ?o .
?id            occursIn           ?l .
?l             hasGeoCoordinates  ?g .
SanFrancisco   hasGeoCoordinates  ?sf .
?g             near               ?sf .
```

We may also create an equivalent SPARQL query using the singleton property approach as follows:

```
?performed#1   rdf:singletonPropertyOf   performed .
?s             ?performed#1              ?o .
?performed#1   occursIn                  ?l .
?l             hasGeoCoordinates         ?g .
SanFrancisco   hasGeoCoordinates         ?sf .
?g             near                      ?sf .
```

Given that `near` is a proximate predicate, it may need to be elaborated in the graph pattern of SPARQL query.

6. EXPERIMENTS

In this section we report the experiments comparing five approaches including the singleton property (denoted by **SP**), standard RDF reification (**R**) and the three flavors of PaCE (**C1**, **C2**, and **C3**). We will repeatly use these important notions in the entire section.

To the best of our knowledge, a benchmarking dataset with SPARQL queries for metadata at triple level is not available. Therefore, in these experiments, we used the BKR dataset previously described in Section 5.1. For evaluating the query performance, we used two set of queries: set A and set B. The set A is obtained from the experiments conducted in [16]. Since all 5 queries of set A include one block of

[2]http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html

provenance-specific triple patterns related to one data triple, one may wonder how the approaches perform with longer queries. Therefore, we created the set B with longer queries, where the lengths of data triple patterns range from 1 to 3. Although the lengths of data triple patterns look small, their corresponding SPARQL query patterns is relatively long, up to 21 triple patterns.

The comparison is based on three quantitative criteria: number of triples, query length and query execution time. Section 6.1 describes the comparison based on the number of triples in the five flavors of the same BKR dataset in detail. Section 6.2 and 6.3 describe the query experiments.

The datasets and queries used in the experiments are provided for reproducing the experiments[3].

6.1 RDF Datasets

In addition to four different RDF datasets from four representation approaches (C1, C2, C3 and R) implemented by Sahoo et al. in [16], we created another dataset SP for our singleton property approach. Instead of reporting the total and provenance-specific triples as in [16], here we analyzed the number of triples in detail based on the triple pattern, whether it is for data triples, meta triples or statement handling triples. This analysis would be useful for understanding how each type of triples would contribute to the total number of triples when the data input increases.

We classified all triples into three main categories: data triples, metadata triples and triple handlers.

Data triples are original triples without any metadata association. The BKR dataset has approximately 23M triples without provenance information. With the provenance information, the number of distinct data triples is 33M because if the same triple occurs in two different articles, it is counted as two data triples. Therefore, we have 33M data triples in the BKR. We denote n = 33M for later use in the total number of triples for each dataset.

Triple handlers are created to represent data triples as individual resources. Particularly, they are statement instances reified by four triple patterns from R, singleton properties from SP, subject instances from C1, subject-property instances from C2, and subject-property-object instances for C3. While the R approach needs 4n = 112M triples to represent statement instances, the SP approach needs only n = 33M triples. The C1 approach needs only 22M triples because it contains duplicate subject instances in the same source. In the worst case, C1 approach would need n triples if all the triples do not share any metadata values.

Metadata triples are additional triples created by each approach in order to attach metadata into triple handlers. Both R and SP need n = 33M triples for this category because one meta property is asserted for each singleton property and statement instance. Again, in the case of C1, within 22M triples representing triple handlers, only 16M subject instances were asserted the derives_from information; the remain 6M are for declaring the type of subject instances.

We present the number of triples of each category in Figure 1. The total number of triples in SP and R datasets is 3n and 6n, respectively. The sizes of C1, C2 and C3 are application-specific and do not depend only on the number of data triples as the SP and R.

Discussion. The size of SP dataset is half of the size of R dataset and is relatively comparable with C2 and C3. The C1 dataset is approximately 30% smaller than the SP dataset due to duplicated triples. However, if the BKR is extended to support provenance at a finer-grain level, such as at sentence level, this C1 approach would lose the advantage and its size will become the same as the size of SP.

[3]Datasets and queries are vailable at http://wiki.knoesis.org/index.php/Singleton_Property

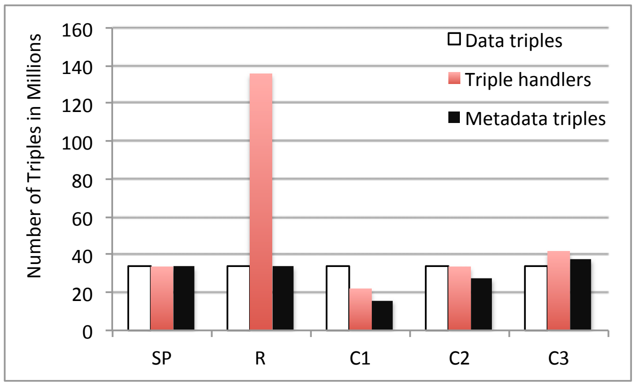

Figure 1: Number of triples in million of 3 categories contributing to the total number of triples.

6.2 Query Set A

In this experiment, we repeated the query experiments performed in [16]. We reused the sets of queries in [16] for evaluating the performance among four representation approaches (C1, C2, C3, and R). In order to compare the performance of these four approaches with our singleton property approach, we created one more equivalent set of queries SP. Therefore, the set A have 5 sets of queries in total. We used Virtuoso Open Source 6.1.7 on a Linux server with 8GB RAM for this experiment. Each query run starts with a cold cache. The set of queries are run in two phases.

In the first phase, each query is evaluated for fixed values. Figure 2 presents the average of the last 5 of a total of 20 runs. In the second phase, each query is executed with a set of 100 random values. The set of those 100 queries are run in 5 times and Figure 3 presents the average of 100 queries in the last run. We eliminate the execution times of Q1 because they are too small (less than 1 msec) to be readable in the two charts.

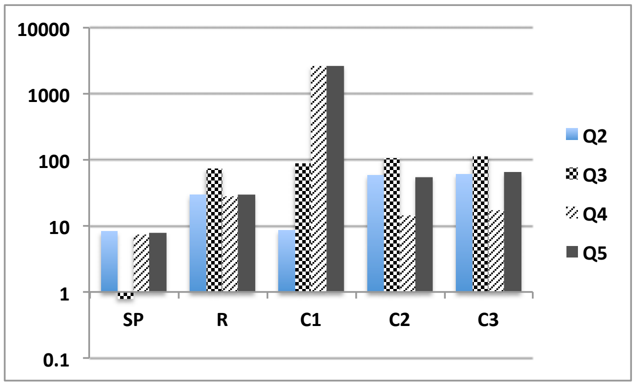

Figure 2: Query performance in msec.: fixed values.

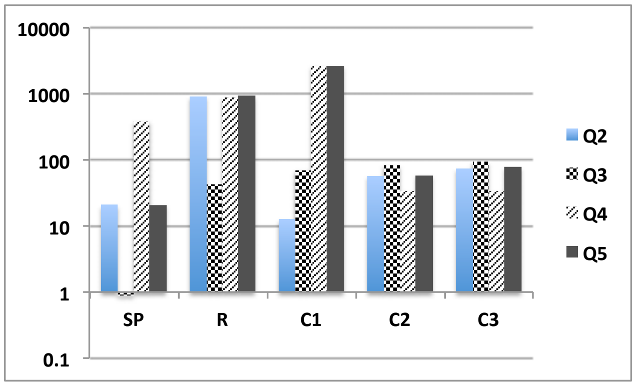

Figure 3: Query performance in msec.: 100 values.

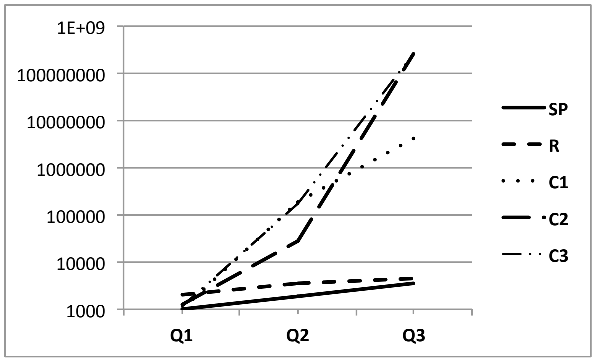

Figure 5: Query performance in msec. of the set B.

Discussion. For the set of fixed values in phase 1, Figure 2 shows that all the SP queries are the fastest ones. For the set of 100 random values, Figure 3 shows that SP queries are faster than all others in Q3 and Q5, and also faster than two approaches in Q2 and Q4. Therefore, we can conclude that for most of the queries in this experiment, the SP queries give better query performance than other approaches.

6.3 Query set B

In this experiment, we created a set of queries of varying path lengths. Particularly, we created three queries (Q1, Q2, and Q3), each query contains a path of 1, 2, and 3 data triple patterns respectively. After incorporating the metadata triples involving the source into their SPARQL queries, the total number of triple patterns become varying among the five representation approaches. The sizes of their corresponding SPARQL queries are presented in Figure 4.

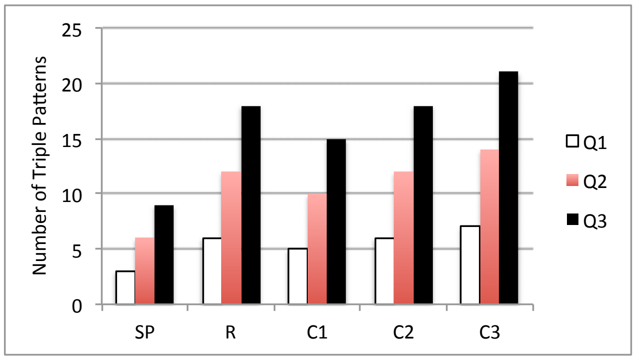

Figure 4: Number of triple patterns within three queries Q1, Q2, and Q3 of query set B.

Since each triple pattern is translated to one query join operator in the query plan, queries with shorter patterns tend to be executed faster. Among all the approaches, the SP queries are the shortest one. Therefore, we expect the SP queries perform better than others in terms of query execution time. We ran the set of queries of each approach in 3 times in cold cache and reported the average execution time in Figure 5. This experiment was performed on a Ubuntu 12.04 desktop with 4GB of RAM.

Discussion. While all the queries in the set A are executed in seconds or minutes, some of the queries in the set B take longer time. Particularly, the longest query of the C1 approach with 18 patterns is in hours, and that of the C2 and C3 approaches with 20 and 21 patterns is in days because of full index lookups in their query plan. On the other hand, the queries in the two approaches, SP and R, still remain being executed in seconds, and the SP queries are little faster than the R queries.

6.4 Overall Discussion

Our experiments show that the SP approach gives a decent performance in terms of number of triples, query size and query execution time. Here we do not conclude that our approach is better than other approaches in all the cases. For the number of triples, the C1 approach is the most compact one in the case of BKR where multiple predications share the same source. However, this C1 approach will have the same size with the SP approach in the cases where the data triples do not share metadata values, such as at a finer-grained level of provenance (e.g. statement level instead of article level), or discrete values for the temporal, spatial and certainty properties. For the query performance, the SP queries give the best performance, which is expected and consistent with the query length comparison. Since only default indexes were created, and no optimization was provided, this leaves a room for query optimization in order to obtain a better query performance.

7. RELATED WORK

Many approaches have been proposed to address the problem of representing and querying statements about statements. We can divide these approaches into three main categories: triples [15, 16], quadruples [1, 3, 19] and quintuples [17] based on the number of elements in the structure each approach employs. Each approach reflects one perspective on how meta knowledge for triples could add elements into tuples. We will discuss about the contribution of each approach, and how our approach fits into the scheme.

Triples. Representing different dimensions of meta knowledge for triples using RDF triples in order to retain the compatibility and interoperability with existing Semantic Web knowledge bases, tools, languages and methods is the main goal of this kind of approach. The reification approach [12, 5] allows meta knowledge to be asserted from reified statements. The singleton property approach differs from the reification in that it provides a formal semantics. Moreover, it requires one triple for creating a singleton property while a reified statement requires four triples. That would make it more efficient because of smaller number of triples, shorter query patterns, and thereby smaller number of joins in query processing.

Sahoo et al. [16] propose the PaCE approach for representing the provenance of a triple by creating different instances for its subject, property and object for different contexts and asserting provenance for those instances. The source of the triple is derived from the source of its individual components. The singleton property approach is similar to PaCE in that it creates different instances for capturing different contexts. However, the main difference between the two approaches has to do with which instances are created for each context. We ground our approach on shifting the focus from entities to properties. That is, within a new context, a new relationship is established between two already existing entities.

Quadruples. In the reification approach, we need to create statement instances and indicate the subject, property, and object for those instances. Intuitively, this verbosity can be avoided by adding one more element into a triple to make it a quadruple. Named graph [1] and other work on top of named graph such as [3, 14] follow the approach, using the fourth component to represent the provenance of a set of triples. Although technically we can restrict a named graph to a single triple and use it to assert meta knowledge to that triple, it does not naturally serve this purpose because originally the named graph is designed for representing provenance and trust of a set of triples. On the other hand, the singleton property approach is complementary to the named graph approach in representing provenance for different granularity levels of triples: one is for an individual triple, the other one is for a set of triples. Data publishers may choose the approach that fulfills specific requirements of their applications.

Straccia et al. [19] also annotate the meta knowledge such as temporal and certainty for RDF triples. We classify this approach into quadruples because it annotates every RDF triple with an annotation term. It proposes a new algebraic structure with well-defined operators for manipulating meta information. This approach is followed up with the RDFS reasoning supported by [2]. Our approach differs from this approach in that we leverage RDF triples for the representation of meta knowledge assertions, allowing them to be queried and entailed using existing languages and tools, while this approach does not.

Quintuples. The RDF^+ approach [17] defines the abstract syntax of RDF^+ statement as a quintuple by extending the named graph quad with a statement identifier. The statement identifier is used as the subject of the meta knowledge assertion, which is an RDF triple. Since the formal semantics is defined in RDF^+, mappings from RDF to RDF^+ and vice versa have to be made. Additionally, the SPARQL syntax and semantics have to be extended to support querying RDF^+. The singleton property approach differs from the RDF^+ approach in two main design points. First, while a statement identifer is defined in the RDF^+ statement which is a quintuple, our approach represents singleton property instances in RDF triples. As a result, our approach does not need any mapping while the RDF^+ does. Secondly, our approach does not require any extension to the syntax or semantics of SPARQL because it is completely compatible with SPARQL.

8. DISCUSSION AND FUTURE WORK

Here we discuss some of the possible future work related to the singleton property approach.

OWL Compatibility. In previous sections, we have shown the compatiblity of the singleton property with RDF, RDFS and SPARQL. The question of whether or not an RDF dataset with singleton properties is compatible with OWL also arises, so we devised and conducted a small experiment for testing this. Since BKR, YAGO2S, and any other datasets are comprised of a number of singleton graph patterns, we used one example from the BKR to construct a set of RDF triples by adding class memberships for all RDF resources. We validated this set of triples against all OWL 2 profiles[4], and we found that they are compatible. This initial experiment encouraged us to study and apply the singleton property in the management of metadata for ontologies such as the Gene Ontology.

Deduction rules. Since a singleton property is an instance of a generic property, intuitively the subsumption rule may be applicable to this relationship. More of these deduction rules for dealing with reasoning in the generic property hierarchy may be desirable. In addition to syntactic rules, we may also study the domain-specific rules for inferring new triples using provenance, temporal or spatial information.

Meta query optimization. Our experiments were carried out with Virtuoso RDBMS, certain optimization techniques for relational databases can also be applied to obtain better query performance. For example, since the meta query has certain singleton graph patterns, creating indexes or views of those patterns might help.

9. CONCLUSION

We have presented our approach for representing and querying meta knowledge using the singleton property. Regular RDF properties are viewed as generic properties in our approach, and the set of singleton properties are viewed as instances of those generic properties. Both singleton properties and meta knowledge assertions are represented using RDF syntax, ensuring their compatibility with existing RDF knowledge bases. The meaning of such a singleton property is defined in the formal semantics that is extended from the current model-theoretic semantics in all three steps of interpretation: simple, RDF, and RDFS. This singleton property approach also fits nicely the syntax of SPARQL query language. Because of those, we are able to demonstrate how this approach can be easily used for representing and querying meta triples, and indeed, we implemented it in two existing knowledge bases, BKR and YAGO2.

Therefore, adopting this approach in representing, querying, and sharing knowledge bases that are aware of meta knowledge would allow those knowledge bases to be compatible with a wide range of Semantic Web languages, tools, and methods.

Acknowledgement. This research was supported by the Intramural Research Program at NLM, NIH. We thank Jagannathan Srinivasan, Jayanta Banerjee, Satya Sahoo, Wenbo Wang, Lu Chen, Pavan Kapanipathi, Franklin Jose and especially anonymous reviewers for excellent feedback.

[4]http://owl.cs.manchester.ac.uk/validator/

10. REFERENCES

[1] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM, 2005.

[2] C. V. Damásio and F. Ferreira. Practical RDF Schema Reasoning with Annotated Semantic Web Data. In *The Semantic Web–ISWC 2011*, pages 746–761. Springer, 2011.

[3] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring RDF Triples to Capture Provenance. In *The Semantic Web-ISWC 2009*, pages 196–212. Springer, 2009.

[4] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. *W3C Working Draft*, 12 May 2011.

[5] P. Hayes and B. McBride. RDF Semantics. *W3C Recommendation*, 10 February 2004.

[6] P. Hitzler, M. Krotzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC, 2011.

[7] J. R. Hobbs and F. Pan. Time Ontology in OWL. *W3C Working Draft*, 27 September 2006.

[8] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: a Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 2012.

[9] P. J. Leach, M. Mealling, and R. Salz. A Universally Unique Identifier (UUID) URN Namespace. 2005.

[10] T. Lebo, S. Sahoo, and D. McGuinness. Prov-o: The PROV Ontology. *W3C Working Draft*, 03 May 2012.

[11] A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On Blank Nodes. In *The Semantic Web–ISWC 2011*, pages 421–437. Springer, 2011.

[12] F. Manola, E. Miller, and B. McBride. RDF Primer. *W3C Recommendation*, 10 February 2004.

[13] L. Moreau. The Foundations for Provenance on the Web. *Foundations and Trends in Web Science*, 2(2–3):99–241, 2010.

[14] P. Pediaditis, G. Flouris, I. Fundulaki, and V. Christophides. On explicit provenance management in rdf/s graphs. *TAPP*, 9:1–10, 2009.

[15] S. Sahoo, V. Nguyen, O. Bodenreider, P. Parikh, T. Minning, and A. Sheth. A Unified Framework for Managing Provenance Information in Translational Research. *BMC Bioinformatics*, 2011.

[16] S. S. Sahoo, O. Bodenreider, P. Hitzler, A. Sheth, and K. Thirunarayan. Provenance Context Entity (PaCE): Scalable Provenance Tracking for Scientific RDF Data. SSDBM'10, pages 461–470, 2010.

[17] B. Schueler, S. Sizov, S. Staab, and D. T. Tran. Querying for Meta Knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 625–634. ACM, 2008.

[18] Y. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in e-Science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[19] U. Straccia, N. Lopes, G. Lukacsy, and A. Polleres. A General Framework for Representing and Reasoning with Annotated Semantic Web Data. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), AAAI Press*, volume 28, 2010.

[20] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.