Andreas Harth, Katja Hose, Ralf Schenkel

Linked Data Management: Principles and Techniques

Contents

Chapter 1

On the Use of Abstract Models for RDF/S Provenance

Irini Fundulaki

FORTH-ICS, Greece

Giorgos Flouris

FORTH-ICS, Greece

Vassilis Papakonstantinou

FORTH-ICS, Greece

1.1 Introduction

Provenance refers to the origin of information and is used to describe *where* and *how* the data was obtained. Provenance is versatile and could include various types of information, such as the source of the data, information on

the processes that led to a certain result, date of creation or last modification, authorship, and others.

Recording and managing the provenance of data is of paramount importance, as it allows supporting trust mechanisms, access control and privacy policies, digital rights management, quality management and assessment, in addition to reputability, reliability, accountability of sources and datasets. In this respect, provenance is not an end in itself, but a means towards answering a number of questions concerning data and processes. It has been argued that provenance of data is sometimes more important than the data itself [14].

The absence, or non-consideration, of provenance can cause several problems; interesting examples of such *provenance failures* can be found in [10]. One such case was the publication, in 2008, of an undated document regarding the near bankruptcy of a well-known airline company; even though the document was 6 years old, and thus irrelevant at the time, the absence of a date in the document caused panic, and the company's share price fell by 75% [10].

Provenance becomes even more important in the context of the Linked Open Data (LOD)[1] initiative, which promotes the free publication and interlinking of large datasets in the Semantic Web [4, 3]. The Linked Open Data cloud is experiencing rapid growth since its conception in 2007; hundreds of interlinked datasets compose a knowledge space which currently consists of more than 31 billion RDF triples.

The unconstrained publication, use and interlinking of datasets that is encouraged by the LOD initiative is both a blessing and a curse. On the one hand, it increases the added-value of interlinked datasets by allowing the reuse of concepts and properties. On the other hand, the unmoderated data publication makes the need for clear and efficient recording of provenance even more imperative for resolving problems related to data quality, data trustworthiness, privacy, digital rights etc [16, 22].

In this Chapter, we focus on the problem of representing and storing provenance in a LOD setting. The main problem here is not the storage per se, but the handling of provenance for inferred information (i.e., in the presence of RDFS or custom inference [34] rules), as well as the efficient management of provenance information during updates. In particular, we discuss how *abstract provenance representation models* can be used to address these problems, and quantify the merits and drawbacks of this approach.

More specifically, in Section 1.2, we informally describe two alternative representations for provenance information, which are explained more formally in Section 1.3. The algorithms for representing, accessing and updating provenance information are given in Section 1.4; some implementation details are given in Section 1.5 and Section 1.6 contains the evaluation of the approach. We conclude in Section 1.7. We assume reader's familiarity with the basic ideas underlying the Semantic Web [4, 3], RDF/S syntax and semantics [27, 6], in-

[1] `http://linkeddata.org/`

cluding RDFS inference [6], and SPARQL [32, 31]; an introduction to these topics can be found in Chapter **??**.

1.2 Representation Models for Provenance

Provenance of data is of course data itself that needs to be adequately represented and stored. In this section, we will describe two representation models for provenance information, namely *concrete* and *abstract* ones, and explain the merits and drawbacks of each.

To explain the two approaches, we will use an example in which provenance is being used in the context of a trust assessment application, where various linked sources are composed to form a linked dataset, and each source is associated with a trust value. Note that, in principle, it could be the case that a given source is associated with several trust values; for example, even though a sports web site could publish information on sports, but also a weather forecast, the former should be more trusted than the latter. This more complicated case can be similarly supported by assuming a richer (i.e., more fine-grained) set of sources, i.e., treating the different information as coming from different sources. In the following, for reasons of simplicity and without loss of generality, we assume the simple case where each source is associated with a single trust value. In Section 1.3 we drop this assumption. Moreover, it should be noted that, even though our running example is about trust assessment, most of our observations in this Chapter hold for any application of provenance, unless mentioned otherwise.

1.2.1 Concrete Representation Models

Concrete representation models are actually straightforward and consist in associating each RDF triple with an *annotation tag* that describes its provenance. This essentially transforms a triple into a *quadruple*, the fourth field being the triple's provenance (which associates the triple with its source in a linked data setting). The most common method for the representation of quadruples is through named graphs [8]. Table 1.1 shows a simple example dataset, inspired by the FOAF ontology[2], that we will use as a running example for this Chapter. Each triple in the dataset is associated with its source, forming a quadruple. Note that namespaces are omitted from URIs for readability purposes.

Implicit information raises some complications, because it may be inferrable from triples coming from different sources [14]. For example, the triple $(\&a, type, Person)$ is implied from $(\&a, type, Student)$ and

[2] `http://www.foaf-project.org/`

subject	*property*	*object*	*source*
Student	*subClassOf*	*Person*	s_1
Person	*subClassOf*	*Agent*	s_2
&*a*	*type*	*Student*	s_3
&*a*	*firstName*	`Alice`	s_4
&*a*	*lastName*	`Smith`	s_4
Person	*subClassOf*	*Agent*	s_1

TABLE 1.1: Concrete Provenance Representation of Explicit Triples

(*Student*, *subClassOf*, *Person*), whose provenance is s_3 and s_1 respectively. Thus, the provenance of (&*a*, *type*, *Person*) is the *combination* of s_3 and s_1. This is different from the case where a triple originates from different sources, in which case it has two *distinct* provenance tags [14], as, e.g., is the case with (*Person*, *subClassOf*, *Agent*), which originates from s_2 and s_1 (cf. Table 1.1). To address this problem, concrete policies associate specific semantics to the two types of provenance combination (combining provenance under inference, or combining provenance in multiply-tagged triples).

A similar problem appears when one "constructs" a triple via a SPARQL query [32, 31]. In that case, the provenance of the constructed triple is the combination of the provenance of the triples that are used to obtain said triple. Note that this is a more complicated case, because, unlike inference, construction in SPARQL queries uses different operators (union, join, projection, selection etc). Depending on the provenance model considered, the employed query operators are recorded. Various works have considered this problem [19] (also in the relational setting, e.g., [39, 5]), which is especially difficult when dealing with the non-monotonic fragment of SPARQL [18, 11, 35]. In this Chapter, we focus on inference only and ignore the more complicated SPARQL case; even though the solutions for the simple version of the problem do not directly apply for the more complicated one, the general ideas can be used in both cases.

We will explain how the interrelationship between inference and provenance works in an example, considering a trust assessment application. This application associates each of the four sources $(s_1, \ldots, s_4)$ in our example, with a trust level annotation value in the range $0 \ldots 1$ (where 0 stands for "fully untrustworthy" and 1 stands for "fully trustworthy"). Let's say that s_1, s_2, s_3, s_4 are associated with the values $0,9$, $0,6$, $0,3$ and $0,1$ respectively. Let us also assume that the trustworthiness of an implied triple is equal to the minimum trustworthiness of its implying ones. When a triple originates from several different sources, its trustworthiness is equal to the maximum trustworthiness of all such sources.

This semantics would lead to the dataset shown in Table 1.2, where the last column represents the trustworthiness level of each

subject	*property*	*object*	*trust level*
Student	*subClassOf*	*Person*	$0,9$
Person	*subClassOf*	*Agent*	$0,9$
&*a*	*type*	*Student*	$0,3$
&*a*	*firstName*	`Alice`	$0,1$
&*a*	*lastName*	`Smith`	$0,1$
Student	*subClassOf*	*Agent*	$0,9$
&*a*	*type*	*Person*	$0,3$
&*a*	*type*	*Agent*	$0,3$

TABLE 1.2: Trustworthiness Assessment (Concrete Representation)

quadruple. The first five quadruples are explicit ones, and result directly from the quadruples in Table 1.1, whereas the rest are implicit. For example, $(Person, subClassOf, Agent)$ originates from both s_1 and s_2 (see Table 1.1) so its trustworthiness equals the maximum of $0,6$ and $0,9$. Similarly, the implicit triple $(Student, subClassOf, Agent)$ is implied by $(Student, subClassOf, Person)$ (whose provenance is s_1) and $(Person, subClassOf, Agent)$ (that originates from two sources and hence has provenance s_1 and s_2). The pair (s_1, s_1) results to a trustworthiness of $0,9$, whereas the pair (s_1, s_2) would give trustworthiness $0,6$; thus, the final trustworthiness level of $(Student, subClassOf, Agent)$ is $0,9$ (the maximum of the two, per our semantics). Using similar arguments, one can compute the trustworthiness levels of the other triples, as shown in Table 1.2.

In summary, annotating triples under a concrete model amounts to computing the trust level of each triple, according to its provenance and the considered semantics, and associating each with its (explicit or computed) trust value. The annotation semantics differ depending on what this annotation is representing (e.g., trust, fuzzy truth value etc [19]) and the application context. Concrete provenance models have been used for applications such as access control [1, 12, 24, 26, 29, 33, 40] and trust [7, 21, 38] among others.

1.2.2 Abstract Representation Models

Despite its simplicity and efficiency, concrete models have drawbacks when used with dynamic data. For example, assume that the linked dataset corresponding to source s_1 is updated and no longer includes triple $(Person, subClassOf, Agent)$. Formally, this amounts to the deletion of the quadruple $(Person, subClassOf, Agent, s_1)$. This deletion affects the trustworthiness of $(Student, subClassOf, Agent)$ in Table 1.2, but since concrete models do not record the computation steps that led to each annotation value, there is no way to know which annotations are affected, or how they are affected; thus,

	subject	*property*	*object*	*label*	*source*
q_1:	*Student*	*subClassOf*	*Person*	a_1	s_1
q_2:	*Person*	*subClassOf*	*Agent*	a_2	s_2
q_3:	&a	*type*	*Student*	a_3	s_3
q_4:	&a	*firstName*	`Alice`	a_4	s_4
q_5:	&a	*lastName*	`Smith`	a_5	s_4
q_6:	*Person*	*subClassOf*	*Agent*	a_6	s_1
q_7:	*Student*	*subClassOf*	*Agent*	$a_1 \odot a_2$	–
q_8:	*Student*	*subClassOf*	*Agent*	$a_1 \odot a_6$	–
q_9:	&a	*type*	*Person*	$a_3 \odot a_1$	–
q_{10}:	&a	*type*	*Agent*	$a_3 \odot a_1 \odot a_2$	–
q_{11}:	&a	*type*	*Agent*	$a_3 \odot a_1 \odot a_6$	–

TABLE 1.3: Trustworthiness Assessment (Abstract Representation)

we have to recompute all annotations to ensure correctness. Similar problems appear in the case of additions, as well as when the application semantics change, e.g., if we revise the trustworthiness associated to some source, or the function that computes the overall trust level of an implicit or multiply-tagged triple.

To address this problem, *abstract models* have been proposed [19, 25], whose underlying intuition is to record *how* the trust level associated with each triple should be computed (rather than the trust level itself). To do so, each explicit triple is annotated with a unique *annotation tag* (a_i – see column "label" in Table 1.3), which identifies both the triple and its source (thus, the same triple coming from different sources would have a different tag and appear more than once in the dataset – cf. a_2, a_6). The annotation tag should not be confused with the quadruple ID (q_i) that is not an integral part of the model and is used just for reference. Even though this tag uniquely associates each explicit quadruple with its source, Table 1.3 includes the source as well for a better understanding of the approach.

The annotation of an inferred triple is the composition of the labels of the triples used to infer it. This composition is encoded with the operator $\odot$. For example, the triple (&a, *type*, *Person*) results from triples (&a, *type*, *Student*) (with tag a_3) and (*Student*, *subClassOf*, *Person*) (with tag a_1); this results to the annotation $a_3 \odot a_1$, quadruple q_9 in Table 1.3. When a triple can be inferred by two or more combinations of quadruples, each such combination results to a different quadruple (cf. q_7, q_8 in Table 1.3).

The concrete trust level is not stored, but can be computed on-the-fly through the annotations using the trust levels and semantics associated with each tag and operator. For example, if we assume the semantics described for the concrete model of Subsection 1.2.1, the trustworthiness level of

(*Student*, *subClassOf*, *Agent*) is the maximum of the trustworthiness associated with quadruples q_7, q_8: the former with $a_1 \odot a_2$ which, per our semantics, amounts to $0,6$ (the minimum of $0,9$ and $0,6$), whereas the latter with $a_1 \odot a_6$ which evaluates to $0,9$ (the minimum of $0,9$ and $0,9$). Thus, the final trustworthiness level of (*Student*, *subClassOf*, *Agent*) is $0,9$.

This computation is essentially identical to the computation taking place in concrete models; the only difference is that concrete models execute this computation at initialization/loading time, whereas abstract models perform this computation *on demand*, i.e., during queries, using stored information that specifies *how* to perform the computation. Abstract models have been considered for provenance (e.g., [19]), as well as for access control (e.g., [30]).

Note that models which focus on what type of metadata information is important for provenance (e.g., when the data was created, how, by whom etc), as well as on how to model such information in an ontology, are orthogonal to the discussion on concrete or abstract provenance models; examples of such approaches are the PROV model[3], the Open Provenance Model (OPM) [28], and the vocabulary proposed in [22], which is based on the Vocabulary of Interlinked Datasets (voiD) [2].

1.2.3 Handling Changes in Abstract Models

The main advantage of abstract models is that most changes in the dataset or in the annotation semantics can be handled easily. For example, if we delete quadruple q_6 (i.e., if we realize that the origin of (*Person*, *subClassOf*, *Agent*) is not s_1), we can easily identify the quadruples to delete (namely, q_8, q_{11}). Even better, if one decides that source s_1 has, e.g., trustworthiness $0,8$ after all, then no recomputation is necessary, because any subsequent computations will automatically consider the new value; the same is true if the semantics for computing the trustworthiness of an implicit triple change.

The main drawback of abstract models is the overhead in storage space (annotations can get complex), and query time (because the annotations have to be computed at query time to determine their actual value).

1.2.4 Alternative Abstract Models

Abstract models can be implemented in different ways; in this subsection, we briefly consider alternatives, which differ along three orthogonal axes, namely: (a) whether multiple annotations of a given triple should be combined or not; (b) the type of identifiers used in complex annotations (provenance IDs, triple IDs or annotation tags); (c) the considered provenance model (how-provenance, why-provenance or lineage) [9].

The first dimension is related to the choice of having several, or just one, annotation(s) for a given triple. Our chosen representation is using several

[3] `http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/`

annotations for the same triple, i.e., if a triple comes from two different sources, or it can be inferred in two different ways, then it will appear twice in the table (in different quadruples). Alternatively, one could require that each triple appears only once in the table, so different annotation tags of the same triple would have to be combined using a second operator, say $\oplus$. For example, q_2, q_6 would be combined in a single quadruple (say q_2') having as annotation the expression $a_2 \oplus a_1$. The advantage of the alternative representation is that the number of quadruples is smaller, but the annotations are usually more complex.

The second dimension is related to the type of identifiers used for labels. In Table 1.3, we used annotation tags, which is the most fine-grained option, as it explicitly describes the dependencies between quadruples. Alternatively, one could use the provenance or triple ID as the main identifier used in the labels. The use of provenance IDs facilitates the computation of concrete tags (because we don't need to go through the annotation tags to identify the trust level of a complex annotation), but is ambiguous as far as the dependencies between quadruples is concerned. The use of triple IDs gives a more compact representation; for example, only one of q_7, q_8 would have to be recorded under this representation (because both are inferred by the same set of triples).

The third dimension is related to the provenance model used. There are three main provenance models that have been discussed in the literature [9]. *Lineage* is the most coarse-grained one and records the input data (labels in our context) that led to the result (complex label), but not the process itself. For example, the lineage label of $(\&a, type, Agent)$ would be $\{a_3, a_1, a_2, a_6\}$, because these are the tags involved in computing its label. *Why-provenance* is more fine-grained; it records the labels that led to the result, but different computations (called *witnesses*) are stored in different sets. In the same example, the why-provenance label of $(\&a, type, Agent)$ would be $\{\{a_3, a_1, a_2\}, \{a_3, a_1, a_6\}\}$; each set in this representation corresponds to one different way to infer $(\&a, type, Agent)$. Finally, *how-provenance* (the approach we employ here) is the most fine-grained type; it is similar to why-provenance, except that it also specifies the operations that led to the generation of the label. Thus, how-provenance requires the use of operators and would produce expressions like the ones appearing in Table 1.3.

In our case, why-provenance and how-provenance could be equivalently used. This is not true in general, because why-provenance cannot capture the order in which the triples/quadruples were used to infer the new triple (which may be relevant for some applications); moreover, why-provenance cannot record the case where a quadruple is used twice in an inference (again, this cannot happen in our setting, but it is possible in general). Finally, the equivalence would break if we considered more operators than just $\odot$, e.g., when triples can be produced via SPARQL queries.

1.3 Abstract Provenance Models: Formalities

1.3.1 Basic Concepts

Now let's see how the above ideas regarding abstract provenance models can be formalized. As explained in Subsection 1.2.2, we will assume that each triple gets a universally unique, source-specific *abstract annotation tag* (taken from a set $\mathcal{A} = \{a_1, a_2, \dots\}$). This tag identifies it across all sources, so if the same triple appears in more than one sources, then each appearance will be assigned a different tag. In some contexts, it makes sense to reserve certain annotation tags to have special semantics, such as "unknown" or "default" [14, 30]. We don't consider this case here, assuming that the provenance of triples is always known.

We define a set of *abstract operators*, which are used to "compose" annotation tags into algebraic expressions. Abstract operators encode the computation semantics of annotations during composition (e.g., for inferred triples). For our setting, we only need one operator, the *inference accumulator operator*, denoted by $\odot$, which encodes composition during inference; more operators would be necessary for other models, e.g., if we considered the more general problem of composing triples using SPARQL queries [18, 11, 35], or propagation semantics such as those proposed in [30, 17, 13]. In general, for each different type of "composition", a different operator should be used.

Depending on the application, abstract operators may be constrained to satisfy properties such as commutativity, associativity or idempotence [19, 30]. These properties provide the benefit that they can be used for efficiently implementing an abstract model. Here, we require that the inference accumulator operator satisfies commutativity ($a_1 \odot a_2 = a_2 \odot a_1$) and associativity ($(a_1 \odot a_2) \odot a_3 = a_1 \odot (a_2 \odot a_3)$). These properties are reasonable for the inference accumulator operator, because the provenance of an implicit triple should be uniquely determined by the provenance of the triples that imply it, and not by the order of application of the inference rules.

The proposed version of the inference accumulator operator supports only binary inference rules, i.e., rules that consider exactly two triples in their body. To support other types, this operator should be defined as a unary operator over $2^{\mathcal{A}}$ (sets of annotation tags). Since we only consider binary inference rules here, we will use the simple version.

A *label* l is an algebraic expression consisting of abstract tags and operators. In our case, a label is of the form $a_1 \odot \dots \odot a_n$, for $n \geq 1$, $a_i \in \mathcal{A}$.

A *quadruple* is a triple annotated with a label. A quadruple is called *explicit* iff its label is a single annotation tag of the form a_i; by definition, such quadruples come directly from a certain source (the one associated with a_i). A quadruple is called *implicit* iff its label is a complex one, containing annotation tags and operators (i.e., of the form $a_1 \odot \dots \odot a_n$, for $n > 1$); implicit quadruples have been produced by inference, thus the complex label.

$f_{\mathcal{A}}(a_1)$	=	0,9	$f_{\mathcal{A}}(a_4)$	=	0,1
$f_{\mathcal{A}}(a_2)$	=	0,6	$f_{\mathcal{A}}(a_5)$	=	0,1
$f_{\mathcal{A}}(a_3)$	=	0,3	$f_{\mathcal{A}}(a_6)$	=	0,9

TABLE 1.4: Defining $f_{\mathcal{A}}$ for our Running Example

Given a label $l = a_1 \odot \ldots \odot a_n$, we say that a_i *is contained in* l (denoted $a_i \sqsubseteq l$) iff $a_i \in \{a_1, \ldots, a_n\}$. Intuitively, this means that the quadruple labeled using a_i was used to produce (via inference) the quadruple with label l.

A *dataset* $\mathcal{D}$ is a set of quadruples. Given a set of inference rules, a dataset $\mathcal{D}$ is called *closed* with respect to said inference rules, iff the application of these inference rules does not generate any new quadruples.

1.3.2 Defining Abstract Provenance Models

An *abstract provenance model* is comprised of an *abstract annotation algebra* and *concrete semantics* for this algebra. An abstract annotation algebra consists of a set of abstract annotation tags and a set of abstract operators, as defined above. The concrete semantics ($\mathcal{S}$) is used to assign "meaning" to these tags and operators, in order to associate labels with concrete values depending on the application at hand (e.g., trust evaluation). The concrete semantics is composed of various subcomponents, as described below.

First, concrete semantics specifies a *set of concrete annotation values*, denoted by $\mathcal{A}_{\mathcal{S}}$, which contains the concrete values that a triple can be annotated with. Depending on the application, this set can be simple (e.g., $\{trusted, untrusted\}$) or more complex (e.g., the $0 \ldots 1$ continuum).

Secondly, concrete semantics associate abstract tokens with concrete values, through a mapping $f_{\mathcal{A}} : \mathcal{A} \mapsto \mathcal{A}_{\mathcal{S}}$. Recall that the abstract tag uniquely identifies the source of the corresponding triple. Thus, given two tags, a_1, a_2 which correspond to the same source (i.e., the triples that have these tags come from the same source), one would expect that $f_{\mathcal{A}}(a_1) = f_{\mathcal{A}}(a_2)$. This is a desirable property, but we don't impose it here, in order to support cases where the same source has more than one associated concrete values (recall the example of Section 1.2). In our running example, the mapping $f_{\mathcal{A}}$ was defined as shown in Table 1.4 (cf. Subsection 1.2.1).

To compute the annotation value of implicit labels (which include operators like $\odot$), we need a concrete definition for abstract operators, i.e., the association of each abstract operator with a concrete one. Moreover, we need a function that will combine different concrete values into a single one to cover the case where a triple is associated to more than one labels. In our running example, $\odot$ was defined to be equal to the *min* function, whereas the combination of different tags was done using the *max* function (cf. Subsection 1.2.1). Note that the operators' concrete definitions should satisfy the properties set by the algebra (e.g., $\odot$ should be commutative and associative).

The concrete semantics allows us to associate each triple with a concrete

Algorithm 1 Annotation

Input: A dataset $\mathcal{D}$
Output: The closure of $\mathcal{D}$
1: $rdfs5 = \emptyset$; $rdfs7 = \emptyset$; $rdfs9 = \emptyset$; $rdfs11 = \emptyset$
2: $rdfs5 = Apply\_\texttt{rdfs5}(\mathcal{D})$
3: $rdfs7 = Apply\_\texttt{rdfs7}(\mathcal{D} \cup rdfs5)$
4: $rdfs11 = Apply\_\texttt{rdfs11}(\mathcal{D})$
5: $rdfs9 = Apply\_\texttt{rdfs9}(\mathcal{D} \cup rdfs11)$
6: **return** $\mathcal{D} \cup rdfs5 \cup rdfs7 \cup rdfs11 \cup rdfs9$

value (see Subsection 1.2.2 for a description of the process), in the same way that concrete models compute concrete labels. As explained before, trustworthiness computation in abstract models takes place *on demand* (e.g., during queries), whereas concrete models perform it at storage time. This causes an overhead at query time for abstract models, but allows them to respond efficiently to changes in the semantics or the data itself, which is a very important property in the context of LOD. In addition, abstract models allow one to associate different semantics to the data (e.g., per user role), or periodically changing semantics (e.g., over time, or in response to certain events), as well as to experiment/test with different semantics; all this can be supported without costly recomputations that would be necessary for concrete models.

1.4 Algorithms

In this section we will present algorithms for: (a) annotating triples; (b) evaluating triples; (c) updating. Our presentation considers only the (binary) inference rules `rdfs5`, `rdfs7`, `rdfs9` and `rdfs11` that appear in Table **??** of Chapter **??** (see also [6]). It is simple to extend the corresponding algorithms for the full list of inference rules in [6] (but in that case the inference accumulator operator should be defined differently – see Subsection 1.3.1).

1.4.1 Annotation

The objective of annotation is to compute all implicit quadruples (i.e., to compute a closed dataset) given a set of explicit ones (i.e., a dataset). The process is shown in Algorithm 1 which calls subroutines executing each of the considered RDFS inference rules sequentially to compute all implicit quadruples. The order of execution that was selected in Algorithm 1 is important, because some rules could cause the firing of others (e.g., an implicit class

subsumption relationship could cause the inference of additional instantiation relationships, so `rdfs11` must be applied before `rdfs9`).

The subroutines called by Algorithm 1 return all the quadruples that can be inferred (using the corresponding inference rule) from their input dataset. Algorithm 2 shows how such a procedure should be implemented for rule `rdfs11` (class subsumption transitivity). To identify all transitively induced subsumption relationships, Algorithm 2 needs to identify all subsumption "chains", regardless of length. In the first step (lines 2-6) the algorithm will identify all chains of length 2, by joining the provided explicit quadruples. Longer chains are identified by joining the explicit quadruples with the newly produced ones (lines 7-16). More specifically, in the first execution of the while loop (lines 7-16), explicit quadruples will be joined with newly generated implicit ones (which correspond to chains of length 2) to compute all chains of length 3. These quadruples are put in *tmp_now* (line 13). In the next execution of the while loop, recently generated implicit quadruples are added to I (the output variable) and joined again with explicit quadruples to generate all chains of length 4. The process continues until we reach a certain chain length in which the while loop does not generate any new quadruples.

The corresponding algorithm for `rdfs5` is similar (just replace *subClassOf* with *subPropertyOf*). The other two considered inference rules (`rdfs7`, `rdfs9`) are simpler, as they consist of a simple join of the quadruples of the appropriate form (in the sense of the for loop in lines 2-6 of Algorithm 2).

Algorithm 2 Class subsumption transitivity rule (`rdfs11`)

Input: A dataset $\mathcal{D}$
Output: All quadruples implied by $\mathcal{D}$ via `rdfs11`

1: $I = \emptyset$; $tmp\_now = \emptyset$; $tmp\_prev = \emptyset$
2: **for all** $q_1 = (u_1, subClassOf, u_2, l_1) \in \mathcal{D}$ **do**
3: **for all** $q_2 = (u_2, subClassOf, u_3, l_2) \in \mathcal{D}$ **do**
4: $tmp\_now = tmp\_now \cup \{(u_1, subClassOf, u_3, l_1 \odot l_2)\}$
5: **end for**
6: **end for**
7: **while** $tmp\_now \neq \emptyset$ **do**
8: $I = I \cup tmp\_now$
9: $tmp\_prev = tmp\_now$
10: $tmp\_now = \emptyset$
11: **for all** $q_1 = (u_1, subClassOf, u_2, l_1) \in \mathcal{D}$ **do**
12: **for all** $q_2 = (u_2, subClassOf, u_3, l_2) \in tmp\_prev$ **do**
13: $tmp\_now = tmp\_now \cup \{(u_1, subClassOf, u_3, l_1 \odot l_2)\}$
14: **end for**
15: **end for**
16: **end while**
17: **return** $\mathcal{D} \cup I$

1.4.2 Evaluation

The evaluation process is used to compute the concrete values of a given (set of) triple(s) in a closed annotated dataset, according to the association of abstract tags and operators with concrete ones provided by the concrete semantics. The process was explained in Subsection 1.2.2; the algorithm is straightforward and omitted due to space limitations.

1.4.3 Adding Quadruples

The main challenge when adding a quadruple q is the efficient computation of the newly-inferred quadruples. Note that we don't support the addition of stand-alone triples, but assume that the provenance information of the triple (via the corresponding explicit label) is an integral part of the update request (e.g., via named graphs [20, 8, 7]). Algorithm 3 describes the process of adding a quadruple q, and is based on a partitioning of quadruples into 4 types, depending on the value of the property (p) of q:

Class subsumption quadruples, which are of the form $(s, subClassOf, o, l)$, can fire `rdfs11` (transitivity of class subsumption) and `rdfs9` (class instantiation) rules only, and are handled in lines 6-21 of Algorithm 3. In the first step (lines 7-9) the new implied quadruple is joined with existing ones that are found "below" the new one in the hierarchy. Then, all produced subsumptions are joined with subsumptions that are found "above" the new one in the hierarchy (lines 10-15). Finally, all new subsumptions are checked to determine whether new instantiations should be produced, according to `rdfs9` (lines 16-20).

Property subsumption quadruples, which are of the form (s, *subPropertyOf*, o, l), can fire `rdfs5` (transitivity of property subsumption) and `rdfs7` (property instantiation) rules only. The process for this case is similar to the previous one and omitted (lines 24-26) due to space limitations.

Class instantiation quadruples, which are of the form $(s, type, o, l)$, fire `rdfs9` (class instantiation) only. These triples are handled in lines 29-31 of Algorithm 3.

Property instantiation quadruples, which are of the form (s, p, o, l) (for $p \neq subClassOf, p \neq subPropertyOf, p \neq type$), can fire `rdfs7` (property instantiation) only, and are handled in lines 33-36 of Algorithm 3.

1.4.4 Deleting Quadruples

Deletion of quadruples is one of the operations where the strength of abstract models becomes apparent. The process consists in identifying (and deleting) all implicit quadruples whose label contains the label of the deleted one (lines 5-7 of Algorithm 4).

Note that Algorithm 4 supports only the deletion of explicit quadruples; deleting implicit quadruples raises some additional complications which are

Algorithm 3 Addition of a Quadruple

Input: A closed dataset $\mathcal{D}$ and an explicit quadruple $q = (s, p, o, a)$
Output: The smallest closed dataset $\mathcal{D}'$ that is a superset of $\mathcal{D} \cup \{q\}$

1: **if** $q \in \mathcal{D}$ **then**
2: **return** $\mathcal{D}$
3: **end if**
4: $\mathcal{D}' = \mathcal{D} \cup \{q\}$
5: $tmp_1 = \emptyset$; $tmp_2 = \emptyset$
6: **if** $p = subClassOf$ **then**
7: **for all** $q_1 = (u_1, subClassOf, s, l_1) \in \mathcal{D}$ **do**
8: $tmp_1 = tmp_1 \cup \{(u_1, subClassOf, o, l_1 \odot a)\}$
9: **end for**
10: **for all** $q_2 = (o, subClassOf, u_2, l_2) \in \mathcal{D}$ **do**
11: $tmp_2 = tmp_2 \cup \{(s, subClassOf, u_2, a \odot l_2)\}$
12: **for all** $q_{tmp} = (u_1, subClassOf, o, l_{tmp}) \in tmp_1$ **do**
13: $tmp_2 = tmp_2 \cup \{(u_1, subClassOf, u_2, l_{tmp} \odot l_2)\}$
14: **end for**
15: **end for**
16: **for all** $q_3 = (u_1, subClassOf, u_2, l_3) \in tmp_1 \cup tmp_2 \cup \{q\}$ **do**
17: **for all** $q_4 = (u, type, u_1, l_4) \in \mathcal{D}$ **do**
18: $\mathcal{D}' = \mathcal{D}' \cup \{(u, type, u_2, l_3 \odot l_4)\}$
19: **end for**
20: **end for**
21: $\mathcal{D}' = \mathcal{D}' \cup tmp_1 \cup tmp_2$
22: **else**
23: **if** $p = subPropertyOf$ **then**
24: ...
25: Similar to lines 7-21
26: ...
27: **else**
28: **if** $p = type$ **then**
29: **for all** $q_1 = (o, subClassOf, u_1, a_1) \in \mathcal{D}$ **do**
30: $\mathcal{D}' = \mathcal{D}' \cup \{(s, type, u_1, a \odot a_1)\}$
31: **end for**
32: **else**
33: **for all** $q_1 = (p, subPropertyOf, p_1, a_1) \in \mathcal{D}$ **do**
34: $\mathcal{D}' = \mathcal{D}' \cup \{(s, p_1, o, a \odot a_1)\}$
35: **end for**
36: **end if**
37: **end if**
38: **end if**
39: **return** $\mathcal{D}'$

Algorithm 4 Deletion of a Quadruple

Input: A closed dataset $\mathcal{D}$ and an explicit quadruple $q = (s, p, o, a)$
Output: The largest closed dataset $\mathcal{D}'$ that is a subset of $\mathcal{D} \setminus \{q\}$
1: **if** $q \notin \mathcal{D}$ **then**
2: **return** $\mathcal{D}$
3: **end if**
4: $\mathcal{D}' = \mathcal{D} \setminus \{q\}$
5: **for all** $q_1 \in \mathcal{D}$ such that $q_1 = (s_1, p_1, o_1, l_1)$ and $a \sqsubseteq l_1$ **do**
6: $\mathcal{D}' = \mathcal{D}' \setminus \{q_1\}$
7: **end for**
8: **return** $\mathcal{D}'$

out of the scope of this work. In particular, to delete an implicit quadruple, one needs to delete at least one explicit as well, otherwise the implicit quadruple will re-emerge due to inference. For example, when deleting q_7 (cf. Table 1.3), one needs to delete either q_1 or q_2. Choosing which quadruple to delete is a non-trivial problem, because the choice should be based on extra-logical considerations and/or user input [15], and is not considered here.

1.4.5 Changing the Semantics

Changing the semantics of the annotation algebra is an important and versatile category of changes, which includes changes like: changing the concrete value associated to an abstract annotation tag; changing the semantics of $\odot$, or the way that different concrete tags associated with the same triple are combined; or changing the concrete values that a triple can take ($\mathcal{A}_\mathcal{S}$).

None of these, seemingly complex, operations requires changes in the dataset. Since we only record how the various quadruples were created, the dataset is not affected by these changes, which are related to how the concrete value associated with a triple is computed. This allows the system administrator to freely experiment with different policies and semantics, without the need for costly recomputations of the values in the dataset.

1.5 Implementation of an Abstract Provenance Model

Our implementation of the proposed abstract provenance model was made over the relational column-store MonetDB[4], to guarantee maximum query efficiency. We present here two different database schemas (named *basic* and

[4] `http://www.monetdb.org/Home`

Exp_Quads

tag	tid	s	p	o
a_1	t_1	*Student*	*subClassOf*	*Person*
a_2	t_2	*Person*	*subClassOf*	*Agent*
a_3	t_3	&a	*type*	*Student*
a_4	t_4	&a	*firstName*	`Alice`
a_5	t_5	&a	*lastName*	`Smith`
a_6	t_2	*Person*	*subClassOf*	*Agent*

Imp_Quads

qid	tid	s	p	o
q_7	t_6	*Student*	*subClassOf*	*Agent*
q_8	t_6	*Student*	*subClassOf*	*Agent*
q_9	t_7	&a	*type*	*Person*
q_{10}	t_8	&a	*type*	*Agent*
q_{11}	t_8	&a	*type*	*Agent*

LabelStore

tag	used_in
a_1	q_7
a_2	q_7
a_1	q_8
a_6	q_8
a_3	q_9
a_1	q_9
a_3	q_{10}
a_1	q_{10}
a_2	q_{10}
a_3	q_{11}
a_1	q_{11}
a_6	q_{11}

TABLE 1.5: Representation Example (Basic Schema)

partitioned) that we used for this implementation; it is easy to adapt the algorithms presented in Section 1.4 to apply to each of those schemas.

1.5.1 Basic Schema

The basic schema consists of three tables, namely **Exp_Quads** (for storing explicit quadruples and their labels), **Imp_Quads** (for storing implicit quadruples) and **LabelStore** (for storing the labels of implicit quadruples). Table 1.5 shows the basic schema populated with the dataset of Table 1.3.

The column **tag** in **Exp_Quads** is used to store the abstract annotation tag. **Imp_Quads** stores a quadruple ID (**qid**) for reference, but does not store the label of the implicit quadruple, because this can get arbitrarily complex and cannot be stored in a single column. **LabelStore** is used for this purpose; in particular, a tuple of the form (a_i, q_j) in **LabelStore** indicates that a_i is contained in the label of q_j. Note that this representation would not be sound had it not been for our hypothesis that $\odot$ is associative and commutative.

The redundant inclusion of the triple ID in tables **Exp_Quads**, **Imp_Quads** (column **tid**) improves certain searches/joins that involve all three fields of the triple (**s**, **p**, **o**). Another optimization (not shown in Table 1.5) was the inclusion of stripped URI IDs (rather than the full URIs) in the above tables, in order to reduce their size and, thus, disk accesses.

1.5.2 Partitioned Schema

The partitioned schema discriminates between the different quadruple types (namely class subsumption, property subsumption, class instantiation and property instantiation – see Subsection 1.4.3), and creates three tables for each type, leading to a total of 12 tables: **Exp_Quads_CS**, **Exp_Quads_PS**, **Exp_Quads_CI**, **Exp_Quads_PI**, **Imp_Quads_CS**, ..., **LabelStore_PI**. The columns in these tables are the same as in the basic schema, except that **p** is omitted when obvious (i.e., in all types except property instantiation).

The explicit/implicit quadruples are stored in the corresponding table depending on their type (e.g., explicit class subsumption quadruples are put in **Exp_Quads_CS**, whereas implicit class instantiation quadruples are put in **Imp_Quads_CI**). We expect this schema to lead to a more efficient implementation, because, e.g., computing class transitivity (`rdfs11`) on the partitioned schema operates over **Exp_Quads_CS** and **Imp_Quads_CS**, whereas the same operation in the basic schema operates over the much larger **Exp_Quads** and **Imp_Quads**; we evaluate this hypothesis in Section 1.6.

1.6 Experimental Evaluation

1.6.1 Experimental Setting

To evaluate abstract models, we performed three experiments. The first one measured the initialization time of abstract and concrete models. For abstract models, this corresponds to the annotation time (computing abstract labels); for concrete ones, it corresponds to the time needed to compute/store the triples' concrete annotation values. The second experiment measured the time required for the evaluation of all triples in the dataset and corresponds to the overhead that abstract models would impose on a query returning the entire dataset. The third experiment measured the time required to add/delete a quadruple, and evaluates the efficiency of abstract models during updates.

All experiments were conducted on a Dell OptiPlex 755 desktop with CPU Intel® Core^{TM}2 Duo CPU E8400 at 3.00GHz, 8 GB of memory, running Linux Ubuntu 10.04.4 LTS release with 2.6.35-31-generic x86_64 Kernel release. We used MonetDB version v11.11.7-Jul2012-SP1 as our relational backend. The implementation of both the inference rules and of all our algorithms (annotation, evaluation, updates) was done using MonetDB's stored procedures. We performed only *cold-cache* experiments, i.e., before running each experiment we flushed out MonetDB's buffers.

We used 4 real datasets from the LOD cloud, namely GADM-RDF[5],

[5] `http://gadm.geovocab.org/`

GeoSpecies[6], CIDOC[7] and GO[8]. These datasets form a diverse set, with different characteristics; statistical information on these datasets was obtained using LODStats[9].

GADM-RDF is a spatial database that stores administrative areas, such as countries and lower level subdivisions. For our experiments, we enhanced GADM-RDF with links from DBPedia, using LDSpider [23], which is a crawling framework that follows RDF links in Linked Data. After this enhancement, GADM-RDF consisted of 11.542.908 million triples that define 26 classes, 40 properties, 25 class subsumptions, and approximately 11 million class/property instantiations.

GeoSpecies contains information on biological orders, families and species, and has a more complex schema than GADM-RDF. As with GADM-RDF, we used LDSpider to enhance it, resulting to a dataset containing 2.596.370 triples describing 111 classes, 40 properties, 97 class subsumptions, 144 property subsumptions and approximately 2 million class/property instantiations.

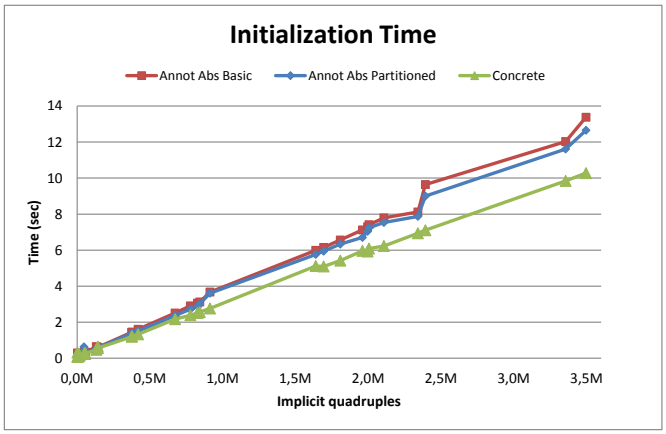
The CIDOC Conceptual Reference Model (CRM) is used in cultural heritage documentation. CIDOC is rather small, but contains a complex schema with several classes/properties organized in a complex hierarchy. In particular, it consists of 3.282 triples that define 82 classes, 260 properties, no instances, 94 class subsumptions and 130 property subsumptions.

The Gene Ontology (GO) aims at standardizing the representation of genes and gene product attributes. GO contains 265.355 triples that define a large number of classes (35.451), structured in a complex hierarchy of 10 levels. It contains 35.451 class instantiations and no properties or property instances.

In addition, to the real datasets above, we also created several synthetic datasets using Powergen [36], which is an RDFS schema generator that produces realistic datasets by taking into account the morphological features that schemas exhibit in reality [37]. We used the parameters of PowerGen to produce 35 ontologies with various characteristics, containing 100-1.000 classes, 113-1.635 properties, 124-50.295 class instances, 70-18.242 property instances and 110-1.321 class subsumptions, with subsumption hierarchy depths ranging from 4 to 8.

We enhanced the above real and synthetic datasets by assigning to each triple a unique annotation tag. To simulate the fact that certain triples may come from different sources, some of the triples were duplicated and assigned more than one tags (namely, 12% got two tags, and 0, 5% got three tags). The set of concrete annotation values $\mathcal{A}_\mathcal{S}$ was taken to be the set of real numbers in the range $0 \ldots 1$. The tags were partitioned into 4 groups, each group being associated with one value from $\mathcal{A}_\mathcal{S}$; this simulates the case where the triples come from 4 different sources, each with its own trustworthiness level. This partitioning essentially defines the function $f_\mathcal{A}$, which maps abstract anno-

[6] http://lod.geospecies.org/
[7] http://www.cidoc-crm.org/
[8] http://www.geneontology.org/
[9] http://wiki.aksw.org/Projects/LODStats



Considered Dataset	Explicit Triples	Implicit Quadruples	Annotation (sec)		Concrete (sec)
			Basic	Partitioned	
GADM-RDF	11.542.908	40	0,282	0,128	0,319
GeoSpecies	2.596.370	721.895	3,744	3,461	3,488
CIDOC	3.282	1.579	0,643	0,228	0,185
GO	265.355	5.581.414	61,135	60,633	32,668

FIGURE 1.1: Initialization Time

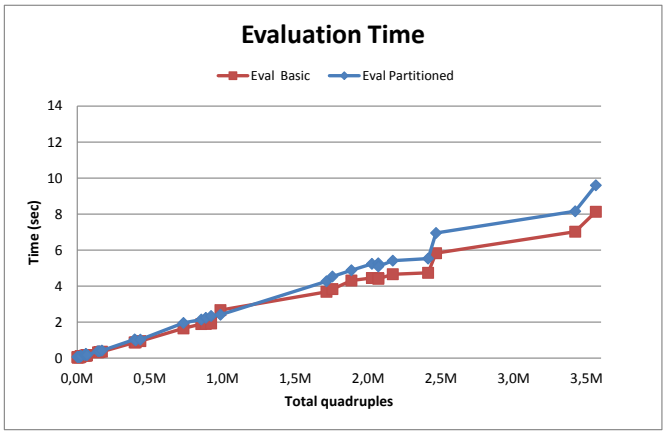
tation tags to concrete values. The inference accumulator operator ($\odot$) was defined to be equal to the *min* function. During evaluation, if a triple is associated with several labels, then the largest one is considered, so the combination of different tags was done using *max*.

1.6.2 Experimental Results

The initialization time is shown in Figure 1.1. Both schemas (basic/partitioned) can annotate 3.5 million implicit quadruples in less than 14 seconds; this is about 20% worse than the time needed to compute and store directly the concrete labels (initialization time for concrete models). Note that the overhead of abstract models is larger for complex datasets (such as GO), due to the more complex labels produced. The small "jumps" of the curves in Figure 1.1 appear at the datasets where the depth of the class subsumption hierarchy increases.

The times in Figure 1.1 are presented as a function of the number of implicit quadruples, rather than the size of the input, because this is the critical parameter for annotation; this is obvious by comparing the performance of complex but small datasets (e.g., GO) with large but simple ones (e.g., GADM-RDF).

The evaluation time is shown in Figure 1.2. In this case, the critical parameter is the total size of the dataset (explicit and implicit quadruples), rather



Considered Dataset	Total Quadruples	Evaluation (sec)	
		Basic	Partitioned
GADM-RDF	13.043.994	28,317	26,176
GeoSpecies	3.655.103	6,911	7,063
CIDOC	5.281	0,074	0,082
GO	5.881.321	18,510	20,606

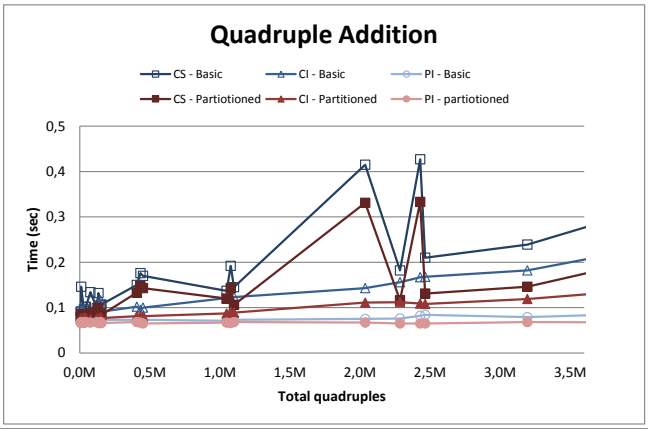
FIGURE 1.2: Evaluation Time

than its complexity (cf. the performance of GADM-RDF and GO). Our experiments show that abstract models impose an overhead of a few seconds for a query returning 3.5 million triples, which is a relatively small overhead for such a demanding query. Note here also the small "jumps" in the curves, which appear at the same points as in Figure 1.1.

Figures 1.3, 1.4 show the times for the addition/deletion of a quadruple to/from a dataset. To perform this experiment, a quadruple was randomly chosen, deleted, and subsequently added; to reduce the effect of randomness, this was repeated 5 times per quadruple type and the averages were reported. The results are given as a function of the input size (number of total quadruples), which is the critical parameter here.

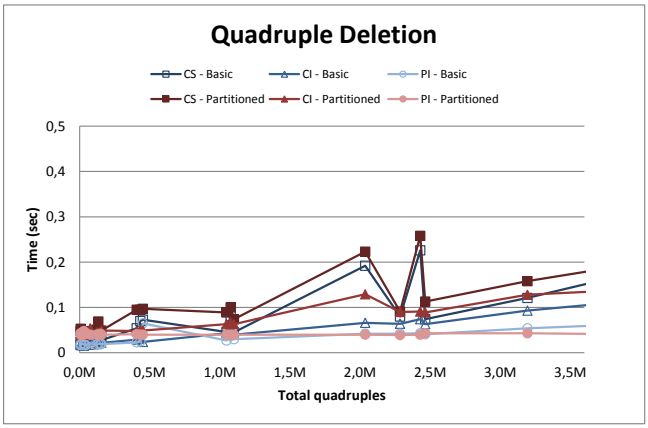
Overall, the time needed to add/delete one quadruple is less than 0,2 seconds in most cases. Deletions are significantly faster than additions. Both operations are affected by the complexity of the dataset, as the results for GO suggest. Note that all synthetic datasets, and some of the real ones, contained no property subsumption quadruples, so no time is reported for this type. This experiment shows clearly that the addition/deletion of quadruples is 1-2 orders of magnitude more efficient for abstract models (compared to concrete ones, where the costly re-initialization has to be performed – see Figure 1.1), and this performance gain increases for larger datasets.

An overall observation from all experiments is that all operations exhibit linear performance with respect to their critical parameter (number of im-



Considered Dataset	Basic (sec)				Partitioned (sec)			
	CS	PS	CI	PI	CS	PS	CI	PI
GADM-RDF	0,432	N/A	0,383	0,390	0,119	N/A	0,111	0,258
GeoSpecies	0,241	0,233	0,166	0,143	0,086	0,125	0,079	0,122
CIDOC	0,112	0,093	0,067	0,082	0,088	0,075	0,061	0,077
GO	3,604	N/A	1,358	0,076	3,165	N/A	0,917	0,071

FIGURE 1.3: Quadruple Addition



Considered Dataset	Basic (sec)				Partitioned (sec)			
	CS	PS	CI	PI	CS	PS	CI	PI
GADM-RDF	0,085	N/A	0,119	0,131	0,062	N/A	0,058	0,118
GeoSpecies	0,062	0,060	0,053	0,050	0,040	0,049	0,052	0,071
CIDOC	0,022	0,019	0,032	0,022	0,066	0,043	0,049	0,040
GO	0,764	N/A	0,328	0,329	0,854	N/A	0,048	0,041

FIGURE 1.4: Quadruple Deletion

plicit or total quadruples respectively) and that the basic and the partitioned schema have similar performance in all cases. Moreover, the tradeoffs involved in the use of abstract models (as opposed to concrete ones) are clear: they provide significant gains in update efficiency, at the expense of a slightly worse performance during initialization and query answering.

1.7 Conclusions

Recording the origin (provenance) of data is important to determine reliability, accountability, attribution or reputation, and to support various applications, such as trust, access control, privacy policies etc. This is even more imperative in the context of LOD, whose unmoderated nature emphasizes this need. In this Chapter, we described an approach towards storing and managing provenance information which is based on abstract models, and discussed the properties of such models compared to standard, concrete ones.

The main conclusion is that abstract models are more suitable for dynamic datasets, as well as in environments where the application supported by provenance (e.g., trust or access control) has dynamic semantics, or needs to support diverse policies/semantics which are fluctuating in a periodical, user-based or event-based fashion. Thus, it is adequate for the dynamic LOD context, where interlinked datasets may change and/or our assessment regarding the quality or trustworthiness of such sources may change.

Bibliography

[1] F. Abel, J. L. De Coi, N. Henze, A. Wolf Koesling, D. Krause, and D. Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *Proceedings of the* 6^{th} *International Semantic Web Conference and the* 2^{nd} *Asian Semantic Web Conference (ISWC/ASWC-07)*, 2007.

[2] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets. In *Proceedings of the Linked Data on the Web Workshop (LDOW-09)*, 2009.

[3] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, USA, 2004.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American-American Edition*, 2001.

[5] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. *The VLDB Journal*, 14:373–396, 2005.

[6] D. Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF Schema. `www.w3.org/TR/2004/REC-rdf-schema-20040210`, 2004.

[7] J.J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the* 3^{rd} *International Semantic Web Conference (ISWC-04)*, 2004.

[8] J.J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs. *Journal of Web Semantics*, 3(4), 2005.

[9] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how and where. *Foundations and Trends in Databases*, 1(4):379–474, 2007.

[10] J. Cheney, S. Chong, N. Foster, M. Seltzer, and S. Vansummeren. Provenance: A future history. In *Proceedings of the* 24^{th} *ACM SIGPLAN Conference Companion on Object Oriented Programming Systems, Languages and Applications (OOPSLA-09)*, pages 957–964, 2009.

[11] C.V. Damasio, A. Analyti, and G. Antoniou. Provenance for SPARQL queries. In *Proceedings of the* 12^{th} *International Semantic Web Conference (ISWC-13)*, 2013.

[12] S. Dietzold and S. Auer. Access control on RDF triple store from a Semantic Wiki perspective. In *Proceedings of the ESWC Workshop on Scripting for the Semantic Web*, 2006.

[13] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 587–598, 2004.

[14] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring RDF triples to capture provenance. In *Proceedings of the* 8^{th} *International Semantic Web Conference (ISWC-09)*, 2009.

[15] G. Flouris, G. Konstantinidis, G. Antoniou, and V. Christophides. Formal foundations for RDF/S KB evolution. *International Journal on Knowledge and Information Systems (KAIS)*, 2013.

[16] G. Flouris, Y. Roussakis, M. Poveda-Villalon, P. N. Mendes, and I. Fundulaki. Using provenance for quality assessment and repair in Linked Open Data. In *Proceedings of the* 2^{nd} *Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn-12)*, 2012.

[17] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: Queries and provenance. In *Proceedings of the ACM SIGMOD/PODS Conference*, 2008.

[18] F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *Proceedings of the Joint EDBT/ICDT 2013 Conference*, 2013.

[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proceedings of the* 26^{th} *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-07)*, pages 31–40, 2007.

[20] H. Halpin and J. Cheney. Dynamic provenance for SPARQL updates using named graphs. In *Workshop on the Theory and Practice of Provenance (TaPP-11)*, 2011.

[21] O. Hartig. Querying trust in RDF data with tSPARQL. In *Procedings of the* 6^{th} *European Semantic Web Conference (ESWC-09)*, 2009.

[22] O. Hartig and J. Zhao. Publishing and consuming provenance metadata on the web of linked data. In *Proceedings of the* 3^{rd} *International Provenance and Annotation Workshop (IPAW-10)*, 2010.

[23] R. Isele, J. Umbrich, C. Bizer, and A. Harth. LDSpider: An open-source crawling framework for the web of linked data. In *Proceedings of* 9^{th} *International Semantic Web Conference (ISWC-10), Posters and Demos*, 2010.

[24] A. Jain and C. Farkas. Secure resource description framework. In *Proceedings of the* 11^{th} *ACM Symposium on Access Control Models and Technologies (SACMAT-06)*, 2006.

[25] G. Karvounarakis and T.J. Green. Semiring-annotated data: queries and provenance. *SIGMOD Record*, 41(3):5–14, 2012.

[26] J. Kim, K. Jung, and S. Park. An introduction to authorization conflict problem in RDF access control. *Innovation in Knowledge-Based and Intelligent Engineering Systems (KES) journal*, 2008.

[27] F. Manola, E. Miller, and B. McBride. RDF primer. `www.w3.org/TR/rdf-primer`, 2004.

[28] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J.V. den Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27:743–756, 2011.

[29] H. Muhleisen, M. Kost, and J.-C. Freytag. SWRL-based access policies for linked data. In *Proceedings of the* 2^{nd} *Workshop on Trust and Privacy on the Social and Semantic Web (SPOT-10)*, 2010.

[30] V. Papakonstantinou, M. Michou, I. Fundulaki, G. Flouris, and G. Antoniou. Access control for RDF graphs using abstract models. In *Proceedings of the* 17^{th} *ACM Symposium on Access Control Models and Technologies (SACMAT-12)*, 2012.

[31] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3), 2009.

[32] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[33] P. Reddivari, T. Finin, and A. Joshi. Policy-based access control for an RDF store. In *IJCAI Workshop on Semantic Web for Collaborative Knowledge Acquisition*, 2007.

[34] C. Strubulis, Y. Tzitzikas, M. Doerr, and G. Flouris. Evolution of workflow provenance information in the presence of custom inference rules. In *Proceedings of the* 3^{rd} *International Workshop on the Role of Semantic Web in Provenance Management (SWPM-12)*, 2012.

[35] Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, 2011.

[36] Y. Theoharis, G. Georgakopoulos, and V. Christophides. PoweRGen: A power-law based generator of RDFS schemas. *Information Systems. Elsevier*, 2011.

[37] Y. Theoharis, Y. Tzitzikas, D. Kotzinos, and V. Christophides. On graph features of semantic web schemas. *TKDE*, 20(5), 2008.

[38] D. Tomaszuk, K. Pak, and H. Rybinski. Trust in RDF graphs. In *Proceedings of the* 17^{th} *East-European Conference on Advances in Databases and Information Systems (ADBIS-13)*, 2013.

[39] S. Vansummeren and J. Cheney. Recording provenance for SQL queries and updates. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 30(4):29–37, 2007.

[40] S. Villata, N. Delaforge, F. Gandon, and A. Gyrard. An access control model for linked data. In *Proceedings of On The Move to Meaningful Internet Systems Workshops 2011 (OTM-11)*, 2011.