Controlling Access to RDF Graphs

Giorgos Flouris[1], Irini Fundulaki[1], Maria Michou[1], and Grigoris Antoniou[1,2]

[1] Institute of Computer Science, FORTH, Greece
[2] Computer Science Department, University of Crete, Greece
{fgeo, fundul, michou, antoniou}@ics.forth.gr

Abstract. One of the current barriers towards realizing the huge potential of Future Internet is the protection of sensitive information, i.e., the ability to selectively expose (or hide) information to (from) users depending on their access privileges. Given that RDF has established itself as the de facto standard for data representation over the Web, our work focuses on controlling access to RDF data. We present a *high-level access control specification language* that allows *fine-grained* specification of access control permissions (at triple level) and formally define its semantics. We adopt an *annotation-based* enforcement model, where a user can explicitly associate data items with annotations specifying whether the item is *accessible or not*. In addition, we discuss the implementation of our framework, propose a set of dimensions that should be considered when defining a benchmark to evaluate the different access control enforcement models and present the results of our experiments conducted on different Semantic Web platforms.

1 Introduction

RDF [14] has been established as a data representation standard over the Web and is expected to play a critical role in the realization of the Future Internet. Several commercial and academic efforts, such as the W3C Linking Open Data initiative [22], target the development of RDF repositories, while the number of applications that publish and exchange possibly sensitive RDF data continuously increases in different domains ranging from bioinformatics [21] to e-government applications [5]. Unfortunately, the potential of these efforts and the realization of the Future Internet is undermined by the lack of an effective mechanism for controlling access to such data. In light of the sensitive nature of such information, the issue of *securing* RDF content and *ensuring the selective exposure of information* to different classes of users depending on their access privileges is becoming all the more important. The building blocks of an access control system are the *specification language*, that allows the expression of access control permissions and policies, and the *enforcement mechanism*, responsible for applying the latter to the data, by denying access to non-accessible data.

It is imperative to support access control at a *fine granularity* (i.e., triple level), and not only at a *coarse-grained level* (i.e., repository level), as done by existing RDF repositories such as Sesame [9] and Jena [2]. To enforce access

control to an RDF repository, we advocate a framework which is repository independent, portable across platforms, and in which *fine-grained access control* is enforced by a component built on top of the RDF repository (as in [6]). More specifically, our contributions in this paper are: *(i)* a high level *access control specification language* for RDF graphs focusing on read-only operations; *(ii)* a formal definition of the language's semantics, based on the triple patterns of SPARQL [20]; *(iii)* an *annotation-based enforcement model* where each triple is automatically marked as accessible or inaccessible based on the available annotations and the access control policy; (*iv*) a set of dimensions that should be taken into account when defining a benchmark for the evaluation of enforcement models for access control; and *(v)* an implementation and experimentation of our framework for different platforms.

The rest of the paper is structured as follows: related work is discussed in Section 2; Section 3 provides a brief introduction to RDF and RDFS; Sections 4 and 5 introduce the access control specification language and its formal semantics respectively; we present the enforcement model, implementation and experiments in Section 6, and conclude in Section 7.

2 Related Work

There have been only a few works dealing with the problem of access control for RDF graphs, and most of them lack formal semantics for the proposed framework. Authors in [6] propose a specification language using graph patterns as defined in SPARQL [20]. Unlike our approach, in [6], triples are not annotated with accessibility information, but the enforcement mechanism is implemented through the injection of the permissions in the query, ensuring that only accessible triples will be obtained. Unfortunately, formal semantics are missing, making impossible to check the correctness of the proposed query-enhancement algorithm. In [23], a specification language for defining permissions for update operations on RDF data is presented, but the authors do not discuss the formal semantics of the language and do not provide an enforcement mechanism, a system implementation, or experiments. Approaches that consider RDFS [7] entailment appear in [17], [18]. In [17], the authors discuss how conflicts can be resolved using RDFS subsumption hierarchies, whereas in [18], inferences are computed without revealing information that might have been explicitly unauthorized. In [11], access control requirements for the context of a Semantic Wiki application are presented, but formal semantics are not discussed and no access control enforcement mechanism is provided. Finally, an approach for access control on XML data, which is very similar to ours, appears in [16].

3 Preliminaries

This paper focuses on RDF graphs [14], i.e., data expressed using *RDF triples* of the form ($subject$, $predicate$, $object$). Given two disjoint and infinite sets U, L, denoting the URIs and literals respectively, an *RDF triple* is any element of the set $U \times U \times (U \cup L)$.

$subject(s)$	$predicate(p)$	$object(o)$
$foaf{:}Person$	rdfs:subClassOf	$foaf{:}Agent$
$foaf{:}age$	rdfs:domain	$foaf{:}Agent$
$foaf{:}age$	rdfs:range	$rdfs{:}Literal$
$foaf{:}mbox$	rdfs:domain	$foaf{:}Agent$
$foaf{:}mbox$	rdfs:range	$rdfs{:}Literal$
$foaf{:}firstName$	rdfs:domain	$foaf{:}Person$
$foaf{:}firstName$	rdfs:range	$rdfs{:}Literal$
$\&a$	rdf:type	$foaf{:}Person$
$\&a$	$foaf{:}age$	17
$\&a$	$foaf{:}mbox$	<mailto:alice@fun.com>
$\&a$	$foaf{:}mbox$	<mailto:alice@work.com>
$\&a$	$foaf{:}firstName$	Alice
$\&b$	rdf:type	$foaf{:}Person$
$\&b$	$foaf{:}mbox$	<mailto:bob@work.com>
$\&b$	$foaf{:}firstName$	Bob

Fig. 1. An RDF Graph $\mathcal{G}$

An *RDF graph* $\mathcal{G}$ is a set of RDF triples. RDF triples can express both schema and data information and can be visualized in a graph, whose nodes are the subjects and objects of triples, and arcs the predicates. Note that, for simplicity, we ignore non-universally identified resources (blank nodes). The RDF Schema (RDFS) language [7] provides a built-in vocabulary for asserting user-defined schemas in the RDF data model, as well as semantics for defining various object types (classes, properties, instances) and relationships between such objects.

Example 1. Throughout this paper, we will use, for illustration purposes, an example taken from the FOAF [1] application. More specifically, we will use the RDF graph consisting of the (data and schema) RDF triples shown in Fig. 1.

4 RDF Access Control Framework

At the core of our access control framework lie the notions of *access control permission* and *access control policy*. Intuitively, an access control permission is used to explicitly set certain triple(s) in an RDF graph to be *accessible* (or *inaccessible*). An access control policy includes a set of access control permissions and information that determines whether a triple is accessible when it is not associated with any permission, or when conflicts arise.

In this work, we concentrate on access control permissions for *read-only queries* only; the specification of permissions for update operations (e.g., write, modify) poses extra difficulties which are out of the scope of this paper and will be considered in the future. As already mentioned, we will use fine-grained access control, where the smallest unit of protection is the RDF triple. Thus, given a user, role, or set of users, an access control permission grants or denies to/from the user the ability to read (i.e., access) the protected RDF triple. In practical applications, it often makes sense to impose access control permissions on sets of triples (say, all triples satisfying a certain property), rather than individual ones. For instance, in Example 1, we may want to deny access to the triples referring to the first names of persons who are less than 18 years old. Therefore, we also allow permissions to be provided for sets of triples.

To express such access control permissions we will use the notion of *triple patterns* from the SPARQL Language Specification [20]. We consider $\mathcal{V}$ to be

a set of variables (denoted by $?x, ?y, \ldots$), disjoint from U,L. We define a *triple pattern* to be an element of the set $(\mathcal{V} \cup U) \times (\mathcal{V} \cup U) \times (\mathcal{V} \cup U \cup L)$. Triple patterns are used to denote the triples in an RDF graph that have a specific form, and can be further restricted using constraints to determine the triples that *are in the scope of* a certain access control permission:

Definition 1. *An* access control permission $\mathcal{R}$ *is of the form*

$$\mathcal{R} = \mathsf{include/exclude}\ (x,\ p,\ y)\ \mathsf{where}\ \mathcal{TP}, \mathcal{C}$$

where (i) $(x,\ p,\ y)$ *is a triple pattern* (ii) $\mathcal{TP}$ *is a conjunction of triple patterns and* (iii) $\mathcal{C}$ *is a conjunction of constraints of the form* $u\ op\ c$ *where* $u \in \mathcal{V}$, $op \in \{=, <, >, \neq\}$ *and* $c \in \mathcal{V} \cup U \cup L$.

Intuitively, an access control permission denotes the RDF triple(s) in an RDF graph $\mathcal{G}$ that are *accessible* (*positive* permissions of the form $\mathsf{include}\ (x,\ p,\ y)\ \mathsf{where}\ \mathcal{TP}, \mathcal{C}$), or *inaccessible* (*negative* permissions of the form $\mathsf{exclude}\ (x,\ p,\ y)\ \mathsf{where}\ \mathcal{TP}, \mathcal{C}$). Essentially, an access control permission can be thought of as a query whose evaluation over an RDF graph results in a (possibly empty) set of triples (the triples in the scope of the permission); said triples are granted or denied access.

$\mathcal{R}_1$: exclude $(?x,\ \textit{foaf:firstName},\ ?y)$ where $(?x,\ \textit{foaf:age},\ ?z), ?z < 18$
$\mathcal{R}_2$: exclude $(?x,\ \text{rdfs:subClassOf},\ ?y)$
$\mathcal{R}_3$: include $(?x,\ \textit{foaf:firstName},\ ?y)$ where $(?x,\ \text{rdf:type},\ \textit{foaf:Person}), (?x,\ \textit{foaf:mbox},\ ?z)$

Fig. 2. Access Control Permissions for the RDF Graph in Fig. 1

Accessible Triples	$\mathcal{R}$
$(\&a,\ \textit{foaf:firstName},\ \texttt{Alice})$	$\mathcal{R}_3$
$(\&b,\ \textit{foaf:firstName},\ \texttt{Bob})$	$\mathcal{R}_3$

Inaccessible Triples	$\mathcal{R}$
$(\&a,\ \textit{foaf:firstName},\ \texttt{Alice})$	$\mathcal{R}_1$
$(\textit{foaf:Person},\ \text{rdfs:subClassOf},\ \textit{foaf:Agent})$	$\mathcal{R}_2$

Fig. 3. Accessible and Inaccessible Triples for the RDF Graph in Fig. 1

Example 2. Consider the access control permissions shown in Fig. 2. Permission $\mathcal{R}_1$ states that the names (triples of the form $(?x,\ \textit{foaf:firstName},\ ?y)$) of individuals $?x$ which are under 18 years old (denoted by a triple of the form $(?x,\ \textit{foaf:age},\ ?z)$ where $?z < 18$) should be inaccessible (exclude). Permission $\mathcal{R}_2$ states that all subsumption relationships should be inaccessible as well. Similarly, permission $\mathcal{R}_3$, states that the names of all persons with an email address should be accessible (include). Fig. 3 shows the accessible and inaccessible triples for the RDF graph shown in Fig. 1, and the related permissions.

Access control permissions explicitly grant or deny access to a certain triple (or set of triples). It would be unrealistic to assume that explicit access rights are set for all triples in an RDF graph, or that permissions are always unambiguous. In our example, the triple $(\&a,\ \textit{foaf:age},\ \texttt{17})$ does not have any permission set (*missing permissions*), whereas the triple $(\&a,\ \textit{foaf:firstName},\ \texttt{Alice})$ is marked as both accessible and inaccessible (*ambiguous permissions*). To determine whether such triples should be accessible, we use the notion of *access control policy*:

Definition 2. *An* access control policy *is a tuple of the form* $\mathcal{P} = (\mathbb{P}, \mathbb{N}, ds, cr)$ *where (i)* $\mathbb{P}$ *is a set of positive permissions (ii)* $\mathbb{N}$ *is a set of negative permissions (iii)* $ds \in \{+, -\}$ *is the* default semantics *that indicates whether access to a triple is* granted *(+) or* denied *(−) by default when access control permissions are missing (iv)* $cr \in \{+, -\}$ *is the* conflict resolution *policy and specifies whether access to a triple is* granted *(+) or* denied *(−) when it is in the scope of both positive and negative permissions.*

Example 3. Consider an access control policy $(\mathbb{P}, \mathbb{N}, ds, cr)$ where $\mathbb{P} = \{\mathcal{R}_3\}, \mathbb{N} = \{\mathcal{R}_1, \mathcal{R}_2\}$ (see Fig. 2), with default semantics *deny* ($ds = -$) and conflict resolution policy again *deny* ($cr = -$). In this case, triple ($\&a$, *foaf*:*firstName*, `Alice`) that is in the scope of both a positive ($\mathcal{R}_3$) and a negative ($\mathcal{R}_1$) permission (see Fig. 3) is eventually denied access ($cr = -$). Similarly, the triple ($\&a$, *foaf*:*age*, `17`), which is not in the scope of any permission, is inaccessible ($ds = -$).

5 Formal Semantics

To formally define the semantics of access control permissions and policies, we will use the notion of *mapping* (as in [19]), which is a function $\mu : \mathcal{V} \mapsto U \cup L$. For ease of presentation, we will denote mappings as sets of pairs, e.g., $\{(?x, \&a), (?y, \texttt{Alice})\}$ is the mapping that maps variables $?x$ and $?y$ to values $\&a$ and `Alice` respectively. We overload the notion of a mapping to apply to triple patterns: in particular, for a triple pattern $tp = (x,\ p,\ y)$, $\mu(tp)$ denotes the triple obtained by replacing the variables in tp with their values according to μ. This way, mappings are used to map triple patterns to triples, by replacing their variables with the corresponding URIs or literals (per μ).

For a triple pattern tp and an RDF graph $\mathcal{G}$, we denote by $\langle\!\langle tp \rangle\!\rangle_{\mathcal{G}}$ the set of mappings obtained by evaluating tp in $\mathcal{G}$, i.e., the mappings that map tp into triples in $\mathcal{G}$. Formally, $\langle\!\langle tp \rangle\!\rangle_{\mathcal{G}} = \{\mu | \mu(tp) \in \mathcal{G}\}$. For example, for the RDF graph $\mathcal{G}$ in Fig. 1 and $tp = (?x,$ *foaf*:*firstName*, $?y)$, two mappings are obtained, namely: $\langle\!\langle tp \rangle\!\rangle_{\mathcal{G}} = \{\{(?x, \&a), (?y, \texttt{Alice})\},\ \{(?x, \&b), (?y, \texttt{Bob})\}\}$.

To define the semantics of a conjunction of triple patterns $\mathcal{TP} = (tp_1, \ldots, tp_n)$, we will use the notion of *compatible mappings* introduced in [19]: the mappings $\mu_1, \ldots, \mu_k$ are *compatible*, iff they have the same value for their common variables. Equivalently, $\mu_1, \ldots, \mu_k$ are compatible iff $\bigcup_{i=1,\ldots,k} \mu_i$ is a mapping. The semantics of $\mathcal{TP}$ for an RDF graph $\mathcal{G}$, denoted by $\langle\!\langle \mathcal{TP} \rangle\!\rangle_{\mathcal{G}}$, is defined as:

$$\langle\!\langle \mathcal{TP} \rangle\!\rangle_{\mathcal{G}} = \{ \bigcup_{i=1,\ldots,n} \mu_i \mid \mu_i \in \langle\!\langle tp_i \rangle\!\rangle_{\mathcal{G}} \text{ for } i = 1, \ldots, n, \text{ and } \mu_1, \ldots, \mu_n \textit{ are compatible}\}$$

Intuitively, $\langle\!\langle \mathcal{TP} \rangle\!\rangle_{\mathcal{G}}$ corresponds to the mappings which map all $tp_i \in \mathcal{TP}$ into triples in $\mathcal{G}$. For instance, for $\mathcal{TP} = ((?x,$ rdf:type, *foaf*:*Person*),($?x$, *foaf*:*mbox*, $?z$)) (see $\mathcal{R}_3$ in Fig. 2) and RDF graph $\mathcal{G}$ in Fig. 1, we get:

$$\begin{aligned}\langle\!\langle \mathcal{TP} \rangle\!\rangle_{\mathcal{G}} = \{\ &\{(?x, \&a), (?z, mailto{:}alice@fun.com)\},\\ &\{(?x, \&a), (?z, mailto{:}alice@work.com)\},\\ &\{(?x, \&b), (?z, mailto{:}bob@work.com)\}\}\end{aligned}$$

For a constraint $\gamma = u\ op\ c$, for $c \in U \cup L$, we say that a mapping μ *satisfies* $u\ op\ c$, denoted by $\mu \vdash (u\ op\ c)$ iff $\mu(u)\ op\ c$ holds; similarly, if $c \in \mathcal{V}$, $\mu \vdash u\ op\ c$ iff $\mu(u)\ op\ \mu(c)$ holds. More generally, we say that a mapping μ satisfies a conjunction of constraints $\mathcal{C}$ (denoted by $\mu \vdash \mathcal{C}$) iff $\mu \vdash \gamma$ for all $\gamma \in \mathcal{C}$. For instance, the mapping $\{(?x, \&a), (?z, \mathsf{17})\}$ satisfies the constraint $?z\ <\ 18$.

We can now define the semantics of an access control permission $\mathcal{R} = \mathsf{include/exclude}$ tp_0 where $\mathcal{TP}, \mathcal{C}$ over some graph $\mathcal{G}$, denoted by $\langle\langle\mathcal{R}\rangle\rangle_{\mathcal{G}}$. The set $\langle\langle\mathcal{R}\rangle\rangle_{\mathcal{G}}$ corresponds to the mappings that map the triple patterns in $\mathcal{TP}$, as well as tp_0, into triples in $\mathcal{G}$, and satisfy $\mathcal{C}$; formally, for $\mathcal{TP} = \{tp_1 \ldots, tp_n\}$:

$$\langle\langle\mathcal{R}\rangle\rangle_{\mathcal{G}} = \{\mu \mid \mu \in \langle\langle\{tp_0, tp_1, \ldots, tp_n\}\rangle\rangle_{\mathcal{G}}, \mu \vdash \mathcal{C}\}$$

The triples that are in the scope of $\mathcal{R}$ are exactly those that tp_0 is mapped to, under some mapping in $\langle\langle\mathcal{R}\rangle\rangle_{\mathcal{G}}$; such triples are denoted by $[[\mathcal{R}]]_{\mathcal{G}}$. Formally:

$$[[\mathcal{R}]]_{\mathcal{G}} = \{t \mid \text{ there exists } \mu \in \langle\langle\mathcal{R}\rangle\rangle_{\mathcal{G}} \text{ such that } \mu(tp_0) = t\ \}$$

Back to our example, Fig. 3 shows the triples in the RDF graph $\mathcal{G}$ of Fig. 1 that are in the scope of permissions $\mathcal{R}_i$ from Fig. 2 ($[[R_i]]_{\mathcal{G}}$).

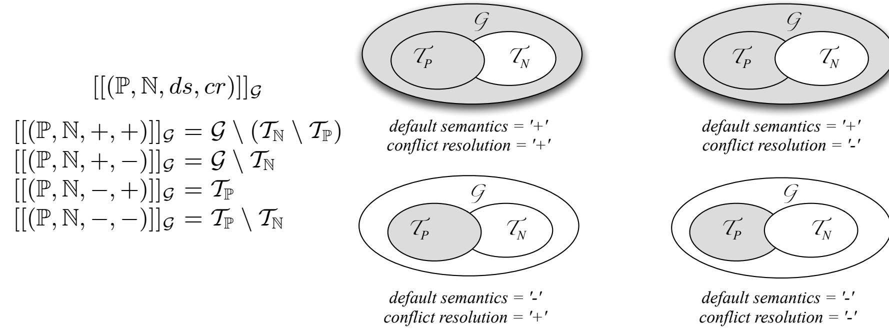


Fig. 4. Access Control Policy Semantics

Now consider some access control policy $\mathcal{P} = (\mathbb{P}, \mathbb{N}, ds, cr)$ and an RDF graph $\mathcal{G}$. The accessible triples of $\mathcal{G}$, given $\mathcal{P}$, are determined by the default semantics (ds) and the conflict resolution policy (cr), in addition to the actual access control permissions, $\mathbb{P}$ and $\mathbb{N}$. Let $\mathcal{T}_{\mathbb{P}} = \bigcup_{\mathcal{R} \in \mathbb{P}} [[\mathcal{R}]]_{\mathcal{G}}$ and $\mathcal{T}_{\mathbb{N}} = \bigcup_{\mathcal{R} \in \mathbb{N}} [[\mathcal{R}]]_{\mathcal{G}}$ be the set of triples in the scope of positive and negative permissions respectively. Triples in $\mathcal{G} \setminus (\mathcal{T}_{\mathbb{P}} \cup \mathcal{T}_{\mathbb{N}})$ (triples that are not in the scope of any permission) or in $\mathcal{T}_{\mathbb{P}} \cap \mathcal{T}_{\mathbb{N}}$ (triples that are in the scope of both positive and negative permissions) may be accessible or not, depending on the values of default semantics (ds) and conflict resolution (cr) respectively. We denote by $[[(\mathbb{P}, \mathbb{N}, ds, cr)]]_{\mathcal{G}}$ the accessible triples for policy $\mathcal{P} = (\mathbb{P}, \mathbb{N}, ds, cr)$. In Fig. 4 we illustrate the policy semantics for the different values of ds and cr (accessible triples are depicted in gray).

6 Implementation and Experiments

In this section we discuss our access control enforcement platform. In particular, we present the architecture of the system, the implementation of the policies' semantics (enforcement mechanism), the dimensions we used to set up our experiments, and the evaluation of our implementation.

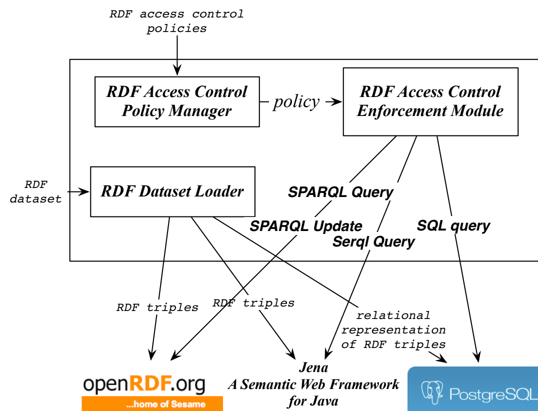


Fig. 5. System Architecture

Architecture: We implemented a main memory platform which serves as an additional access control layer on top of an arbitrary RDF repository. Our goal was for our system to be portable across platforms, so it was designed in a repository-independent way. The system's architecture is shown in Fig. 5. It is comprised of the following modules, all implemented in Java: the RDF Dataset Loader, responsible for loading the RDF triples in the underlying repositories, the RDF Access Control Policy Manager that loads in memory the access control policies and the RDF Access Control Enforcement Module, which translates the access control policies into the appropriate programs that compute the accessible triples of an RDF dataset and annotates accordingly the data in the repositories with accessibility information. Those programs can be SPARQL [20], Serql [8] and Sparul [25] queries (for the corresponding RDF repositories) or SQL queries (when the repository is a relational database).

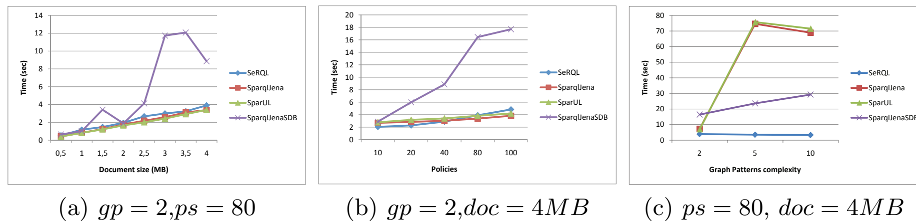
Access Control Enforcement: To enforce an access control policy, we translate it into a SPARQL/Serql/Sparul/SQL query (depending on the query language supported by the underlying repository) which incorporates the access control policy's default semantics and conflict resolution policy. The triples in the result of the evaluation of this query are exactly the accessible triples in the RDF graph. We use this knowledge to appropriately *annotate* each triple as being accessible or inaccessible. Note that annotations could be (and have been) used for other purposes as well, e.g., to implement trust, uncertainty or provenance information [13, 12, 15]. Conceptually, annotations can be represented by adding a fourth column to an RDF triple (hence obtaining a quadruple), denoting whether the triple is accessible or not; if several different user roles need to be supported, one column per role should be added. Physically, annotations can be stored using the *named graphs* mechanism of RDF repositories [10], or, in the case of relational backend, using a large triple table with four columns, following the schema oblivious approach.

Due to lack of space, we will only present the SPARQL/Serql queries used to annotate the RDF triples when the default semantics and conflict resolution policy is "deny" (the rest is similar). Given a policy $\mathcal{P} = (\mathbb{P}, \mathbb{N}, -, -)$, annotations are made using a query of the form $(Q_{\mathbb{P}} \setminus Q_{\mathbb{N}})$ where $Q_{\mathbb{P}}$, $Q_{\mathbb{N}}$ are the queries that compute the set of triples in the scope of permissions in $\mathbb{P}$ and $\mathbb{N}$ respectively (see also Fig. 4). In Fig. 6, we show the form of $Q_{\mathbb{P}}$ and $Q_{\mathbb{N}}$ queries, where tp_0^i is the triple pattern in an access control permission $\mathcal{R}_i$ = include/exclude tp_0^i where $\mathcal{TP}_i, \mathcal{C}_i$ and $expr_i$ is a SPARQL *graph pattern*, i.e., a join of triple patterns and filters that appear in the where clause of $\mathcal{R}_i$. Given that SPARQL does not support the set minus operator ("\" in Fig. 4) between triple sets, we had to implement this operator using main memory set manipulation. In the case of Serql, we use the MINUS operator of the language.

CONSTRUCT $\{tp_0^1$ AND tp_0^2 AND $\ldots tp_0^k\}$
WHERE $expr_1$
UNION
WHERE $expr_2$
...
UNION
WHERE $expr_k$

Fig. 6. Form of Queries $Q_{\mathbb{P}}$ and $Q_{\mathbb{N}}$

Experiments: Our experiments measured the time required to annotate the set of RDF triples, using the above methodology, in state-of-the-art RDF repositories (Sesame [4, 9], Jena [2]) or relational backend (Postgres [3]). All our experiments ran on a 2.2GHz Intel Core 2 Duo running Ubuntu v9.10 Linux, with 4GB of physical memory. We used the SP2Bench [24] data generator to obtain the input RDF graphs. We implemented our approach on top of Jena v2.6.2 using the Java engine ARQ v2.8.2, SDB v1.3.1 which links ARQ to an SQL database back-end (Postgresql v8.4), the Java implementation Sesame v2.3.1 and a relational database (Postgresql v8.4). For Jena we tested the SparqlJenaModule and SparqlJenaSDBModule (processing SPARQL queries) as well as the SPARULModule (processing SPARQL/Update language queries) modules. SparqlJenaModule and SPARULModule load the datasets into main memory whereas the SparqlJenaSDBModule stores the datasets to a Postgresql database. For Sesame we used the SeRQLModule, which processes SPARQL [20] and SeRQL [8] queries in memory.



(a) $gp = 2, ps = 80$ (b) $gp = 2, doc = 4MB$ (c) $ps = 80$, $doc = 4MB$

Fig. 7. Experiments

We measured the time required for the annotation as a function of four different parameters: *(i) document size* (*doc*), i.e., the size of the input RDF graph (size ranging between 500KB-4MB with a 500KB increase); *(ii) policy size* (*ps*), i.e., the number of permissions in the access control policy (for sizes of 10, 20, 40, 80 and 100, with an equal share of positive/negative permissions in each case); *(iii) permission size* (*gp*), i.e., the number of triple patterns and constraints in the where clause of each access control permission (values considered: 2, 5, 10); and *(iv) policy parameters*, i.e., the values of the ds, cr parameters of the input policy (all 4 combinations considered).

Evaluation: Fig. 7 shows a subset of the results of our experiments. In each graph, the annotation time is presented as a function of one of the above parameters *(i)*-*(iv)*, for fixed values for the other parameters (see Fig. 7). Due to lack of space[1] we do not include the results of experiments when varying the policy parameters (default semantics and conflict resolution policy): we report here on the (*deny*, *deny*) case only because it is the most common one. The results show that our approach scales along the considered parameters. All the platforms that we ran our experiments on demonstrated a linear behavior as document, policy sizes and permission complexity increased (except the Jena SPARUL and SPARQL Modules). We do not show here the results of our experiments for the pure relational solution since it proved to be extremely expensive (in some cases causing an increase in annotation time of more than 100%) when compared to the RDF-based solutions. We believe that this is caused by the large number of self-joins (required to implement the WHERE clause of the respective SQL query) on the large triple table.

7 Conclusions

We addressed the problem of selectively exposing information in RDF graphs to different classes of users depending on their access privileges, a problem of great importance for the Future Internet, given the sensitive nature of several datasets. We advocated in favor of a *fine-grained solution*, in which the smallest unit of protection is the RDF triple. In our proposal, triples are specifically annotated as accessible (or inaccessible) using access control permissions, which can be set on sets of triples using *triple patterns* as defined in SPARQL. We then showed how our policies can be translated into SPARQL queries which are subsequently evaluated on different Semantic Web platforms in order to assess the applicability of our approach.

In the future, we will consider extensions of our framework to support RDFS entailment semantics, in effect allowing the determination of the (in)accessibility of non-explicit triples. Another possible extension is the incorporation of blank nodes in our model, as well as the consideration of security levels, i.e., non-binary accessibility annotations.

[1] The full set of our experimental results can be found at `www.ics.forth.gr/isl/RAC`.

References

1. Friend of a friend: www.foaf-project.org.
2. Jena A Semantic Web Framework for Java. http://jena.sourceforge.net/.
3. PostgreSQL. http://www.postgresql.org/.
4. Sesame: RDF Schema Querying and Storage. http://www.openrdf.org/.
5. www.data.gov, data.gov.uk.
6. 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 *ISWC/ASWC*, 2007.
7. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. www.w3.org/TR/2004/REC-rdf-schema-20040210, 2004.
8. J. Broekstra and A. Kampman. SeRQL: A Second Generation RDF Query Language. In *Workshop on Semantic Web Storage and Retrieval*, 2003.
9. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *ISWC*, 2002.
10. J. J. Carroll, C. Bizer, P. J. Hayes, and P. Stickler. Named Graphs. *J. Web Semantics*, 3(4), 2005.
11. S. Dietzold and S. Auer. Access Control on RDF Triple Store from a Semantic Wiki Perspective. In *ESWC Workshop on Scripting for the Semantic Web*, 2006.
12. R. Queiroz Dividino, S. Sizov, S. Staab, and B. Schueler. Querying for Provenance, Trust, Uncertainty and Other Meta Knowledge in RDF. *Journal of Web Semantics*, 7(3), 2009.
13. M. Knechtel F. Baader and R. Penaloza. A Generic Approach for Large-Scale Ontological Reasoning in the Presence of Access Restrictions to the Ontology's Axioms. In *ISWC*, 2009.
14. B. McBride F. Manola, E. Miller. RDF Primer. www.w3.org/TR/rdf-primer, February 2004.
15. G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring RDF Triples to Capture Provenance. In *ISWC*, 2009.
16. I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *SACMAT*, 2004.
17. A. Jain and C. Farkas. Secure Resource Description Framework. In *SACMAT*, 2006.
18. J. Kim, K. Jung, and S. Park. An Introduction to Authorization Conflict Problem in RDF Access Control. In *KES*, 2008.
19. J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *ISWC*, 2006.
20. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. www.w3.org/TR/rdf-sparql-query, January 2008.
21. Gene Ontology. www.geneontology.org.
22. W3C Linking Open Data. esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.
23. P. Reddivari, T. Finin, and A. Joshi. Policy-Based Access Control for an RDF Store. In *Semantic Web for Collaborative Knowledge Acquisition*, 2007.
24. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL Performance Benchmark. Technical report, arXiv:0806.4627v1 cs.DB, 2008.
25. A. Seaborne and G. Manjunath. SPARQL/Update: A language for updating RDF graphs. Technical report, Hewlett-Packard, 2007.