*f*Query: SPARQL Query Rewriting to Enforce Data Confidentiality

Said Oulmakhzoune[1], Nora Cuppens-Boulahia[1], Frédéric Cuppens[1], and Stephane Morucci[2]

[1] IT/Telecom-Bretagne, 2 Rue de la Chataigneraie, 35576 Cesson Sevigne - France
{said.oulmakhzoune,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
[2] Swid, 80 Avenue des Buttes de Coësmes, 35700 Rennes - France
stephane.morucci@swid.fr

Abstract. RDF is an increasingly used framework for describing Web resources, including sensitive and confidential resources. In this context, we need an expressive language to query RDF databases. SPARQL has been defined to easily localize and extract data in an RDF graph. Since confidential data are accessed, SPARQL queries must be filtered so that only authorized data are returned with respect to some confidentiality policy. In this paper, we model a confidentiality policy as a set of positive and negative filters (corresponding respectively to permissions and prohibitions) that apply to SPARQL queries. We then define rewriting algorithms that transform the queries so that the results returned by transformed queries are compliant with the confidentiality policy.

1 Introduction

The RDF [1](Resource Definition Framework) data model is based upon the idea of making statements about resources (in particular Web resources) in the form of subject-predicate-object expressions. These expressions are known as triples in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the proposition "Bob's salary is 60k" in RDF is as the triple: a subject denoting "Bob", a predicate denoting "has salary", and an object denoting "60k". A collection of RDF statements intrinsically represents a labeled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models traditionally used in computing today.

However, in practice, as more data is being stored in RDF formats, a need has arisen for a simple way to locate specific information. SPARQL [2] (Simple Protocol And RDF Query Language) is a powerful query language which fills that space, making it easy to find the data you need in the RDF graphs. It was standardized by the RDF Data Access Working Group of the World Wide Web Consortium, and is considered a key semantic web technology. A

SPARQL query consists of triple patterns, conjunctions, disjunctions, and optional patterns. SPARQL allows users to write globally unambiguous queries. For example,the following query returns names and salaries of all employees.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX emp:<http://tb.eu/employer/0.1/>
SELECT ?name ?salary
WHERE
{
     ?employee rdf:type   emp:Employee.
     ?employee foaf:name  ?name.
     ?employee emp:salary ?salary.
}
```

Basically, the SPARQL syntax resembles SQL, but the advantage of SPARQL is that it enables queries spanning multiple disparate (local or remote) data sources containing heterogeneous semi-structured data. However, since a SPARQL query may access confidential data, it is necessary to design security mechanisms to control the evaluation of SPARQL queries and prevent these queries from illegally disclosing confidential data.

Our approach is to rewrite the user SPARQL query by adding some SPARQL filters to that query. When, the user sends his or her SPARQL query to the server, our system will intercept this query and checks the security rules corresponding to that user (Figure 1). Then it rewrites the query by adding the corresponding SPARQL filters. The execution result of the rewritten query is returned to the user. The figure 1 illustrates fQuery, our approach.

In our approach, the answer to the rewritten may differ from the user's initial query. In that case and as suggested in [3], we can check the query validity of the rewritten query with respect to the initial query and notify the user when the query validity is not guaranteed.

The rest of this paper is organized as follows. Section 2 presents the basic principales of rewriting SPARQL query by introducing some examples. Section 3 presents some definition and theorems that are used in other sections. Section 4 defines the security policy model for SPARQL and some of its properties. In section 5, we specify the rewriting query algorithm. Section 6 presents some related works and finally section 7 concludes this paper.

2 Rewriting SPARQL Query: Basic Principles

Let us take an example of query transformation. We assume that the user Bob tries to select the name and the salary of each employee. We assume also that Bob is not permitted to see salaries of employees who earn more than 60K. The table 1 shows Bobs SPARQL query before and after transformation. The presence of the OPTIONAL construct in the transformed query makes it a non-conjunctive (disjunctive) one. It means that: if the condition inside the OPTIONAL clause is False then the value of the salary variable is assigned to Null.

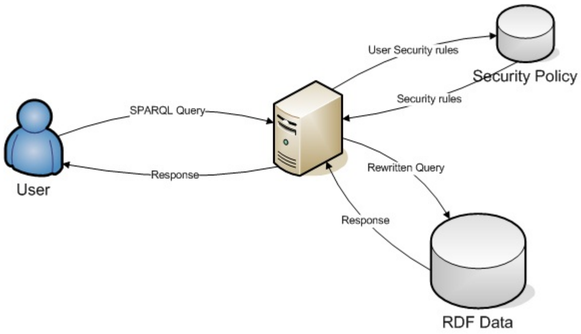

Fig. 1. *f*Query approach

Before transformation	After transformation

SELECT ?name ?salary
WHERE
{
  ?employee rdf:type   emp:Employee.
  ?employee foaf:name ?name.
  ?employee emp:salary ?salary.
}
	SELECT ?name ?salary
	WHERE
	{
	  ?employee rdf:type emp:Employee.
	  ?employee foaf:name ?name.
	  Optional {
	    ?employee emp:salary ?salary.
	    Filter(?salary<60000)
	  }
	}

Table 1. Example of query transformation

The access control policy is based on filter definitions. For each user or group of users, we assign a set of filters. Depending on the policy type, we consider two different types of filter: (1) Positive filters corresponding to permission and (2) Negative filters corresponding to prohibition.

Those filters may be associated with a simple condition or an involved condition. The example 5 (section 5.1) illustrates filter associated with a simple condition. The example 6 (section 5.2) shows an example of filter associated with an involved condition. Filters associated with involved condition provides means to protect relationships, as illustrated in the example 6. In our approach we assume that when a user asks a query, we can get additional information like the user identity. This additional information may be used in the filter definition (see the example 6).

Filters actually provide a generic approach to represent an access control policy for RDF documents which does not rely on a specific language. However, it would be also interesting to define a user friendly specification language to express an access control policy for RDF documents. Due to space limitation, this issue is not addressed in this paper but represent an extension of our work.

3 Notations, Definitions and Theorems

As mentioned in the introduction, an RDF database is represented by a set of triples. So, we denote E as the set of all RDF triples of our database. We denote $E_{subject}$ (respectively $E_{predicate}$,E_{object}) as the projection of E on subject (resp. predicate and object). $E_{subject}$ represents (resp. $E_{predicate}$, E_{object}) the set of all subjects (resp. predicates, objects) of the RDF triples of E.

Definition 1: We define a "condition of RDF triples" as the application ω : $E \rightarrow Boolean$ which associates each RDF triple $x = (s, p, o)$ of E to an element of set $Boolean = \{True, False\}$.

$$\omega : E \rightarrow Boolean,\ x \rightarrow \omega(x)$$

$\omega(x)$ is expressed in terms of s, p and o where $x = (s, p, o)$. We define also the negation of ω denoted $\bar{\omega}$ as follows:

$$\bar{\omega} : E \rightarrow Boolean,\ x \rightarrow \bar{\omega}(x) \text{ such that } (\forall x \in E)\bar{\omega}(x) = \overline{\omega(x)} = \neg(\omega(x))$$

For each element x of E, we say that $\omega(x)$ is satisfied if $\omega(x) = True$. Otherwise we say that $\omega(x)$ is not satisfied.

Definition 2: We define the "simple condition of RDF triples" as the condition of RDF triples that uses the same operators as the SPARQL filter ($regex, bound, =, <, >$ ...) and constants (see [2] for a complete list of possible operators).

Example 1. $(\forall x = (s, p, o) \in E)\omega(x) = (s \neq$emp:Alice$) \vee ((p =$foaf:name$) \wedge (o \neq$'Alice'$))$

Definition 3: Let tp be a triple pattern of the where clause of a SPARQL query and ω be a condition on RDF triples. We define the projection of ω relative to tp as the condition $\omega(tp)$ expressed in terms of the tp SPARQL variables. We denote that projection as $\pi_{\omega/tp}$, $\pi_{\omega/tp} = \omega(tp)$.

Example 2. Let $x = (s, p, o) \in E$ such that
$\omega(x) = (s \neq$ emp:Alice$) \wedge (p =$foaf:name$) \wedge (s \neq o)$ and $tp =$ (emp:Charlie,?m,?n)
$\pi_{\omega/tp} = \omega(tp)$
$\pi_{\omega/tp}(?m, ?n) = (?m =$foaf:name$) \wedge (?n \neq$emp:Charlie$)$

We denote constants of conditions of RDF triple Ω_{True} and Ω_{False} applications defined as follows:

$$\begin{aligned} \Omega_{True} : E &\rightarrow Boolean \\ x &\rightarrow True \end{aligned} \qquad \begin{aligned} \Omega_{False} : E &\rightarrow Boolean \\ x &\rightarrow False \end{aligned}$$

Definition 4: Let ω_1 and ω_2 be two conditions on RDF triples. We define the conditions $\omega_1 \wedge \omega_2$ and $\omega_1 \vee \omega_2$ as follows:

$$\begin{aligned} \omega_1 \wedge \omega_2 : E &\rightarrow Boolean \\ x &\rightarrow \omega_1(x) \wedge \omega_2(x) \end{aligned} \qquad \begin{aligned} \omega_1 \vee \omega_2 : E &\rightarrow Boolean \\ x &\rightarrow \omega_1(x) \vee \omega_2(x) \end{aligned}$$

Definition 5: Let ω be a condition on RDF triples. We define the subset of E that satisfies the condition ω, denoted $I(\omega)$, as follows:

$$I(\omega) = \{x \in E| \quad \omega(x) = True\}$$

We define the complement of the set $I(\omega)$ in E, denoted $\overline{I(\omega)}$, as follows:

$$\overline{I(\omega)} = \{x \in E| \quad x \notin I(\omega)\} = E \backslash I(\omega)$$

Theorem 1: Let ω be a condition on RDF triples, $I(\bar{\omega}) = \overline{I(\omega)} = E \backslash I(\omega)$

Proof of theorem 1:
$x \in I(\bar{\omega}) \iff \{x \in E| \quad \bar{\omega}(x) = True\} \iff \{x \in E| \quad \overline{\omega(x)} = True\}$
$\iff \{x \in E| \quad \omega(x) = False\} \iff \{x \in E| \quad x \notin I(\omega)\} \iff x \in \overline{I(\omega)}$. □

Theorem 2: Let ω_1 and ω_2 be two conditions on RDF triples. We have the following properties: $I(\omega_1 \wedge \omega_2) = I(\omega_1) \cap I(\omega_2)$ and $I(\omega_1 \vee \omega_2) = I(\omega_1) \cup I(\omega_2)$

Proof of theorem 2:
$x \in I(\omega_1 \wedge \omega_2) \iff \omega_1(x) \wedge \omega_2(x) = True$
$\iff \omega_1(x) = True$ and $\omega_2(x) = True \iff x \in I(\omega_1)$ and $x \in I(\omega_2)$
$\iff x \in I(\omega_1) \cap I(\omega_2)$
Then $I(\omega_1 \wedge \omega_2) = I(\omega_1) \cap I(\omega_2)$
With the same reasoning we can prove that $I(\omega_1 \vee \omega_2) = I(\omega_1) \cup I(\omega_2)$.
By induction (recurrence) we can prove the properties bellow. □
Generalization of the theorem 2: Let $n \in \mathbf{N}^*$ and $\{\omega_i\}_{0 \leq i \leq n}$ be a set of conditions on RDF triples.

$$I(\wedge_{i=0}^{n} \omega_i) = \cap_{i=0}^{n} I(\omega_i)$$
$$I(\vee_{i=0}^{n} \omega_i) = \cup_{i=0}^{n} I(\omega_i)$$

4 Security policy

In our proposal we define the security policy as a set of permissions or a set of prohibitions. We also assume that the policy is closed.

4.1 Permission

A security policy rule is defined as the permission for a user to select a set of RDF triples of E that satisfies a condition on RDF triples denoted ω. It means that the user is permitted to select only the RDF triples of the subset $I(\omega)$. We denote this permission as $Permission(\omega)$.

Example 3. Bob is permitted to see the name and email of all employees data stored in the RDF database. This rule can be expressed as the permission to select a set of RDF triples of E that satisfies the condition ω defined as follows:

$$(\forall x = (s, p, o) \in E) \quad \omega(x) = \begin{cases} True & \text{if } p \in P \\ False & \text{if } p \notin P \end{cases}$$

Such that $P = \{$foaf:name, foaf:mbox$\}$ is a set of predicates associated with the information name and email.

Let $\{Rule_i\}_{(1 \leq i \leq n)}$ be a set of security rules (permission rules) associated with a user and $\{\omega_i\}_{1 \leq i \leq n}$ be a set of conditions on RDF triples such that $n \in \mathbf{N}^*$ and $Rule_i = Permission(\omega_i)$. So the user could select the RDF triples of each set $I(\omega_i)$. It means that the user could select the RDF triples of the set $\cup_{i=1}^{n} I(\omega_i)$. According to the result of the theorem 2, the user is permitted to select the RDF triples of $I(\vee_{i=1}^{n} \omega_i)$. So the user is permitted to select RDF triples that satisfies the condition $\omega = \vee_{i=1}^{n} \omega_i$. We deduce that:

$$\bigcup_{i=1}^{n} Permission(\omega_i) = Permission(\bigvee_{i=1}^{n} \omega_i)$$

It means that a set of permission rules $\{Permission(\omega_i)\}_{1 \leq i \leq n}$ could be expressed as one permission rule defined as the permission to select RDF triples that satisfies the condition $\omega = \vee_{i=1}^{n} \omega_i$.

4.2 Prohibition

In the case of prohibition we define the security policy rule as the prohibition for a user to select a set of RDF triples of E that satisfies a condition on RDF triples denoted ω. It means that the user is prohibited to select any RDF triples of the subset $I(\omega)$. We denote this prohibition as $Prohibition(\omega)$.

Example 4. Bob is not permitted to select the salary and the birth day of all employees data stored in a RDF database. This rule can be expressed as $Prohibition(\omega)$ such that ω is defined as follows:

$$(\forall x = (s,p,o) \in E) \quad \omega(x) = \begin{cases} True & \text{if } p \in P \\ False & \text{if } p \notin P \end{cases}$$

Such that $P = \{$emp:salary, foaf:birthday$\}$ is a set of the predicates associated with the information salary and birth day. So ω could be written as:
$(\forall x = (s,p,o) \in E) \quad \omega(x) = (p = \text{emp:salary}) \vee (p = \text{foaf:birthday})$

With the same reasoning as on the previous section 4.1, we deduce that:

$$\bigcup_{i=1}^{n} Prohibition(\omega_i) = Prohibition(\bigvee_{i=1}^{n} \omega_i)$$

Assuming that $\{Prohibition(\omega_i)\}_{1 \leq i \leq n}$ are all security rules associated with a user, we can prove the following result:

$$\bigcup_{i=1}^{n} Prohibition(\omega_i) = Permission(\bigwedge_{i=1}^{n} \overline{\omega_i})$$

5 *f*Query: Our query rewriting model

We rewrite the user query by adding filters and/or removing triples of pattern from the where clause following the associated security policy (see section 5.1). Sometimes it is also necessary to add triples of pattern to the query in order to satisfy the security policy (see section 5.2).

Our query rewriting algorithm treats each BGP [2] (Basic Graph Pattern) of a SPARQL query. Each BGP is handled separately from the others.

5.1 Case of simple condition ω

Let Bgp be a basic graph pattern of the where clause of a SPARQL query. We check the security rule associated with the condition ω ($Permission(\omega)$ or $Prohibition(\omega)$) for each triple pattern $tp = (s, p, o)$ of Bgp by calculating the projection $\pi_{\omega/tp}$. There are three cases depending on the $\pi_{\omega/tp}$ value.
Permission case:

- $\pi_{\omega/tp} = \Omega_{True}$
  It means that $\pi_{\omega/tp}$ is always **true** for each SPARQL variable of the triple pattern tp. In this case the triple pattern *tp* matches with the security policy. So there is no action to do for *tp*. We check the security condition ω for the next triple pattern.
- $\pi_{\omega/tp} = \Omega_{False}$
  It means that $\pi_{\omega/tp}$ is always **false** for each SPARQL variable of the triple pattern *tp*. In this case the triple pattern tp does not match with the security policy. So we delete this triple pattern *tp* from *Bgp*. Then we check the security condition ω for the next triple pattern.
- Otherwise $\pi_{\omega/tp}$ is expressed in terms of *tp* variables. In this case, we put tp in an OPTIONAL construct and we add the positive filter φ to it. Then we add this optional construct to *Bgp*. The positive filter φ is defined as follows:

  $$\varphi(tp) = FILTER(\pi_{\omega/tp}) = FILTER(\omega(tp))$$

  This filter filters the RDF triples that satisfy the condition ω. The presence of the OPTIONAL construct in the transformed query makes it a non-conjunctive one.

Prohibition case:

- $\pi_{\omega/tp} = \Omega_{True}$
  It means that $\pi_{\omega/tp}$ is always **true** for each SPARQL variable of the triple pattern *tp*. In this case, RDF triples that match with the triple pattern *tp* are prohibited. So we delete this triple pattern *tp* from the basic graph pattern *Bgp*. Then we check the security condition ω for the next triple pattern.
- $\pi_{\omega/tp} = \Omega_{False}$
  It means that $\pi_{\omega/tp}$ is always **false** for each SPARQL variable of the triple pattern *tp*. In this case RDF triples that match with the triple pattern *tp* are allowed to be selected. So there is no action to do for *tp*. We check the security condition ω for the next triple pattern.

– Otherwise $\pi_{\omega/tp}$ is expressed in terms of tp variables. In this case, we put tp in an OPTIONAL construct and we add the filter φ to it. Then we add this optional construct to Bgp. The filter φ is defined as follows:

$$\varphi(tp) = FILTER(\overline{\pi_{\omega/tp}}) = FILTER(\overline{\omega(tp)})$$

This filter filters the RDF triples that do not satisfy the condition ω.

We define *Algo1*, the query rewriting algorithm for a simple condition and *handleBGP* the related algorithm that handles a basic graph pattern, in the case of a clause "where" with simple condition.

Algorithm 1 Algo1 (Query, ω, ruleType). Query rewriting Algorithm for a simple condition

```
Require: $\omega$ is simple condition
  for each basic graph pattern $Bgp$ of Query do
    $handleBGP(Bgp, \omega, ruleType)$
  end for
```

Algorithm 2 handleBGP (Bgp, ω, ruleType)

```
Require: $\omega$ is simple condition
  for each triple pattern $tp$ of $Bgp$ do
    if $\pi_{\omega/tp}$=$\Omega_{True}$ then
      if $ruleType$ = PROHIBITION then
        delete $tp$ from $Bgp$
      end if
    else if $\pi_{\omega/tp}$=$\Omega_{False}$ then
      if $ruleType$ = PERMISSION then
        delete $tp$ from $Bgp$
      end if
    else
      create new optional element $opEl$
      move $tp$ to $opEl$
      if $ruleType$ = PERMISSION then
        add the filter $FILTER(\pi_{\omega/tp})$ to $opEl$
      else if $ruleType$ = PROHIBITION then
        add the filter $FILTER(\overline{\pi_{\omega/tp}})$ to $opEl$
      end if
      add $opEl$ to $Bgp$
    end if
  end for
```

Example 5. Bob is not permitted to see salaries of employees who earn more than 50K and their premiums if it is greater than 9K. This prohibition could be expressed as $Prohibition(\omega)$ where ω is defined as: $(\forall x = (s, p, o) \in E)$

$$\omega(x) = ((p = \text{emp:salary}) \wedge (o \geq 50000)) \vee ((p = \text{emp:premium}) \wedge (o \geq 9000))$$

Bob tries to select the name, the salary of each employee and their premium if it is greater than 10K. He executes the following query:

```
SELECT ?name ?salary ?premium
WHERE
{
     ?s1 foaf:name  ?name,
         emp:salary ?salary.
     Optional{
           ?s1 emp:premium ?premium. Filter(?premium > 10000)
   }
}
```

Let $tp_1 = (?s1$,foaf:name$, ?name)$ and $tp_2 = (?s1$,emp:salary$, ?salary)$ and $tp_3 = (?s1$,emp:premium$, 10000)$ be triples of pattern of the where clause of Bob's query. The query has two basic graph patterns $Bgp_1 = \{tp_1, tp_2\}$ and $Bgp_2 = \{tp_3\}$.

We have $\pi_{\omega/tp_1} = \omega(tp_1) = False = \Omega_{False}$

$\pi_{\omega/tp_2} = \omega(tp_2) = (?salary \geq 50000)$

$\pi_{\omega/tp_3} = \omega(tp_3) = (?premium \geq 9000)$

$\pi_{\omega/tp_1} = \Omega_{False}$ so there is nothing to do with the triple pattern tp_1.

$\pi_{\omega/tp_2} = (?salary \geq 50000)$ so we add the filter FILTER($?salary < 50000$) =FILTER($\overline{\pi_{\omega/tp_2}}$) to the Bgp_1.

$\pi_{\omega/tp_3} = (?premium \geq 9000)$ so we add the filter FILTER($?premium < 9000$) to Bgp_2. The rewritten query will be as follows:

```
SELECT ?name ?salary ?premium
WHERE
{
     ?s1 foaf:name  ?name.
     Optional{
          ?s1 emp:salary ?salary.
          FILTER (?salary < 50000)
     }
     Optional{
          Optional{
               ?s1 emp:premium ?premium. Filter(?premium < 9000)
          }
          Filter(?premium > 10000)
     }
}
```

5.2 Case of involved condition ω

Definition 6 Before formally defining the concept of involved condition, let us first take an example. Bob is permitted to select the information of the network department employees. The condition ω associated with this rule could be expressed as follows:

$(\forall x = (s, p, o) \in E)\ \omega(x) = \begin{cases} True & \text{if } (\exists y \in E) | y = \text{(s,emp:dept,'Network')} \\ False & \text{Otherwise} \end{cases}$

It means that Bob can select only the RDF triples where the subject has also the predicate emp:dept with the value 'Network'. $\omega(x)$ does not depend only on the RDF triple x but it also depends on another RDF triple y that shares the same subject of x and its predicate is $emp : dept$ with the value 'Network'.

Let $n \in \mathbf{N}^*$, $\{\omega_i\}_{1\leq i\leq n}$ be a set of simple conditions on RDF triples and $\{p_i\}_{1\leq i\leq n}$ a set of predicates of $E_{predicate}$. We can generalize the example above by defining the condition ω as follows: $(\forall x = (s, p, o) \in E)$

$$\omega(x) = \begin{cases} True & \text{if } (\exists(x_1, ..., x_n) \in E^n)|(\forall 1 \leq i \leq n)x_i = (s, p_i, o_i) \\ & \text{where } o_i \in E_{object} \text{ and } \omega_i(x_i) = True \\ False & \text{Otherwise} \end{cases}$$

$\omega(x)$ does not depend only on the RDF triple x but it also depends on other RDF triples $(x_1, \cdots, x_n)$ that share the same subject of x and satisfy respectively the simple conditions $(\omega_1, \cdots, \omega_n)$. The condition ω is called the involved condition. In other words, the involved condition for an element x of E is the existence of other properties $\{p_i\}_{1\leq i\leq n}$ (predicates) of the subject of x and the value of each property p_i satisfies the simple condition ω_i.

In this section, we are interested in this kind of involved condition ω.

Algorithm 2 Let Bgp be a basic graph pattern, $\{s_i\}_{1\leq i\leq m}$ the set of subjects of the Bgp triple patterns and $\{Gp_i\}_{1\leq i\leq m}$ a set of group patterns [2] where Gp_i is a set of triple patterns of Bgp which has the same subject s_i. There are two cases to consider: $Permission(\omega)$ and $Prohibition(\omega)$.

Permission case: We handle each Gp_i separately from the others. For each $1 \leq j \leq n$, the subject s_i should have the property p_j such that its value should satisfy the simple condition ω_j. We verify if there exists a triple pattern $tp = (s, p, o)$ of Gp_i which has the property p_j $(p = p_j)$. If this triple exists, then it should satisfy the simple condition ω_j. For this purpose, we add a new SPARQL filter with the condition $\omega_j(tp)$. If there is no triple pattern with the property p_j on Gp_i then we create a new one $tp_{ij} = (s_i, p_j, ?\alpha_j)$ and we add it to Gp_i (where $?\alpha_j$ is a SPARQL variable). tp_{ij} should then satisfy the simple condition ω_j. So we add a new SPARQL filter with the condition $\omega_j(tp_{ij})$.

Prohibition case: In this case we verify for each $1 \leq j \leq n$ if there exists a triple pattern $tp = (s, p, o)$ of Gp_i with the property p_j $(p = p_j)$. If this pattern exists, then there are two cases. If its value 'o' is a SPARQL variable then it should not satisfy the condition ω_j or it should be unbound (i.e. s_i does not have the property p_j). In this case we add a new SPARQL filter with the condition $(\overline{\omega_j(tp)} \vee !bound(o))$. Otherwise, the value 'o' could not be unbound, then the triple pattern tp should not satisfy ω_j. In this case we add a new filter with the condition $\overline{\omega_j(tp)}$.

Now if there is no triple pattern with the property p_j on Gp_i, then we create a new one $tp_{ij} = (s_i, p_j, ?\alpha_j)$ and we add it to Gp_i. So tp_{ij} should not satisfy the condition ω_j or it should be unbound. In this case we add a new SPARQL filter with the condition $(\overline{\omega_j(tp_{ij})} \vee !bound(?\alpha_j)$. The expression $bound(variable)$ returns true if *'variable'* is bound to a value. It returns false otherwise [2].

Algorithm 3 Algo2 (Query, ω, ruleType). Query rewriting Algorithm for an involved condition

```
Require: $\omega$ is an involved condition
  for each basic graph pattern $Bgp$ of Query do
    Let $\{s_i\}_{1\leq i\leq m}$ bea set of the subjects of the triples pattern of $Bgp$
    for each subject $s_i$ do
      Let $Gp_i$ be a set of triple pattern of $Bgp$ with the same subject $s_i$
      $handleBGP(Bgp, Gp_i, \omega, ruleType)$
    end for
  end for
```

Example 6. Bob is a doctor and he can see only the information of his patients. The involved condition assigned (in the case of permission) to Bod could be expressed as:

$$(\forall x = (s,p,o) \in E)\ \omega(x) = \begin{cases} True & \text{if } (\exists y \in E) | y = \text{(s,pat:doctor,\$Bob\_id)} \\ False & \text{Otherwise} \end{cases}$$

where $Bob_id is the identifier of Bob. Bob tries to select names and locations of all patients. The table 2 shows Bob's query before and after transformation.

Before transformation	After transformation

SELECT ?name ?location
WHERE
{
    ?p rdf:type   pat:Patient.
    ?p foaf:name   ?name.
    ?p pat:location ?location.
}
	SELECT ?name ?location
	WHERE
	{
	    ?p rdf:type pat:Patient.
	    ?p foaf:name ?name.
	    ?p pat:location ?location.
	    ?p pat:doctor ?doct.
	    Filter(?doct=Bob_id)
	}

Table 2. Bob's query transformation

5.3 Composition of simple and involved conditions

Let **Algo** be a rewriting query algorithm which takes a query Q as inputs, condition ω and type of security rule (permission, prohibition) and returns a new query Q'. In the case of a permission rule, the execution result of the query Q', denoted RQ', is composed of elements of $I(\omega)$, i.e. the execution result of Q' satisfies the condition ω. If we suppose that RQ is the execution result of Q, then $RQ' = RQ \cap I(\omega)$ (Figure 2-A). In the case of prohibition rule, the execution result of the query Q' is composed of elements of $\overline{I(\omega)} = E \backslash I(\omega)$, i.e. $RQ' = RQ \cap \overline{I(\omega)} = RQ \backslash I(\omega)$ (Figure 2-B).

Algorithm 4 handleBGP (Bgp, Gp, ω, ruleType)

```
Require: $\omega$ is an involved condition, Gp is set of triples pattern of Bgp with same
  subject s
  $\{\omega_j\}_{1\leq i\leq n}$ a set of simple condition associated to $\omega$
  $\{p_j\}_{1\leq i\leq n}$ a set of predicates associated to $\omega$
  for $j = 1$ to $n$ do
    if $\exists tp = (s, p, o) \in Gp| \quad p = p_j$ then
      for each $tp = (s, p, o) \in Gp| \quad p = p_j$ do
        if $ruleType$= PERMISSION then
          add $FILTER(\pi_{\omega_j/tp})$ to $Bgp$
        else if $ruleType$= PROHIBITION then
          if o is SPARQL variable then
            add $FILTER(\overline{\pi_{\omega_j/tp}} \vee !bound(o))$ to $Bgp$
          else
            add $FILTER(\overline{\pi_{\omega_j/tp}})$ to $Bgp$
          end if
        end if
      end for
    else
      let $tp_j = (s, p_j, ?\alpha_j)$ be a triple of pattern
      add $tp_j$ to $Gp$
      if $ruleType$= PERMISSION then
        add $FILTER(\pi_{\omega_j/tp_j})$ to $Bgp$
      else if $ruleType$= PROHIBITION then
        add $FILTER(\overline{\pi_{\omega_j/tp_j}} \vee !bound(?\alpha_j))$ to $Bgp$
      end if
    end if
  end for
```

Composition in the case of permission Let ω_1 be a simple condition, ω_2 be an involved condition, ω the condition $\omega_1 \wedge \omega_2$ and ω' the condition $\omega_1 \vee \omega_2$. Let $Algo_1$ and $Algo_2$ be respectively the rewriting query algorithms of the simple conditions and involved conditions. Let Q, Q_1 and Q_2 be SPARQL queries, RQ, RQ_1 and RQ_2 be respectively the execution result of Q, Q_1 and Q_2 such that $Q_1 = Algo_1(Q, \omega_1, permission)$ and $Q_2 = Algo_2(Q_1, \omega_2, permission)$.
Logical AND: $\omega = \omega_1 \wedge \omega_2$ (Figure 3-A)
We have $RQ_2 = RQ_1 \cap I(\omega_2)$ and $RQ_1 = RQ \cap I(\omega_1)$ then $RQ_2 = RQ \cap I(\omega_1) \cap I(\omega_2)$. According to the result of theorem 2 we deduce that $RQ_2 = RQ \cap I(\omega_1 \wedge \omega_2) = RQ \cap I(\omega)$.

Thus we can use $Algo_1$ and $Algo_2$ to rewrite the query Q in order to satisfy the security rule $Permission(\omega) = Permission(\omega_1 \vee \omega_2)$. The rewriting query algorithm corresponding to this case is defined as follows:

$$Algo(Q, \omega_1 \wedge \omega_2, permission) = Algo_2(Algo_1(Q, \omega_1, permission), \omega_2, permission)$$

Example 7. Bob is permitted to select salaries of the network department employees. This rule could be expressed as $Permission(\omega) = Permission(\omega_1 \vee \omega_2)$

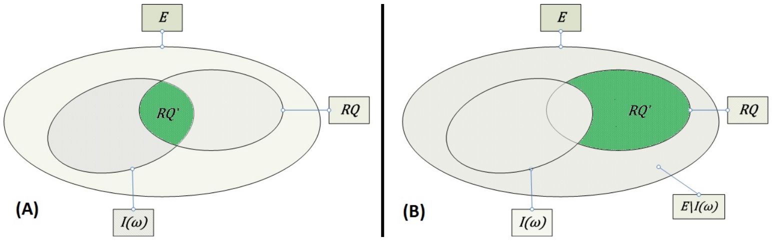

Fig. 2. (A) Permission case. (B) Prohibition case

where: $\forall x = (s, p, o) \in E\ \omega_1(x) = (p =\text{emp:salary})$

$$\omega_2(x) = \begin{cases} True & \text{if } (\exists y \in E)|y = \text{(s,emp:dept, 'Network')} \\ False & \text{Otherwise} \end{cases}$$

Logical OR: $\omega = \omega_1 \vee \omega_2$ (Figure 3-B)
Let $Q_1^{'}$ and $Q_2^{'}$ be SPARQL queries, $RQ_1^{'}$ and $RQ_2^{'}$ be respectively the execution result of $Q_1^{'}$ and $Q_2^{'}$ such that $Q_1^{'} = Algo_1(Q, \omega_1, permission)$ and $Q_2^{'} = Algo_2(Q, \omega_2, permission)$.
We have $RQ_2^{'} = RQ \cap I(\omega_2)$ and $RQ_1^{'} = RQ \cap I(\omega_1)$ then $RQ_1^{'} \cup RQ_2^{'} = (RQ \cap I(\omega_1)) \cup (RQ \cap I(\omega_2)) = RQ \cap (RQ_1^{'} \cup RQ_2^{'})$.
So $RQ_1^{'} \cup RQ_2^{'} = RQ \cap I(\omega_1 \vee \omega_2)$.

We deduce that the rewriting query Q_{final} corresponding to $Permission(\omega^{'})$ $=Permission(\omega_1 \vee \omega_2)$ is the union of the queries $Q_1^{'}$ and $Q_2^{'}$. So we can write $Q_{final} = Q_1^{'} \cup Q_2^{'}$ as well as

$$Algo(Q, \omega_1 \vee \omega_2, permission) = Algo_1(Q, \omega_1, permission) \bigcup Algo_2(Q, \omega_2, permission)$$

Example 8. Bob is permitted to select the employees salaries. He is also permitted to select all the information of the network department employees. This rule could be expressed as $Permission(\omega) = Permission(\omega_1 \vee \omega_2)$ where: $\forall x = (s, p, o) \in E$, $\omega_1(x) = (p =\text{emp:salary})$

$$\omega_2(x) = \begin{cases} True & \text{if } (\exists y \in E)|y = (s\text{,emp:dept, 'Network')} \\ False & \text{Otherwise} \end{cases}$$

Composition in the case of prohibition Let ω_1 be a simple condition, ω_2 be an involved condition. Let $Algo_1$ and $Algo_2$ be respectively the rewriting query algorithms of the simple condition and the involved condition. We use the same reasoning as in the previous section and by applying De Morgan's laws for sets, we obtain the following results:
Logical AND: $\omega = \omega_1 \wedge \omega_2$ (Figure 3-C)

$$Algo(Q, \omega_1 \wedge \omega_2, permission) = Algo_1(Q, \omega_1, permission) \bigcup Algo_2(Q, \omega_2, permission)$$

Logical OR: $\omega = \omega_1 \vee \omega_2$ (Figure 3-D)

$$Algo(Q, \omega_1 \vee \omega_2, permission) = Algo_2(Algo_1(Q, \omega_1, permission), \omega_2, permission)$$

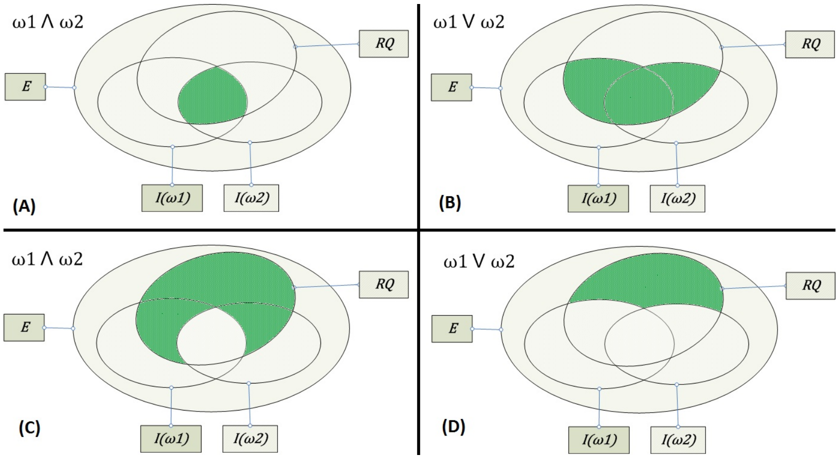

Fig. 3. (A) and (B) Permission case. (C) and (D) Prohibition case

6 Related Works

SPARQL is a recent query language. Even if there is a clear need to protect SPARQL queries, there is still no proposal to define an approach to evaluate SPARQL with respect to an access control policy.

If we now compare the approach suggested in this paper with SQL, we can observe that SQL security is based on view definitions. Using GRANT and REVOKE operators, one can specify which views a given user (or user role) is permitted to access. Transformation to apply security rules in SQL is based on a mechanism called view expansion. This mechanism is similar to macro expansion and consists in replacing a view by its definition when the query is evaluated. Thus, the initial query must only use authorized views, else the query is rejected.

An interesting variant to transform SQL queries was suggested by Oracle with its VPD [4] (Virtual Private Database) mechanism. In this case, the security policy is specified through the definition of predicates in PL-SQL that will apply as filters to transform the query. The general idea is similar to the one presented in this paper but the approach suggested by Oracle requires to know PL-SQL in order to implement the access control policy. This may lead to security policies complex to define and maintain.

Another interesting work was suggested by Stonebraker [5]. In this case, the query transformation is specified by adding conditions to qualification portion of the original query. The general idea is also similar to ours but the approach suggested by Stonebraker assumes that a similar mechanism would apply for *insert* and *update* operators, which is not generally true. A more recent approach was proposed by Wang et al. [6] where the objective is to securely maximize the answer provided to the user. This would represent a relevant extension to the work presented in this paper.

We can also compare our proposal with approaches to control access to XML documents. Two main approaches have been suggested in the literature: view materialization [7–12] and query transformation [13]. Most proposals are actually based on view materialization. In this case, for each user, the base of XML documents is transformed to extract the sub-part called the authorized view which is compliant with the access control policy. The query is then evaluated on the authorized view without modification. Unfortunately, it is generally considered that the view materialization process creates an intolerable overhead with respect to performance. Thus, more recent proposals suggest using query transformation, see for instance [13] that shows how to transform XPath queries.

However, there is a main difference between RDF and SPARQL: XML documents correspond to oriented graphs. As noticed in [12], this may lead to complication to protect some relationships in an XML document. This issue has been addressed using two different approaches: In [13], protection of XML relationships is embedded in document transformation whereas [12] suggests specifying access control policies using the concept of blocks in order to break some relationships that must be protected. We have no similar problems with RDF (or relational database). In our approach, every relationship may be protected using an involved condition filter.

7 Conclusion and Future Works

In this paper, we have defined an approach to protect SPARQL queries using query transformation. It is a generic approach to specify and apply an access control policy to protect RDF documents. An access control policy is modelled as a set of filters. A filter may be associated with a simple condition or an involved condition. Involved conditions provide means to protect relationships. In this paper, we consider two different types of filter: Positive filters corresponding to permission and negative filters corresponding to prohibition.

There are several possible extensions to this work. First, we only consider in this paper the case of *select* queries. There are some recent proposals to extend SPARQL to specify queries for updating RDF documents. Thus, an interesting extension of our work would be to also consider how to transform update queries with respect to an access control policy.

Second, in this paper, the access control policy is specified through a set of filters. This provides a generic approach to represent an access control policy for RDF documents which does not rely on a specific language. However and as suggested in section 2, a possible extension would be to define a user friendly specification language to express such an access control policy. For this purpose, a possible research direction would be to derive the filter definition from the specification of an access control policy based on RBAC [14] or OrBAC [15].

Finally, in the near future we intend to integrate other security related transformations in the policy specification, for instance anonymisation. Also we need to integrate our approach into service composition management.

References

1. Klyne, G., Carroll, J.: Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
2. PrudHommeaux, E., Seaborne, A.: Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/ (January 2008)
3. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. Proc. ACM Sigmod Conf. (June 2004)
4. Huey, P.: Oracle database security guide : Chapter 7, using oracle virtual private database to control data access. http://download.oracle.com/docs/cd/E11882_01/network.112/e10574.pdf
5. Stonebraker, M., Wong, E.: Access control in a relational data base management system by query modification. Proceedings of the 1974 annual conference (June 1974) 180–186
6. Wang, Q., Yu, T., Li, N., Lobo, J., Bertino, E., Irwin, K., Byun, J.: On the correctness criteria of fine-grained access control in relational databases. Proceedings of the 33rd international conference on Very large data bases (September 2007)
7. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for xml documents. ACM Trans. Inf. Syst. Secur. **vol. 5(2)** (2002) 169–202
8. Gabillon, A.: A formal access control model for xml databases. In Proc. Of the 2005 VLDB Workshop on Secure Data Management (SDM) (2005)
9. Finance, B., Medjdoub, S., Pucheral, P.: The case for access control on xml relationships. Proc. of CIKM (2005)
10. Kudo, M., Hada, S.: Xml document security based on provisional authorization. Proc. of ACM CCS (2000)
11. Stoica, A., Farkas, C.: Secure xml views. Proc. of the 16th IFIP WG11.3 Working Conference on Database and Application Security (2002)
12. Cuppens, F., Cuppens-Boulahia, N., Sans, T.: Protection of relationships in xml documents with the xml-bb model. In Proc. of ICISS2005 (2005)
13. Damiani, E., Fansi, M., Gabillon, A., Marrara, S.: A general approach to securely querying xml. In Proc. of the 5th International Workshop on Security in Information Systems (WOSIS 2007) (2007)
14. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and Systems Security (TISSEC) **4**(3) (2001)
15. Abou El Kalam, A., El Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G.: Organization Based Access Control. In: 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy (June 2003)