Universidad de Concepción
Dirección de Postgrado
Facultad de Ingeniería - Programa de Magíster en Ciencias de la Computación

Control de Acceso Basado en Roles para RDF
(Role Based Access Control for RDF)

Tesis para optar al grado de Magíster en Ciencias de la Computación

JORGE ANDRÉS MIRANDA VARGAS
CONCEPCIÓN, CHILE
2015

Profesora Guía: Loreto Bravo Celedón
Departamento de Ingeniería Informática y
Ciencias de la Computación, Facultad de Ingeniería
Universidad de Concepción

Acknowledgments

I would like to thank Fondecyt Project N°1130902 for funding this work.

Table of Contents

List of Figures

Abstract

In order to control the access to the information available on the Semantic Web, different access control mechanisms have been proposed. As RDF is the standard format for publishing data on the Semantic Web, some of these mechanisms are able to enforce access restrictions on the granularity level of triples. However, there exists the need for a finer-grained access control that manages the access to the subject, predicate and object of the triples, since there exist information requests that could be satisfied with only a part of a triple, while still prohibiting the access to sensible data. The purpose of this thesis is to formalize an access control policy model for RDF data, for which the smallest unit of protection are the parts of a triple, this is, its subject, predicate and object. First, we present the syntax and semantics of our policies, which are based on permissions. A permission consists of two parts: an `APPLY` statement that specifies the parts of the triples managed by the permission and a query `SELECT`-`WHERE` that determines the triples to which the permission applies. The set of allowed and forbidden permissions will respectively grant or deny access to data. In case of conflict deny overwrites an allow permission. Then, we study under what conditions the leaks of hidden information are possible for our policies. We found that our policies are consistent, which means that the answers of a series of queries, performed on a graph, cannot be combined to safely obtain triples that are hidden. After that, we propose an algorithm to enforce the policies defined with our model, in which the process of computing the answers to queries is divided in three parts: obtaining for each triple the permissions that apply on it, anonymizing the graph according to the permissions, and performing the queries over the anonymized graph. The analysis of the running time shows that enforcing a policy is done in polynomial time over the size of the graph under data complexity, and the space analysis shows that the maximum space that will be used to store one anonymized graph for each role is twice the size of the original graph. Finally, considering these results and the fact that RDF graphs can be large enough that storing multiple copies of the same information becomes infeasible for multiple roles, we propose an algorithm that can store the anonymized data in a more compact way. Our time cost analysis concludes that enforcing the policies for multiple roles is still done in polynomial time over the size of the graph, even though it is slightly more inefficient than the first algorithm. It shows too that the maximum space that will be used to store the anonymized

data is independent from the number of roles, and it is eight times the size of the original graph.

Chapter 1

Introduction

The Web of Data, proposed by Tim Berners-Lee in 2006, is based on the idea of publishing and linking data in the same way documents are being published and linked on the internet [BL09]. For this, the RDF data format [Gro14b] has been established as the standard format for publishing information on the internet with over 52 billion of RDF triples as of March of 2012 [wLO14]. The information that can be found in the Web of Data belongs to several areas such as biology, geography, literature, music, movies, tourism and government data. The Web of Data has its own Wikipedia in the form of the DBPedia, which basically publishes the same data but in the RDF format, making it one of the biggest sources of RDF data and the most connected to the rest of the Web.

This Semantic Web is structured in a way that is easily processed by computers, unlike the Web of Documents that has been developed for ease of use by people. One of the advantages of using RDF to publish data on the internet is that we get the possibility to ask questions about the information by using structured query languages such as SPARQL [PS08]. For example, if a student is trying to compare different university programs, and he wants to know their cost and if the program is accredited, he could normally get all the data he needs by searching through several web pages and collecting the information from them, while on the Web of Data he could perform queries to obtain directly the information he needs.

In order to allow companies and institutions to publish data on the RDF format for use on the Semantic Web, it is necessary to address the problem of controlling the access to the data, ensuring private data to be accessible only to authorized users. The development of a powerful and flexible access control system is fundamental for data publishers who want to enforce specific requirements on the access to their data.

There are several types of access control models that are based on identity, roles, clearance or attributes [PSA01; PS99; YT05; BCEPM08]. For this work, we are adopting a

role based access control architecture [PSA01; PS99] and we are assuming that the proper mechanisms for authentication of the users exist.

We say that an access control mechanism that manages the access to a data set in large data chunks provides a coarse-grained level of control. In contrast, if the access control mechanism handles individually the access to each data in the data set, we are in presence of a fine-grained level of control. So, if we want to comply with the principle of least privilege, our access control mechanism must allow the possibility to provide to each user the minimal permissions needed to perform their tasks, i.e., our mechanism must provide a fine-grained access control.

Research on access control for RDF is scarce. General frameworks for RDF access control are given in [PSA01; DA06; ACH+07]. It has also been proposed, by the authors of [CVDG12], an access control framework that considers factors related to the access from mobile devices. However, none of them provided formal semantics for the policies or enforcement techniques. Only in the work presented in [FFMA10] the access control policy is formalized and an enforcement mechanism is implemented and tested. Their work is based on permissions that are described using triple patterns, that either allow or forbid the access to specific triples. Thereby, only the triples that are available for a user are used to compute the answer to his queries, due to the fact that each triple will be annotated with permissions. Even though triples are the smallest unit of information of RDF, we believe that an access control mechanism of a finer-grained level is possible, since the parts of a triple still contain useful information.

There is plenty of research about access control policies for other types of database models [CFMS94]. Access control policies that specify permissions to allow users to read links and to browse through them have been studied in the context of XML documents [BCFM00] and unstructured HTML documents [SBJ96]. A European project called FASTER [Sam02] was devoted to design an access control system for Web-publishing of statistical data. The results of the project are used by a commercial partner of the project (Nesstar [wNE12]) and are not freely available.

This thesis focuses on the development of a fine-grained role based access control policy for RDF data. Our access control policy will consist on sets of allowed and forbidden permission. These permissions are based on queries and include information about the parts of the RDF triples over which access is being controlled. We will define syntax and

semantics of the policy model from which we will propose an algorithm to enforce the rules established by a policy. We will detail the design considerations that were made during the creation of our policy model and enforcement algorithm. And then, we will determine the algorithm's running time and space cost. We propose a different algorithm to enforce policies in the presence of multiple roles, one that could potentially outperform the first algorithm in aspects like the quantity of the space utilized.

Chapter 2

Preliminaries

In this section we provide some concepts of access control, the RDF data model and its SPARQL query language.

2.1 Access Control

There are multiple security mechanisms [FKC07; SS94] to satisfy the need to protect sensible data on computer systems. Authentication, auditing and access control are security needs that help to protect data from possible leaks. Ideally, there will be a clear separation between the three, but sometimes they are handled as one or one or more of them could not be present.

Authentication deals with the task of determining the correct identity of a user, some forms of authentication include the use of user identifiers (IDs) and passwords, smart cards, and fingerprints readers. *Auditing* is the analysis of users' behavior in a system, so that attempted and actual violations to the security system are detected, as well as possible flaws in the security system. To perform the analysis, all requests and activities of users in the system need to be stored in logs, which also serve to help to discourage users from attempting violations on the system and misusing their privileges. *Access control* is the selective restriction of the actions and operations a user can perform in a computer system. The focus of this thesis will be on access control.

Determine if a user has the right to use a resource, maintain private information safe from the access of unauthorized users, and protect data from being altered by unscrupulous users are tasks in the scope of access control. Access control allows the possibility to read the data only to authorized users, tackling security risk related to confidentiality. Access control helps to maintain the integrity of the information in a system, guaranteeing that its modification is performed only by users accredited to do it and that they are doing it through

authorized ways.

An access control system has three levels of abstraction: access control policies, access control models, and access control mechanisms. *Access control policies* are high-level requirements that specify how access is managed and who, under what circumstances, may access what information. *Access control mechanisms* are low-level software and hardware functions that can be configured to implement a policy, that is to say, the mechanism is in charge of the enforcement of the policy. *Access control models* are a way to describe the properties and requirements of an access control system. These models are the link connecting policies and mechanisms.

Access control policies can be classified as Discretionary Access Control (DAC) policies, Mandatory Access Control (MAC) policies and Role Based Access Control (RBAC) policies. Access restriction to the resources on DAC policies are based on users' identity and/or the identity of the groups they belong. In DAC policies, the rules are stated for each user, or group, and for each resource, so all requests to access a certain resource made by any user have to be checked against the specified authorization rules. The flexibility of DAC policies is the main advantage of this type of access control, and its main disadvantage is that any users that has access to a resource can pass the information contained on it to unauthorized users.

In MAC policies, security levels are assigned to each user and resource of the system. Security levels are hierarchical, for example, starting on the bottom of the hierarchy we could have the levels Unclassified (U), Confidential (C), Secret (S) and Top Secret (TS). The security level of a user is called the clearance, and it reflects how much trust can be put on him in regard to not disclose information to users not enabled to see the information. To avoid leaking information to bottom levels of security, a user can start sessions at lower levels than his clearance level. For example a user with authorization level S can start sessions at the levels of security S, C and U. Access to resources are granted according to the *simple security property* and *star property*. The former property states that if the user's security level dominates the security level of the resource, the user is allowed to read it. The latter states that if the user's security level is dominated by the security level of the resource, the user is allowed to write on it. Thanks to the simple security property, users are not able to read information above their clearance level, while the star property prevents the flow of information from high to low levels by not letting users to write on levels lower

than the one of his current session.

On top of this, users and resources can be classified on categories, adding the possibility of allowing access only when user and resource have at least one category in common.

To ensure information integrity, MAC can be applied to resources. For example, the levels Crucial (C), Important (I), and Ordinary (O) could be the integrity levels, with C > I > O. The integrity level of a resource suggests how sensitive it is to modifications while the integrity level of a user shows the level of trust given to him to modify the data in that level. As in the previous case, the access to the resources in the system will be managed according to two properties. The *simple integrity property* allows users to read a resource only if its security level is dominated by the security level of the resource, while the *integrity star property* allows users to write on a resource only if its security level dominates the security level of that resource. As a result, the users cannot write from lower levels, nor read resources whose security level is lower than theirs.

The main advantage of the MAC model is that it allows us to define policies that control the flow of information in the database, which is achieved by forbidding the reading/writing of data based on trust levels and categories. MAC's main disadvantage is that policies defined with this model are difficult to manage when the number of users and resources is large or when the relations between them are constantly changing, making it prone to leaks.

In RBAC policies, permissions over the resources of the system are assigned to the defined roles and each user is assigned to one or more roles depending on the tasks he has to do. This type of policy is useful when the system is used by a big quantity of users, as in a large company, since using a simple access control mechanism, to manage who has rights over which resource, can be quite complicated and prone to leaks of information, considering that users could be constantly leaving and entering the company or performing new assignments inside it. In an organization, users can change faster than the roles they perform, so it is reasonable to associate the permissions over the resources of the system to the roles instead of the users, now when a user leaves or joins the company or changes position within it, he is going to have his old role and permissions revoked and a new one, with its corresponding permissions, will be assigned to him.

The main advantage of this model is that the authorization management is highly simplified by breaking the task in two parts, assign roles to users and giving access rights for resources

to roles. Some other advantages of this model is the *support of hierarchical roles*, simplifying further the authorization management; *least privilege*, by allowing users to sign on with roles with powerful privileges only when needed. Also constraints like *separation of duties*, preventing users to execute more than one role in order to perform actions that requires more than one person to be executed. The main disadvantage of the RBAC model is the lack of flexibility, this mean that we cannot establish additional permissions for specific users.

The management of roles in RBAC is different from the one for groups in DAC policies. In DAC, a user could have access to a resource directly and indirectly, if he has rights over the resource explicitly and he is part of a group with rights over that resource. In RBAC, a user can only have rights over a resource by means of a role. Thus, if we want to forbid the access of that user to the resource, for the case of RBAC, we only need to remove the role granting him with the access right from his roles, while in DAC, we need to change the access rights of the user and each of its groups, with respect to the resource, to forbidden.

We will deal with RBAC read operations and, as stated earlier, we assume proper authentication methods to ensure that users belong to specific roles. We chose RBAC over the others because it is the most suited to help companies to adopt the RDF data format.

2.2 RDF

RDF stands for Resource Description Framework and is a data model composed by resources, properties and statements. A resource is anything that has an identity, and they are identified through Uniform Resource Identifiers (URI). Some examples of URIs are:

- file:///home/username/Document.pdf
- urn:isbn:1-59693-113-2
- doi:10.1000/182
- ftp://asmith@ftp.example.org

RDF properties are specific aspects, characteristics, attributes or relations of a resource. Each property is a resource, has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties. An *RDF statement*, or *triple*, consists of three parts: a subject, a predicate and an object. The subject can be

any resource, the predicate is a property of that resource and a resource by itself, and the object is a resource or the value of that property. Examples of statements and its parts are presented in Figure 2.1. In Figure 2.1, all subjects, predicates and objects are resources, i.e., they represent URIs. For example, Dave stands for http://www.example.com/Dave. In general, URIs are represented in ovals, while objects that are values, formally called literals, are represented in squares in the graph.

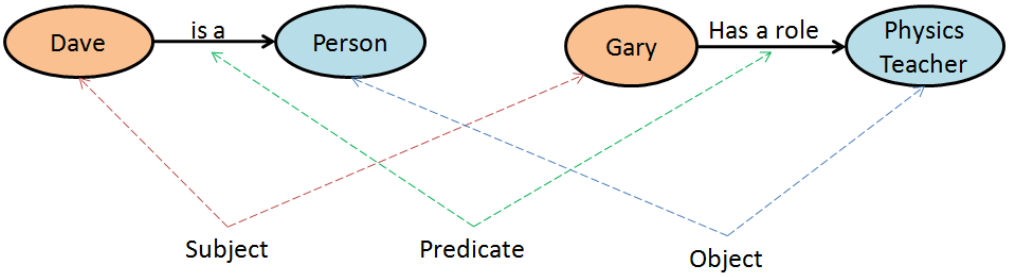

Figure 2.1: RDF statements.

Formally, a triple t is of the form $(\mathsf{s}, \mathsf{p}, \mathsf{o})$, where s is the subject, p is the predicate and o is the object, with $\mathsf{s}, \mathsf{p} \in U$ and $\mathsf{o} \in U \cup L$, where U is an infinite set of Universal Resource Identifiers (URIs) and L is an infinite set of Literals. To refer to the value of the subject, predicate and object of a triple t, we are going to use $t[\mathsf{s}]$, $t[\mathsf{p}]$ and $t[\mathsf{o}]$, respectively.

A set of RDF triples is called an RDF Graph, RDF data set or RDF store. The graph $\mathcal{G}_1$ shown in Figure 2.2 is an example of an RDF graph, and will be used as an ongoing example. For this purpose, we labeled the triples $t_1 \ldots t_{22}$ to reference them through out the document. We can also represent it in a node diagram as in Figure 2.3 for ease of understanding.

2.3 SPARQL

SPARQL is a query language used primarily to obtain specific data from data sets in the RDF format. The recursive acronym SPARQL stands for SPARQL Protocol and RDF Query Language and one of its features is that the rules for the exchange of SPARQL queries and results, between the client program and the SPARQL processing server, are defined in their own document [DuC11].

	subject(s)	predicate(p)	object(o)
t_1	foaf:Person	rdfs:subClassOf	foaf:Agent
t_2	ex:Student	rdfs:subClassOf	foaf:Person
t_3	ex:Teacher	rdfs:subClassOf	foaf:Person
t_4	ex:area	rdfs:domain	ex:Teacher
t_5	ex:area	rdfs:range	rdfs:Literal
t_6	ex:collaborateWith	rdfs:domain	ex:Student
t_7	ex:collaborateWith	rdfs:range	ex:Student
t_8	ex:completedProject	rdfs:domain	ex:Student
t_9	ex:completedProject	rdfs:range	rdfs:Literal
t_{10}	foaf:firstName	rdfs:domain	foaf:Person
t_{11}	foaf:firstName	rdfs:range	rdfs:Literal
t_{12}	&a	rdf:type	ex:Student
t_{13}	&a	foaf:firstName	William
t_{14}	&a	ex:completedProject	5
t_{15}	&b	rdf:type	ex:Student
t_{16}	&b	foaf:firstName	Emma
t_{17}	&b	ex:collaborateWith	&a
t_{18}	&b	ex:completedProject	1
t_{19}	&c	rdf:type	ex:Teacher
t_{20}	&c	foaf:firstName	Allen
t_{21}	&c	ex:completedProject	20
t_{22}	&c	ex:area	Physics

Figure 2.2: Graph $\mathcal{G}_1$.

In this work we will focus on conjunctive SPARQL queries (CQ). These queries are composed by two parts, a `SELECT` statement describing the data variables we want, and a `WHERE` statement using triple patterns to describe the conditions the data needs to meet. Triple patterns, defined as in [PS08], are used to express groups of triples that conform to a particular form.

Let $\mathcal{V}$ be an infinite set of variables such that $\mathcal{V} \cap U = \varnothing$ and $\mathcal{V} \cap L = \varnothing$, and the variables always start with the symbol $?$, e.g. $\mathcal{V} = \{?x, ?y, ?z, ?x_2, \dots\}$.

Definition 2.1 A triple pattern tp is of the form $tp = (p_1, p_2, p_3)$, where $p_1, p_2 \in \mathcal{V} \cup U$ and $p_3 \in \mathcal{V} \cup U \cup L$. □

Intuitively, a triple pattern has the same form of a triple, with the difference that the subject, predicate and object can also be a variable. For example, using the variables

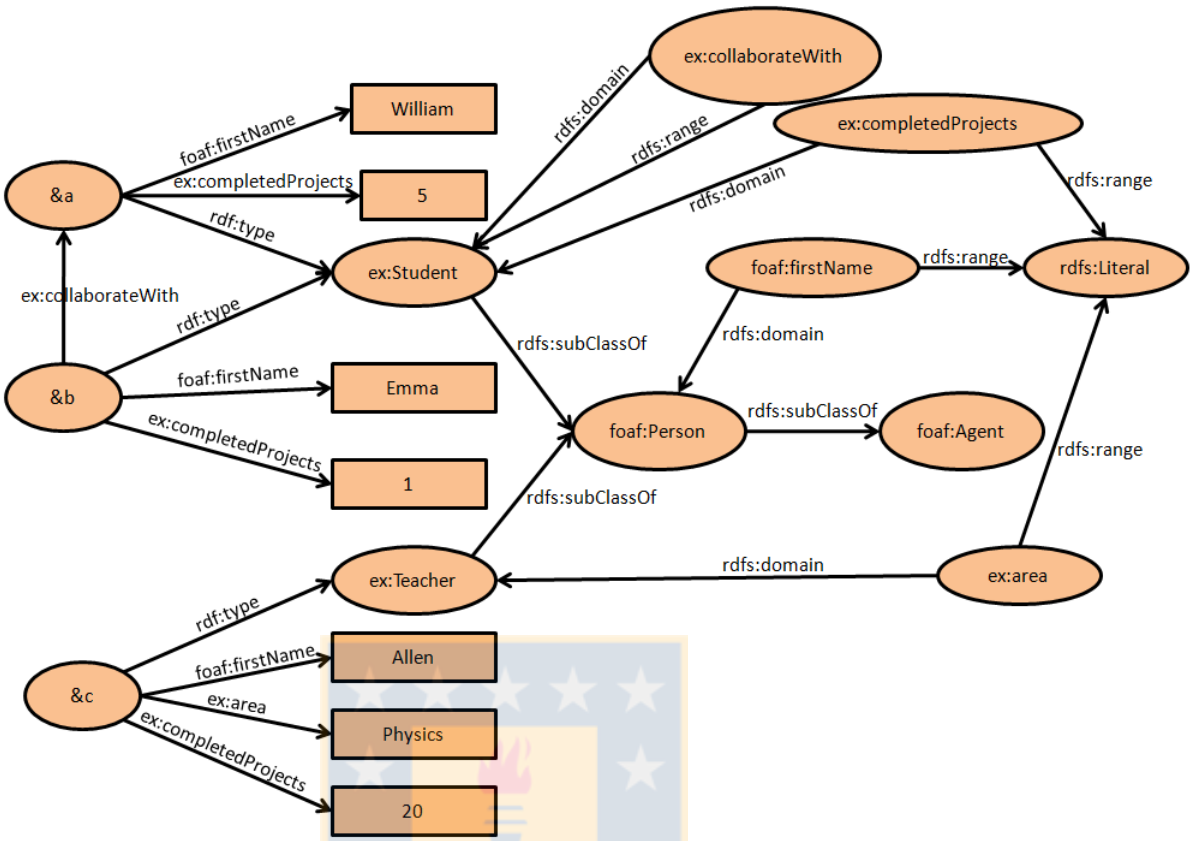

Figure 2.3: Node diagram of graph $\mathcal{G}_1$.

$?x$, $?z$, $?u$ and $?s$, we can define the triple patterns $tp_1 = (?x, \text{foaf:firstName}, ?z)$, $tp_2 = (?u, ?s, \text{ex:Student})$, $tp_3 = (?s, \text{ex:area}, \textit{Physics})$.

Definition 2.2 A conjunctive query (CQ) is of the form

$$\texttt{SELECT}\ ?x_1, ?x_2, \ldots, ?x_n\ \texttt{WHERE}\ \mathcal{TP}, \mathcal{C}$$

where (i) $?x_1, ?x_2, \ldots, ?x_n \in \mathcal{V}$ and they appear in $\mathcal{TP}$, (ii) $\mathcal{TP}$ is a conjunction of triple patterns and (iii) $\mathcal{C}$ is an optional conjunction of constraints of the form $?u\ op\ c$ where $?u \in \mathcal{V}$ and it appears in $\mathcal{TP}$, $op \in \{<, >, \leq, \geq, =, \neq\}$ and $c \in \mathcal{V} \cup U \cup L$.

□

For simplicity, we are restricting the form of the constraints in $\mathcal{C}$, but there are other types of constraints that allow more complex queries. For example, we can ask if a variable is mapped to a boolean value, numeric value or string, etc. To perform these complex queries

we need to include more built-in SPARQL functions. For a complete list of SPARQL functions see [SHP13].

Example 2.1 Consider the following queries and graph $\mathcal{G}_1$ from Figure 2.2:

The query to ask for the names of all people in the graph is `SELECT` $?z$ `WHERE` (&a, foaf:firstName, $?z$) and the answer is $?z$ = {*William*, *Emma*, *Allen*}. If we want to ask for the names of the people who have completed five or more projects, we use the query `SELECT` $?z$ `WHERE` ($?x$, foaf:firstName, $?z$), ($?x$, ex:completedProject, $?w$), $?w \geq 5$, and the answer to this is $?z$ = {*William*, *Allen*}. To ask what types of persons are present in graph $\mathcal{G}_1$, we perform the query `SELECT` $?v$ `WHERE` ($?v$, rdfs:subClassOf, foaf:Person), that gets for answer $?v$ = {ex:Student, ex:Teacher}. □

In order to formally define the answers to a query CQ we need to define a mapping μ that assigns values to variables in $\mathcal{V}$. Mapping $\mu : \mathcal{V} \cup U \cup L \rightarrow U \cup L$ where $\mu(a) = a$ for every $a \in U \cup L$. To simplify presentation we will write μ as a set of pairs so that for every variable $?x$, if $\mu(?x) = u$ then $(?x, u) \in \mu$. We will also omit the mapping for elements in $U \cup L$ which map to themselves. For example, if $\mu(?x)$ = &a, $\mu(?y)$ = foaf:firstName, $\mu(?z)$ = *William*, then its set representation would be $\mu = \{(?x, \text{\&a}), (?y, \text{foaf:firstName}), (?z, \textit{William})\}$, and if $\mu(?u)$ = &b, μ(rdf:type) = rdf:type, $\mu(?v)$ = ex:Student, then its set representation would be $\mu = \{(?u, \text{\&b}), (?v, \text{ex:Student})\}$. Given a triple pattern $tp = (p_1, p_2, p_3)$, we denote by $\mu(tp)$ the result of applying the mapping to each element, i.e., $\mu(tp) = (\mu(p_1), \mu(p_2), \mu(p_3))$.

We are also interested in the triples in a graph that match a conjunction of triple patterns. Given a graph $\mathcal{G}$ and a conjunction of triple patterns $\mathcal{TP} = \{tp_1, ..., tp_n\}$, we will denote by $\langle\langle \mathcal{TP} \rangle\rangle_{\mathcal{G}}$ the set of mappings that if applied to each tp in $\mathcal{TP}$ result in triples in $\mathcal{G}$. More formally: $\langle\langle \mathcal{TP} \rangle\rangle_{\mathcal{G}} = \{\mu |$ for all $tp \in \mathcal{TP}, \mu(tp) \in \mathcal{G}\}$.

Example 2.2 Consider graph $\mathcal{G}_1$, the triple patterns tp_1 = (&a, $?y$, $?z$), tp_2 = ($?x$, rdf:type, ex:Student), tp_3 = ($?x$, ex:collaborateWith, $?z$) and the conjunctions of patterns $\mathcal{TP}_1$ = $\{tp_1\}$, $\mathcal{TP}_2$ = $\{tp_2\}$, $\mathcal{TP}_3$ = $\{tp_3\}$ and $\mathcal{TP}_4$ = $\{tp_2, tp_3\}$. The set of mappings that match these conjunctions of patterns are:

$\langle\!\langle \mathcal{TP}_1 \rangle\!\rangle_{\mathcal{G}_1}$ = $\{\{(?y, \text{rdf:type}), (?z, \text{ex:Student})\}, \{(?y, \text{foaf:firstName}), (?z, \textit{William})\},$
$\{(?y, \text{ex:completedProject}), (?z, \textit{5})\}\}$
$\langle\!\langle \mathcal{TP}_2 \rangle\!\rangle_{\mathcal{G}_1}$ = $\{\mu(?x, \text{rdf:type}, \text{ex:Student})\}$ = $\{\{\mu(?x), \mu(\text{rdf:type}), \mu(\text{ex:Student})\}\}$ =
$\{\{(?x, \text{\&a})\}, \{(?x, \text{\&b})\}\}$
$\langle\!\langle \mathcal{TP}_3 \rangle\!\rangle_{\mathcal{G}_1}$ = $\{\mu(?x, \text{ex:collaborateWith}, ?z)\}$ = $\{\mu(?x), \mu(\text{ex:collaborateWith}), \mu(?z)\}$ =
$\{\{(?x, \text{\&b}), (?z, \text{\&a})\}\}$
$\langle\!\langle \mathcal{TP}_4 \rangle\!\rangle_{\mathcal{G}_1}$ = $\{\{(?x, \text{\&b}), (?z, \text{\&a})\}\}$.

□

Let μ be a mapping and $c = ?u\ op\ w$ a constraint in $\mathcal{C}$ where $?u$ is a variable; w is a variable, a URI or a Literal; and $op \in \{<, >, \leq, \geq, =, \neq\}$. If $\mu(?u)\ op\ \mu(w)$ is true, we say that μ satisfies $?u\ op\ w$. Therefore, for a conjunction of constraints $\mathcal{C}$ and a mapping μ, if each $?u\ op\ w \in \mathcal{C}$ is satisfied by μ, then $\mathcal{C}$ is satisfied by μ. For example, the constraint $c_1 = ?z < 10$ is satisfied by the set of mappings $\langle\!\langle \mathcal{C} \rangle\!\rangle_{\mathcal{G}_1} = \{\mu_1, \mu_2\}$ = {{($?x$, &a), ($?y$, ex:completedProject), ($?z$,5)}, {($?x$, &b), ($?y$, ex:completedProject), ($?z$,1)}}. Note that the operators <, >, ≤, ≥, =, ≠ follow the rules for mapping defined for SPARQL [PS08]. This means that when the variables $?u$ and w are mapped to different types of resources, they will be incompatible and do not satisfy the constraint, e.g. $?z$ = &a does not satisfy the constraint c_1.

Given a graph $\mathcal{G}$, a conjunction of triple patterns $\mathcal{TP}$ and a conjunction of constraints $\mathcal{C}$, we will denote by $\langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}}$ the set of mappings that match every triple pattern in $\mathcal{TP}$ and for each $c \in \mathcal{C}$, μ satisfies c. More formally: $\langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}} = \{\mu |$ for all $tp \in \mathcal{TP}, \mu(tp) \in \mathcal{G}$ and μ satisfies $\mathcal{C}\}$.

For example, if we have $\mathcal{TP}_5$ = {($?x$, ex:completedProject, $?z$)} and $\mathcal{C}_1$ = {($?z < 10$)}, $\langle\!\langle \mathcal{TP}_5, \mathcal{C}_1 \rangle\!\rangle_{\mathcal{G}_1}$ = {{($?x$, &a), ($?z$,5)}, {($?x$, &b), ($?z$,1)}}.

Definition 2.3 Given a graph $\mathcal{G}$ and a query CQ = `SELECT` $?x_1, ?x_2, \ldots, ?x_n$ `WHERE` $\mathcal{TP}, \mathcal{C}$. The answer to query CQ is the set $\mathbf{Ans}(\text{CQ}, \mathcal{G}) = \{\mu \mid \mu \in \langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}}\}$

□

Example 2.3 Consider a query CQ_1 that asks for the rdfs:literals on graph $\mathcal{G}_1$ from Figure 2.2, a query CQ_2 that asks for the name of people on the graph with their corresponding rdf:type, and a query CQ_3 that asks for William's information. These queries and their respective answers are shown below.

CQ_1 = `SELECT` $?z$ `WHERE` $(?x, ?y, ?z)$, $(?y, ?u, ?v)$, $?v$ = rdfs:Literal

$\mathbf{Ans}(\mathrm{CQ}_1, \mathcal{G}_1)$ = $\{\{(?z, \textit{Physics})\}, \{(?z, \textit{5})\}, \{(?z, \textit{1})\}, \{(?z, \textit{20})\}, \{(?z, \textit{William})\}, \{(?z, \textit{Emma})\}, \{(?z, \textit{Allen})\}\}$

CQ_2 = `SELECT` $?x, ?y$ `WHERE` $(?z, \text{foaf:firstName}, ?y), (?z, \text{rdf:type}, ?x)$

$\mathbf{Ans}(\mathrm{CQ}_2, \mathcal{G}_1)$ = $\{\{(?x, \text{ex:Student}), (?y, \textit{William})\}, \{(?x, \text{ex:Student}), (?y, \textit{Emma})\}, \{(?x, \text{ex:Teacher}), (?y, \textit{Allen})\}\}$

CQ_3 = `SELECT` $?y, ?z$ `WHERE` $(?x, ?y, ?z), (?x, ?v, ?w), ?w$ = *William*

$\mathbf{Ans}(\mathrm{CQ}_3, \mathcal{G}_1)$ = $\{\{(?y, \text{ex:completedProject}), (?z, \textit{5})\}, \{(?y, \text{rdf:type}), (?z, \text{ex:Student})\}\}$

□

2.4 RDFS

In the graph $\mathcal{G}_1$ shown in Figure 2.2 we can see that some predicates include the prefix `rdfs:`. This prefix references the RDF Schema (**RDFS**) [DBM14] vocabulary, which is expressed in RDF and provides the necessary mechanisms to describe groups of resources and the relationships between them, allowing us to infer new triples that are not included in an RDF graph, but are implicit.

In RDFS, resources can be grouped in classes. These classes are defined in RDF with triples of the form $(\mathsf{s}, \texttt{rdf:type}, \mathsf{o})$, where the predicate `rdf:type` states that the resource in the subject is an instance of the class in the object of the triple. Subclasses can be specified with triples of the form $(\mathsf{s}, \texttt{rdfs:subClassOf}, \mathsf{o})$, where the predicate `rdfs:subClassOf` states that the subject s is a subclass of the class o. In the same way, subproperties can be specified with the predicate `rdfs:subPropertyOf`.

RDFS let us explicitly state what types of resources the properties can affect, which is achieved with the predicates `rdfs:domain` and `rdfs:range`. When a resource appears in a triple as a predicate, the subject of that triple has to be an instance of the class related by `rdfs:domain`, and the object has to be an instance of the class related by `rdfs:range`.

For example, considering the graph in Figure 2.2, the predicate `foaf:firstName` can be present in t_{13} because the subject of the triple is an instance of a subclass of `foaf:Person`,

and we can assume that the object of the triple is a Literal. Some other data that we can obtain, which is not explicitly included in the graph, are the triples (&a, rdf:type, foaf:Person), (&b, rdf:type, foaf:Person), (&c, rdf:type, foaf:Person).

RDFS inference is the process of deducing new information from an RDF graph. The inferred triples are obviously not part of the graph, however, they are valid and could be used by users to obtain information that has been hidden to them, if the access control mechanism does not enforce the restrictions on the inferred triples. E.g. if we hide the triple (&c, rdf:type, ex:Teacher) from the graph $\mathcal{G}_1$, but we show triples (&c, ex:area, *Physics*) and (ex:area, rdfs:domain, ex:Teacher), we can infer the hidden triple because only instances of the class ex:Teacher can be subjects in a triple whose predicate is ex:area.

In addition to RDFS, there exist other vocabularies and ontologies that help people to infer information from RDF graphs, e.g., OWL.

Chapter 3

Related Work

There is not a lot of work focused on access control for RDF. In most of them the smallest unit of protection is the RDF triple. Thus, in order to let users access to information about a determined resource, this resource has to be part of an allowed triple. By restricting access at triple level we are losing the possibility of answering queries that could be replied only with a part of a triple, i.e., with only the subject, predicate or object.

Consider the queries 1 and 2 as described below, if we want to allow users to retrieve the id of students with phone numbers associated to them, but we want to forbid the access to the number, we should allow query 1 to be answered, while query 2 should be forbidden and should give no results.

#Query 1	#Query 2
SELECT $?id$ WHERE ($?id$, :phoneNum, $?phone$)	SELECT $?id$, $?x$, $?phone$ WHERE ($?id$, $?x$, $?phone$), $?x$ = :phoneNum

Under a policy that controls the access to data at triple level, like the one in [FFMA10], to deny access to phone numbers, we need to forbid access to triples of the form ($?id$, :phoneNum, $?phone$) and, therefore, both queries would provide no results. Thus, policies that possess the capability to control access at a finer level could be used in this type of situation.

In the work of Abel et al. [ACH+07], the access control is handled on a layer built on top of the RDF store, in such a way that it is portable across different RDF graphs. The enforcement of policies is through query rewriting, so that queries cannot access nor return forbidden triples. They limit the access at triples level by defining policy rules of the form $pred(triple(\mathsf{s},\mathsf{p},\mathsf{o})) \leftarrow cp_1, ..., cp_n, pe_1, ..., pe_m, be_1, ..., be_p$. The access to triples matching the path expression $triple(\mathsf{s},\mathsf{p},\mathsf{o})$, where s, p and o are URIs or variables and o can also be a literal, will be determined according to if $pred$ is $allow$ or $disallow$. To express

contextual conditions that are not related to the data, like time, location, properties of the requester, etc; the contextual predicates cp_i are used. The path expressions pe_i and boolean expressions be_i are used to specify graph patterns that the RDF graph needs to meet. They deny the access to triples that have no policy rules being applied to them, as well as denying the access to the triples that have $deny$ and $allow$ policy rules being applied to them at the same time. We want our policies to have the same default policy and conflict resolution as theirs. Their policies are expressed in a high level syntax that needs to be mapped to policy languages such as SeRQL and SPARQL. If they store a graph for each set of policy rules, they would need large amounts of memory for answering queries, because of the endless quantity of contextual conditions that could exist. They avoided this problem by taking the query rewriting approach, but this comes at the expense of increasing the response time.

The access control policies defined using the *RAP framework*, developed by Reddivari et al. [RFJ05], let "agents" manage an RDF store through a set of "actions" that can be allowed or forbidden by an explicit collection of policy rules. With this framework they can control write operations, unlike us who manage read operations. If T_1 and T_2 are triples and T_c is a set of triples, the actions that an agent A can perform, directly or indirectly, on the store are: insert or remove a triple or set of triples from the graph (**insert**(A, T_1)**, remove**(A, T_1)**, insertSet**(A, T_c)**, removeSet**(A, T_c)), insert or remove inferred triples from a graph (**insertModel**(A, T_1)**, removeModel**(A, T_1)), update a triple (**update**(A, T_1, T_2)) and see or use a triple for query processing (**see**(A, T_1)**, use**(A, T_1)). Policy rules have the form $Modality(Action(A, T))$: $-Condition$, where $Modality$ is permit or prohibit, $Action$ is one of the actions already defined, A is an agent, T is a triple of the form (subject, predicate, object) where any of its parts can be a "?" to describe triple patterns, and $Condition$ is a boolean combination of triples for expressing constraints, which can use metadata about the triples maintained on the store. For example `permit(insert(A,(?,rdf:type, C))):-createNode(A,C)` allows agent A the possibility of inserting triples whose rdf:type is C only if he created the node C. Also, it allows the creation of custom predicates that are useful for building policies. An agent can only perform an action if he is allowed to do the action to the requested triple and he is permitted to do all its effects, for example inserting or deleting inferred triples. On their implementation the inferred triples are added to the store, it can only register one proof per inferred triple, even when more are possible, and it requires the temporal removal of all triples that cannot be seen by the agent making a query before query execution.

Jain and Farkas [JF06] developed an access control model for RDF and RDFS data that considers entailments. In this model, the security policy is defined by a set of pairs (pattern, security label), which are used to create a set of Security Objects with the triples obtained by pattern mapping. These security objects are pairs (triple, security label), making this a MAC policy. A Security Cover is the set of security objects that contains each triple in the graph only once, so in order to label the triples that can be mapped with more than one pattern, their security object is going to store the security label with the most restrictive classification, which has been obtained from the policy. From this, an Extended Security Cover is created to include triples obtained by inference of RDF and RDFS entailments. Warnings are generated when triples with a higher security levels can be inferred from triples with lower security levels. The difference between their model and our model is that, in their model, the access to the data is controlled at triples level and their policies can be classified as MAC, while our model provides a finer level of protection and is based on Roles.

Kim et al. [KJP08] proposed an access control model that focuses on read operations of RDF graphs described in XML, considering RDFS subclasses and subproperties and treating the problems of inheritance and inference. The access is managed through access authorizations, which are tuples of the form $< subj, obj, act, sign, type >$, where $subj$ is the user (subject) being granted the authorization, obj is a pattern that matches the triples to be protected, act is the operation (read in their work) to be controlled, $sign$ is a symbol (+ or -) that determines if the access to the triples is allowed or forbidden, and $type$ is R (Recursive) or L (Local) and says if the authorization is propagated to lower subclasses and subproperties recursively or not. The difference with our model is that the access to the data is protected at triple level; the patterns defined by them are too simple and, as a result of this, they do not allow restricting the access by means of SPARQL queries, e.g. the object in their patterns can only be a variable. The effects of recursive authorization can be obtained with our approach, but additional permissions need to be produced.

Flouris et al. [FFMA10] formalize an access control policy, and they present enforcement techniques. In their work, the access to specific triples is controlled through permissions of the form include/exclude (x, p, y) where $\mathcal{TP}, \mathcal{C}$. The triples in the scope of the triple pattern (x, p, y) and the constraints $\mathcal{TP}, \mathcal{C}$ will be accessible if there is an include permission associated to them or inaccessible otherwise. The conjunction of triple patterns $\mathcal{TP}$ and conjunction of constraints $\mathcal{C}$, let us express permissions in a way that is similar to SPARQL

queries. Thanks to this, one can think in controlling the access to triples by granting and denying the possibility to answer specific queries. This thesis is based on their work and extends it to develop a finer access control that is capable of managing restrictions at the level of subjects, predicates and objects.

Finin et al. [FJK+08] use the Web Ontology Language (OWL) to express authorization policies, focusing on RBAC. They define OWL ontologies that represent the RBAC model, taking two different approaches to represent roles, as OWL classes and subclasses, and as instances of the generic Role class. Under the former approach, roles are defined as subclasses of the Role class, an ActiveRole class is defined to manage actives roles and for each role a subclass activeRoleName is created. The use of the predicate **rdfs:subclassOf** allows them to handle role hierarchy, while OWL's property **disjointWith** handles static and dynamic separation of duties. The class Action and its subclasses PermittedAction and PhohibitedAction are used to associate permissions and prohibitions to the roles.

Under the latter approach, roles are instances of the generic Role class and the properties role and activeRole tie them to the subjects, role hierarchy is managed through additional rules and static and dynamic separation of duty are expressed with the properties *ssod* and *dsod*. The properties **permitted** and **prohibited** are used to associate permissions to the roles. The main disadvantage of this way of expressing policies is that we cannot control easily the access to groups of data, to deny/grant read access to triples, or part of them, we need to do it individually, instead of using a pattern followed by them.

Dietzold and Auer [DA06] describe an access control model based on RDF whose granularity is at triple level and provides access control to RDFS, OWL and other metamodels based on RDF. The operations managed are reading, adding and removing a set of triples from a store, through rules and query filters, both described in RDF with the vocabulary provided by the Lightweight Access Control Schema (LACS). The rules are represented in the Semantic Web Rule Language and the filters that can be triggered by them are defined in the RDF format. For example, if we want *admins* to be able to read all triples, the rule would be rdf:type(lacs:CurrentAction,lacs:Read) ∧ rdf:sameAs(lacs:CurrentAccount, ?a) ∧ foaf:member(:Admins,?a) → lacs:addAndStop(:currentAction,:AllFilter), and this rule reference the filter:

```
:AllFilter a lacs:Filder;
  rdfs:label "no restriction filter";
  lacs:sparql "CONSTRUCT{ ?s ?p ?o } WHERE { ?s ?p ?o }".
```

Their framework creates a virtual model from the real graph and these three models: a Session Model, which contains information pertaining to the active session, a User Model, which is the data that the user wants to get access, and a Maintenance Model, which consists of decision rules, filters and data pertaining to groups or accounts. To process an action, the access control processor changes the Session Model for the current session, a rule is evaluated by the rule processor to decide if query filters are fired or not and then the query engine applies the filters to the corresponding model. Rules with lower priority number (lacs:priority) are evaluated first, and after all the rules have been processed, the Virtual Model is presented.

Access control policies have been studied for data in the XML format. The authors in [BCF07] control write operations on XML documents that conform to a DTD. Their policies do not control precisely what data is being modified, instead they control which actions are performed based in the atomic updates, e.g., replacing a node in an XML tree with another. An update access type is a pair (node, operation(subnode)), that is used to specify the access permissions. The operations can be insert(subnode), replace(subnode_1, subnode_2), replace(string, string) or delete(subnode), where all the subnodes conform to the rules of the DTD of their parent node. A pair ($\mathcal{A}$, $\mathcal{F}$) constitutes a policy, where $\mathcal{A}$ is the set of allowed update access types and $\mathcal{F}$ is the set of forbidden update access types. When a subtree in an XML graph is replaced by another, the permissions over the root node of the subtree are checked and the action is performed if it is allowed. Now, for example, if the user is not allowed to deleted a node X, but he can delete or replace a parent node of X, he can bypass his prohibition by performing some of the operations on the parent node. They call this situation an inconsistency and they propose algorithms to repair the policies, which means transforming policies that are inconsistent to consistent. As in their policies, our policies are defined in function of the set of allowed and forbidden authorizations, but, unlike their work, our policies take care of controlling read access to the data. Checking if forbidden operations can be emulated or not by a series of allowed operations is an important part of their and our work.

Chapter 4

Hypothesis and Objectives

We propose to do the following work in this thesis:

1. Define the syntax and semantics of an access control policy for RDF that allows controlling the access to subjects, predicates and objects.
2. Propose, study and compare different enforcement mechanisms for the policy defined.

4.1 Hypothesis

Our main hypothesis is that it is possible to define access control policies for RDF data which:

1. Provides finer-grained control than the one defined in [FFMA10] where permissions are defined at the triple level.
2. Has, under a set of conditions over the policy, no sequence of allowed queries that allow a user to deduce forbidden data. Checking these conditions can be done in polynomial time over the size of the policy.
3. Has an enforcement algorithm that runs in polynomial time on the size of the RDF database.

4.2 Objectives

The objectives of the thesis are:

1. **Policy Definition:** define a suitable access control policy language for RDF data that provides control not only over triples but also over their subjects, predicates and

objects.

2. **Enforcement:** provide an efficient algorithm that enforces a policy and study its running time and space cost.
3. **Enforcement for Multiple Roles:** propose an algorithm that is able to handle multiple roles, and therefore multiple policies. This algorithm has to improve the naive version that would use separately the algorithm suggested in Objective 2. We will study its running time and space cost.

Chapter 5

Policy Definition

In [FFMA10], the smallest unit of information on which the security is managed is the RDF triple. The access to information is handled by a policy, which is a set of permissions that can grant or deny access to triples, according to the conditions stated by each permission. In order to obtain a finer-grained policy, we will add to these permissions a statement that describes exactly to what part of the data (triple) the access will be granted or denied.

5.1 Syntax of Fine-Grained Policies

As shown previously, controlling the access to RDF data at triples level is not enough to answer some queries that one could wish to perform on the graph. Imagine we want to collect data regarding the age of people in the system to get some statistics, but the person in charge of performing this task should not be able to relate the people to their respective age. If we block the access to the entire triple, and the triples are of the form (ID_1, age, *25*), we would not obtain any data; however, if we restrict the access to the part of the triple that contains the ID of the people, there would be no leaks of information, because the person performing the queries will not be able to link the people to their age.

The set Σ of *valid security patterns* is $\Sigma = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\},\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{s}\},\{\mathsf{o}\}\}$ and corresponds to all the possible ways in which we want to control a triple. For example, $\{\mathsf{p},\mathsf{o}\}$ corresponds to control over the predicate, object and their relationship. For a discussion of why $\{\mathsf{s},\mathsf{o}\}$ and $\{\mathsf{p}\}$ are not considered valid refer to the discussion in Section 5.3.

Definition 5.1 An access control permission (ACP) ρ is of the form

$$\rho = \texttt{APPLY}\ \mathcal{S}\ \texttt{SELECT}\ ?x, ?y, ?z\ \texttt{WHERE}\ \mathcal{TP}, \mathcal{C}$$

where $\mathcal{S}$ is called the security set with $\mathcal{S} \subseteq \Sigma$, $\mathcal{TP}$ is a conjunction of triple patterns that

contain at least variables $?x$, $?y$ and $?z$; and $\mathcal{C}$ is a conjunction of constraints of the form $?u$ op c, where $?u$ is a variable, $op \in \{<, >, \leq, \geq, =, \neq\}$ and c can be a variable, a URI or a literal. We refer to the elements in $\mathcal{S}$ by *security patterns*. □

Intuitively, an access control permission selects, with the query SELECT $?x$, $?y$, $?z$ WHERE $\mathcal{TP}, \mathcal{C}$, the set of triples to which the permission refers to, and then, with the expression APPLY $\mathcal{S}$, it states which data and relations within the triple the restriction should apply to. For example, if we want to apply the restriction only on the subject of the selected triples, then $\{\mathsf{s}\}$ would belong to $\mathcal{S}$. If $\mathcal{S} = \{\{\mathsf{s}, \mathsf{p}\}\}$ we want to put a restriction on the relationships between subjects and predicates but we are saying nothing about restricting access to the objects $\{\mathsf{o}\}$. Besides, the access to subject $\{\mathsf{s}\}$ will depend on the type of the restriction being imposed on $\{\mathsf{s}, \mathsf{p}\}$: the access to $\{\mathsf{s}\}$ will be granted if the access to $\{\mathsf{s}, \mathsf{p}\}$ is being granted, and denying the access to $\{\mathsf{s}, \mathsf{p}\}$ has no implications on the access restrictions of $\{\mathsf{s}\}$. This will be explained on the next section.

Example 5.1 Consider graph $\mathcal{G}_1$ in Figure 2.2 and the following set of ACP:

ρ_1 = APPLY $\mathcal{S}_1$ $\quad\quad$ $\mathcal{S}_1 = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$
SELECT $?x, ?w, ?y$
WHERE $(?x, \text{foaf:firstName}, ?y), (?x, \text{ex:completedProject}, ?z), ?z > 3, ?w = \text{foaf:firstName}$

ρ_2 = APPLY $\mathcal{S}_2$ $\quad\quad$ $\mathcal{S}_2 = \{\{\mathsf{p}, \mathsf{o}\}, \{\mathsf{s}, \mathsf{p}\}, \{\mathsf{s}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, \text{ex:area}, ?z), ?y = \text{ex:area}$

ρ_3 = APPLY $\mathcal{S}_3$ $\quad\quad$ $\mathcal{S}_3 = \{\{\mathsf{p}, \mathsf{o}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, \text{foaf:firstName}, ?z), (?x, \text{rdf:type}, \text{foaf:Student}), (?x, \text{ex:collaborateWith}, ?w),$
$?y = \text{foaf:firstName}$

ρ_4 = APPLY $\mathcal{S}_4$ $\quad\quad$ $\mathcal{S}_4 = \{\{\mathsf{o}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, \text{foaf:firstName}, ?z), ?y = \text{foaf:firstName}$

Permission ρ_1 selects triples (&c, foaf:firstName, *Allen*) and (&a, foaf:firstName, *William*) from Graph $\mathcal{G}_1$. The fact that $\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}$ belongs to $\mathcal{S}_1$ implies that we want to hide or allow access to the relationship between all the elements in these triples but not necessarily to

the individual elements in s, p and o. On the other hand, permission ρ_2 selects the triple (&c, ex:area, *Physics*). The security set $\mathcal{S}_2 = \{\{p, o\}, \{s, p\}, \{s\}\}$ says that we want to control both the data in s and the relationships between p and o, and s and p. In this case in particular, the permission would apply over (ex:area, *Physics*), (&c, ex:area) and (&c). Permission ρ_3 will select (&b, foaf:firstName, *Emma*), and the security set $\mathcal{S}_3$ will control the access to (foaf:firstName, *Emma*). Finally, permission ρ_4 selects the same triples as ρ_1 and ρ_3 together, that is to say (&c, foaf:firstName, *Allen*), (&a, foaf:firstName, *William*) and (&b, foaf:firstName, *Emma*), but will control the access to the object of the triples. □

ACPs allow us to represent the data or relationships in an RDF graph which we want to control. Now, we need to provide a syntax for the policies that will specify if those are going to be accessible or inaccessible.

Definition 5.2 An access control policy $\mathcal{P}$ is a tuple $(\mathcal{A}, \mathcal{F})$ where $\mathcal{A}$ and $\mathcal{F}$ are sets of ACPs and correspond to the allowed and forbidden permissions, respectively.

Example 5.2 (example 5.1 continued) Intuitively, a policy $\mathcal{P}_1 = (\mathcal{A}_1, \mathcal{F}_1)$ with $\mathcal{A}_1 = \{\rho_1, \rho_4\}$ and $\mathcal{F}_1 = \{\rho_2, \rho_3\}$ enforces that the tuples controlled by ρ_1 and ρ_4 can be accessed by the user, but the ones controlled by ρ_2 and ρ_3 cannot be seen by him. □

The issues of default permissions and conflict between them is going to be addressed in the next section where we formally define the semantics of our policy.

5.2 Semantics of Fine-Grained Policies

In this section we provide the semantics of the access control policies. To understand the intuition behind it, we will first explain the semantics through examples, and then we proceed to formalize it.

A policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$ controls the access to an entire RDF graph. The triples to which access is being allowed are determined from the set of permissions $\mathcal{A}$, and the triples to which access is being forbidden are determined from the set of permissions $\mathcal{F}$. The triples in the graph that are not affected by any permission from these sets are dealt with the

default permission, which states if the triples are going to be forbidden or allowed. There is a conflict if a triple is affected by permissions that belong to $\mathcal{A}$ and permissions that belong to $\mathcal{F}$ at the same time. Triples that are in conflict follow the rule for conflict resolution. In this work, the default permission forbids the access to the triples, in accordance with the principle of least privilege, and the rule for conflict resolution states that the access to the parts of a triple affected by both types of permissions will be forbidden. Note, however, that the default permission can be modified simply by adding a permission that gives access to the whole graph, APPLY $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ SELECT $?x,?y,?z$ WHERE $(?x,?y,?z)$.

5.2.1 By Example

A policy that affects a graph will control the access to its triples through permissions. Each Triple can be affected by none, one or multiple permissions. There are four possible scenarios for a triple.

1. It is affected only by permissions that allow access to it.
2. It has no permissions affecting it.
3. It is affected only by permissions forbidding access to it.
4. It is affected by both types of permissions.

To explain each case we will use RDF graph $\mathcal{G}_{ex}$ in Figure 5.1 and the following permissions:

	s	p	o		Allowed ACP	Forbidden ACP
t_{ex1}	a	b	c	⇒	ρ_{A1}, ρ_{A2}, ρ_{A3}, ρ_{A4}	ρ_{F2}, ρ_{F3}
t_{ex2}	d	c	e	⇒	ρ_{A1}, ρ_{A3}, ρ_{A4}	ρ_{F1}, ρ_{F2}, ρ_{F3}
t_{ex3}	e	f	b	⇒	ρ_{A1}, ρ_{A3}, ρ_{A4}	ρ_{F3}

Figure 5.1: Graph $\mathcal{G}_{ex}$ and the permissions affecting its triples.

ρ_{A1} = APPLY $\mathcal{S}_{A1}$ SELECT $?x,?y,?z$ WHERE $(?x,?y,?z)$ $\quad\mathcal{S}_{A1} = \{\{\mathsf{s}\}\}$

ρ_{A2} = APPLY $\mathcal{S}_{A2}$ SELECT $?x,?y,?z$ WHERE $(?x,\mathsf{b},?z), ?y = \mathsf{b}$ $\quad\mathcal{S}_{A2} = \{\{\mathsf{p},\mathsf{o}\}\}$

ρ_{A3} = APPLY $\mathcal{S}_{A3}$ SELECT $?x,?y,?z$ WHERE $(?x,?y,?z)$ $\quad\mathcal{S}_{A3} = \{\{\mathsf{o}\}\}$

ρ_{A4} = APPLY $\mathcal{S}_{A4}$ SELECT $?x,?y,?z$ WHERE $(?x,?y,?z)$ $\quad\mathcal{S}_{A4} = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$

ρ_{F1} = APPLY $\mathcal{S}_{F1}$ SELECT $?x,?y,?z$ WHERE $(?x,?y,?z),(?z,?v,?w)$ $\quad\mathcal{S}_{F1} = \{\{\mathsf{s}\}\}$

ρ_{F2} = APPLY $\mathcal{S}_{F2}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z), (?m, ?n, ?y)$ $\quad \mathcal{S}_{F2} = \{\{\mathsf{o}\}\}$

ρ_{F3} = APPLY $\mathcal{S}_{F3}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z)$ $\quad \mathcal{S}_{F3} = \{\{\mathsf{p}, \mathsf{o}\}\}$

To understand what means that a triple is affected by a permission, we need to know the concept of matching permissions. The matching permissions of a triple, given a set of permissions Π, are the union of the security patterns of the permission in Π that affect the triple.

Now that we know when a triple is being affected by a permission, we can explain in detail the cases previously stated.

Case 1: The triple is affected only by allowed permissions. In this case, the data from the triple will be available according to the security sets of the permissions involved. Let us consider policy $\mathcal{P}_{ex1} = (\mathcal{A}_{ex1}, \mathcal{F}_{ex1})$ with $\mathcal{A}_{ex1} = \{\rho_{A1}, \rho_{A2}\}$ and $\mathcal{F}_{ex1} = \varnothing$.

The triple (a, b, c) is affected by both permissions in $\mathcal{A}_{ex1}$. This means that we want to provide access only to the subject, because the permission ρ_{A1} provides the security set $\mathcal{S}_{A1} = \{\{\mathsf{s}\}\}$; and we want to permit the access to the pair predicate-object, because the permission ρ_{A2} affects the parts of the triple indicated by the set $\mathcal{S}_{A2} = \{\{\mathsf{p}, \mathsf{o}\}\}$. This is, the matching permissions of (a, b, c), according to $\mathcal{A}_{ex1}$, is the set $\mathcal{M} = \{\{\mathsf{s}\}, \{\mathsf{p}, \mathsf{o}\}\}$. From this set we get that (a, _ , _) is visible thanks to $\{\mathsf{s}\}$ and (_ , b, c) thanks to $\{\mathsf{p}, \mathsf{o}\}$.

For the other two triples only permission ρ_{A1} applies and, therefore, only their subject should be visible. The user is expected to have access to the data as shown in graph $\mathcal{G}'_{ex}$, where some of the values are missing.

Graph $\mathcal{G}'_{ex}$:

	s	p	o
t'_{ex1}	a		
t''_{ex1}		b	c
t'_{ex2}	d		
t'_{ex3}	e		

If we add permission ρ_{A3} to $\mathcal{A}_{ex1}$ we would be providing access to the objects of the triples, thanks to its set $\mathcal{S}_{A3} = \{\{\mathsf{o}\}\}$. However, in the case of (a, b, c), this new permission is subsumed by ρ_{A2} that already gives access to the object of the triple. Thus, the user has access to the data as shown below, on graph $\mathcal{G}''_{ex}$.

Graph $\mathcal{G}''_{ex}$:

	s	p	o
t'_{ex1}	a		
t''_{ex1}		b	c
t'_{ex2}	d		
t''_{ex2}			e
t'_{ex3}	e		
t''_{ex3}			b

Case 2: The triple is not affected by any permission. If the triple has no permissions associated with it, the rules for the default case will be enforced. Which means that the triple would be entirely hidden. This is aligned with the principle of least privilege. For example, for policy $\mathcal{P}_{ex2} = (\mathcal{A}_{ex2}, \mathcal{F}_{ex2})$ and $\mathcal{A}_{ex2} = \{\rho_{A2}\}$ and $\mathcal{F}_{ex2} = \varnothing$, we have that no permission applies to triples (d, c, e) and (e, f, b) in $\mathcal{G}_{ex}$. Therefore, these triples will not be visible to the user, who will be able to access data as shown in graph $\mathcal{G}'''_{ex}$.

Graph $\mathcal{G}'''_{ex}$:

	s	p	o
t'_{ex1}		b	c

Case 3: The triple is affected only by forbidden permissions. Consider the policy $\mathcal{P}_{ex3} = (\mathcal{A}_{ex3}, \mathcal{F}_{ex3})$ with $\mathcal{A}_{ex3} = \varnothing$ and $\mathcal{F}_{ex3} = \{\rho_{F1}, \rho_{F2}\}$. Permission ρ_{F1} will affect the triple (d, c, e) of graph $\mathcal{G}_{ex}$, whose object is also the subject of (e, f, b) in the graph. Permission ρ_{F2} will affect triples (d, c, e) and (a, b, c) of graph $\mathcal{G}_{ex}$ because the predicate of these triples are the object of some triple in the graph. The triple (e, f, b) will not be affected by forbidden permissions. The next graph shows the permissions affecting each triple in graph $\mathcal{G}_{ex}$.

s	p	o		ACP	matching permissions
a	b	c	⇒	ρ_{F2}	$\mathcal{S}_{F2} = \{\{\mathsf{o}\}\}$
d	c	e	⇒	ρ_{F1}, ρ_{F2}	$\mathcal{S}_{F1} \cup \mathcal{S}_{F2} = \{\{\mathsf{s}\}, \{\mathsf{o}\}\}$
e	f	b	⇒	–	–

Since there are no permissions that give access to any of the triples, all of them will be hidden.

Case 4: The triple is affected by both types of permissions. In order to provide semantics in the case of conflict we use the *security set tree* shown in Figure 5.2, which is based on the relationships between the security sets.

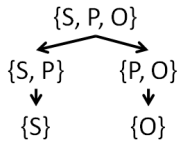

Figure 5.2: Security set tree.

When we are giving access to parts of a triple, we are implicitly allowing access to all the subparts of it. For example, if $\{\mathsf{p},\mathsf{o}\}$ is allowed, we are implicitly also giving access to $\{\mathsf{o}\}$. If we want to allow the access to the entire triple the security set $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ is allowed and the parts $\{\mathsf{s},\mathsf{p}\}$, $\{\mathsf{p},\mathsf{o}\}$, $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$ have to be allowed too. This is because if we know a triple we can infer all its parts. Thus, when access is given to a security set, all the nodes below it in the security set tree are also implicitly allowed.

To deny access to a security set of the triple, we apply the permission over the security set and implicitly over all the security sets that are its ancestors on the tree. This is, if we restrict access to $\{\mathsf{s}\}$, the nodes $\{\mathsf{s},\mathsf{p}\}$ and $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ have to be hidden because they are showing the existence of $\{\mathsf{s}\}$ from that specific triple in the graph.

The interaction/conflict between ACPs could be used, for example, to hide $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ and show the parts $\{\mathsf{s},\mathsf{p}\}$ and $\{\mathsf{p},\mathsf{o}\}$ of triples. However, if we try to give access to $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ and forbid $\{\mathsf{s},\mathsf{p}\}$ the conflict would result in that only $\{\mathsf{p},\mathsf{o}\}$ can be accessed.

Under policy $\mathcal{P}_{ex4} = (\mathcal{A}_{ex4}, \mathcal{F}_{ex4})$ with $\mathcal{A}_{ex4} = \{\rho_{A4}\}$ and $\mathcal{F}_{ex4} = \{\rho_{F1}\}$, the only triple with a conflict is (d, c, e) which is affected by both ρ_{A4} and ρ_{F1}. The ACP ρ_{F1} forbids the access to the subject of the triple so $\{\mathsf{s}\}$, $\{\mathsf{s},\mathsf{p}\}$ and $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ are forbidden, but ρ_{A3} allows access to $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$ and consequently to $\{\mathsf{s}\}$, $\{\mathsf{o}\}$, $\{\mathsf{s},\mathsf{p}\}$ and $\{\mathsf{p},\mathsf{o}\}$. There are conflicts between the ACPs and we will only want to provide access when it is not forbidden (explicitly or implicitly). Therefore, even though there are allow ACPs for $\{\mathsf{s}\}$, $\{\mathsf{s},\mathsf{p}\}$ and $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$, none of these parts are going to be available for the users associated with this policy, since they are also forbidden by ρ_{F1}. As a consequence the only security sets that are allowed and not involved in conflicts are $\{\mathsf{o}\}$ and $\{\mathsf{p},\mathsf{o}\}$. As it was shown in Case 1, $\{\mathsf{p},\mathsf{o}\}$ subsumes $\{\mathsf{o}\}$. Graph $\mathcal{G}''''_{ex}$ shows the visible data for policy $\mathcal{P}_{ex4}$.

Graph $\mathcal{G}''''_{ex}$:

	s	p	o
t'_{ex1}	a	b	c
t'_{ex2}		c	e
t'_{ex3}	e	f	b

Now, let us add ρ_{F3} to the set of forbidden permissions $\mathcal{F}_{ex4}$. In this case all triples in the graph will have conflicts. Using the same method we will obtain the graph $\mathcal{G}''''_{ex}$. From the triple (d, c, e) only e will be shown because $\{\mathsf{s}\}$ and $\{\mathsf{p}, \mathsf{o}\}$ are forbidden.

Graph $\mathcal{G}''''_{ex}$:

	s	p	o
t'_{ex1}			c
t''_{ex1}	a	b	
t'_{ex2}			e
t'_{ex3}			b
t''_{ex3}	e	f	

5.2.2 Formalization

Now that we have shown the intuition of the enforcement of the policies through examples, we will proceed to formalize the semantics. We will first associate with each triple the permissions that apply over it, compute the allowed permissions that are a consequence of them (the *conflict-clear permissions*) and then we compute an *anonymized graph* that shows only the data that should be visible under the policy.

Definition 5.3 Given a graph $\mathcal{G}$ and a set of permissions Π, the *matching permissions* of triple $t \in \mathcal{G}$, denoted $\mathcal{M}(\mathcal{G}, \Pi, t)$ are:

$$\mathcal{M}(\mathcal{G}, \Pi, t) = \{b \mid b \in \mathcal{S}, (\texttt{APPLY}\ \mathcal{S}\ \texttt{SELECT}\ ?x, ?y, ?z\ \texttt{WHERE}\ \mathcal{TP}, \mathcal{C}) \in \Pi, \\ \mu \in \langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}}, \mu(?x) = t[\mathsf{s}], \mu(?y) = t[\mathsf{p}], \mu(?z) = t[\mathsf{o}]\}$$

□

Example 5.3 (example 5.1 continued) For graph $\mathcal{G}_1$ and a set $\Pi_1 = \{\rho_1, \rho_2, \rho_3, \rho_4\}$ we have, for example:

$$\mathcal{M}(\mathcal{G}_1, \Pi_1, t_{13}) = \{b \mid b \in \mathcal{S}_2, \rho \in \Pi_1, \mu \in \langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}1}, \mu(?x) = \&\mathsf{a}, \mu(?y) = \mathsf{foaf{:}firstName}, \\ \mu(?z) = \textit{William}\} = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}, \{\mathsf{o}\}\}$$
$$\mathcal{M}(\mathcal{G}_1, \Pi_1, t_{22}) = \{b \mid b \in \mathcal{S}_2, \rho \in \Pi_1, \mu \in \langle\!\langle \mathcal{TP}, \mathcal{C} \rangle\!\rangle_{\mathcal{G}1}, \mu(?x) = \&\mathsf{c}, \mu(?y) = \mathsf{ex{:}area}, \mu(?z) = \\ \textit{Physics}\} = \{\{\mathsf{p}, \mathsf{o}\}, \{\mathsf{s}, \mathsf{p}\}, \{\mathsf{s}\}\}$$

□

Example 5.4 (example 5.1 continued) Consider the graph $\mathcal{G}_1$ and a set $\Pi_2 = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7\}$, with ρ_5, ρ_6 and ρ_7 as defined below. Figure 5.3 shows the triples in $\mathcal{G}_1$, the permissions affecting them, and the matching permissions generated by the set Π_2.

ρ_5 = APPLY $\mathcal{S}_5$ $\quad\quad\quad\quad\quad\quad\quad\quad$ $\mathcal{S}_5 = \{\{\mathsf{o}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, \mathsf{foaf{:}firstName}, ?z), (?x, \mathsf{rdf{:}type}, \mathsf{ex{:}Teacher}), ?y = \mathsf{foaf{:}firstName}$

ρ_6 = APPLY $\mathcal{S}_6$ $\quad\quad\quad\quad\quad\quad\quad\quad$ $\mathcal{S}_6 = \{\{\mathsf{s}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, ?y, ?z)$

ρ_7 = APPLY $\mathcal{S}_7$ $\quad\quad\quad\quad\quad\quad\quad\quad$ $\mathcal{S}_7 = \{\{\mathsf{s}, \mathsf{p}\}\}$
SELECT $?x, ?y, ?z$
WHERE $(?x, ?y, \mathsf{ex{:}Student}), ?z = \mathsf{ex{:}Student}$

□

	s	p	o	ACP	$\mathcal{M}(\mathcal{G}_1, \Pi_2, t)$
t_1	foaf:Person	rdfs:subClassOf	foaf:Agent	ρ_6	$\{\{\mathsf{s}\}\}$
t_2	ex:Student	rdfs:subClassOf	foaf:Person	ρ_6	$\{\{\mathsf{s}\}\}$
t_3	ex:Teacher	rdfs:subClassOf	foaf:Person	ρ_6	$\{\{\mathsf{s}\}\}$
t_4	ex:area	rdfs:domain	ex:Teacher	ρ_6	$\{\{\mathsf{s}\}\}$
t_5	ex:area	rdfs:range	rdfs:Literal	ρ_6	$\{\{\mathsf{s}\}\}$
t_6	ex:collaborateWith	rdfs:domain	ex:Student	ρ_6, ρ_7	$\{\{\mathsf{s}\}, \{\mathsf{s}, \mathsf{p}\}\}$
t_7	ex:collaborateWith	rdfs:range	ex:Student	ρ_6, ρ_7	$\{\{\mathsf{s}\}, \{\mathsf{s}, \mathsf{p}\}\}$
t_8	ex:completedProject	rdfs:domain	ex:Student	ρ_6, ρ_7	$\{\{\mathsf{s}\}, \{\mathsf{s}, \mathsf{p}\}\}$
t_9	ex:completedProject	rdfs:range	rdfs:Literal	ρ_6	$\{\{\mathsf{s}\}\}$
t_{10}	foaf:firstName	rdfs:domain	foaf:Person	ρ_6	$\{\{\mathsf{s}\}\}$
t_{11}	foaf:firstName	rdfs:range	rdfs:Literal	ρ_6	$\{\{\mathsf{s}\}\}$
t_{12}	&a	rdf:type	ex:Student	ρ_6, ρ_7	$\{\{\mathsf{s}\}, \{\mathsf{s}, \mathsf{p}\}\}$
t_{13}	&a	foaf:firstName	William	ρ_1, ρ_4, ρ_6	$\{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}, \{\mathsf{o}\}, \{\mathsf{s}\}\}$
t_{14}	&a	ex:completedProject	5	ρ_6	$\{\{\mathsf{s}\}\}$
t_{15}	&b	rdf:type	ex:Student	ρ_6, ρ_7	$\{\{\mathsf{s}\}, \{\mathsf{s}, \mathsf{p}\}\}$
t_{16}	&b	foaf:firstName	Emma	ρ_3, ρ_4, ρ_6	$\{\{\mathsf{p}, \mathsf{o}\}, \{\mathsf{o}\}, \{\mathsf{s}\}\}$
t_{17}	&b	ex:collaborateWith	&a	ρ_6	$\{\{\mathsf{s}\}\}$
t_{18}	&b	ex:completedProject	1	ρ_6	$\{\{\mathsf{s}\}\}$
t_{19}	&c	rdf:type	ex:Teacher	ρ_6	$\{\{\mathsf{s}\}\}$
t_{20}	&c	foaf:firstName	Allen	ρ_1, ρ_4, ρ_5, ρ_6	$\{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}, \{\mathsf{o}\}, \{\mathsf{s}\}\}$
t_{21}	&c	ex:completedProject	20	ρ_6	$\{\{\mathsf{s}\}\}$
t_{22}	&c	ex:area	Physics	ρ_2, ρ_6	$\{\{\mathsf{p}, \mathsf{o}\}, \{\mathsf{s}, \mathsf{p}\}, \{\mathsf{s}\}\}$

Figure 5.3: ACPs and matching permissions of graph $\mathcal{G}_1$.

Of particular interest are the sets associated with allowed and forbidden permissions in a policy $\mathcal{P}$ because they will be used to determine the restrictions imposed on the triple.

Example 5.5 (example 5.1 continued) If we have $\mathcal{A}_1 = \{\rho_1, \rho_4\}$ and $\mathcal{F}_1 = \{\rho_2, \rho_3\}$, the triples that have a non-empty set of matching permissions are:

	s	p	o	$\mathcal{M}(\mathcal{G}_1, \mathcal{A}_1, t)$	$\mathcal{M}(\mathcal{G}_1, \mathcal{F}_1, t)$
t_{13}	&a	foaf:firstName	*William*	$\mathcal{S}_1 \cup \mathcal{S}_4 = \{\{s, p, o\}, \{o\}\}$	–
t_{16}	&b	foaf:firstName	*Emma*	$\mathcal{S}_4 = \{\{o\}\}$	$\mathcal{S}_3 = \{\{p, o\}\}$
t_{20}	&c	foaf:firstName	Allen	$\mathcal{S}_1 \cup \mathcal{S}_4 = \{\{s, p, o\}, \{o\}\}$	–
t_{22}	&c	ex:area	Physics	–	$\mathcal{S}_2 = \{\{p, o\}, \{s, p\}, \{s\}\}$

□

Example 5.6 (example 5.1 continued) Intuitively, a policy $\mathcal{P}_2 = (\mathcal{A}_2, \mathcal{F}_2)$ with $\mathcal{A}_2 = \{\rho_1\}$ and $\mathcal{F}_2 = \{\rho_2, \rho_3\}$ enforces that the tuples controlled by ρ_1 can be accessed by the user, but the ones controlled by ρ_2 and ρ_3 cannot be seen by him. The triples with non-empty matching permissions are:

Triple	ACP	$\mathcal{M}(\mathcal{G}_1, \mathcal{A}_2, t)$	$\mathcal{M}(\mathcal{G}_1, \mathcal{F}_2, t)$	Availability
t_{13}	ρ_1	$\mathcal{S}_1 = \{\{s, p, o\}\}$	–	Accessible
t_{16}	ρ_3	–	$\mathcal{S}_3 = \{\{p, o\}\}$	Inaccessible
t_{20}	ρ_1	$\mathcal{S}_1 = \{\{s, p, o\}\}$	–	Accessible
t_{22}	ρ_2	–	$\mathcal{S}_2 = \{\{p, o\}, \{s, p\}, \{s\}\}$	Inaccessible

We should be able to access the triples t_{13} = (&a, foaf:firstName, *William*) and t_{20} = (&c, foaf:firstName, *Allen*) thanks to the security set $\mathcal{S}_1$. The rest of the tuples in Graph $\mathcal{G}_1$ should be hidden. Note that in this case there is no conflict between the allowed and forbidden matching permissions. □

Given a set of security patterns $\mathcal{S}$, we denote by $\mathcal{S}^{\subseteq}$ (and $\mathcal{S}^{\supseteq}$ respectively) the set of security patterns that are subsets (superset), of the elements in $\mathcal{S}$. More formally, $\mathcal{S}^{\subseteq} = \{b' | b'$ is a security pattern, exists $b \in \mathcal{S}$ and $b' \subseteq b\}$, and $\mathcal{S}^{\supseteq} = \{b' | b'$ is a security pattern, exists $b \in \mathcal{S}$ and $b' \supseteq b\}$. In other words $\mathcal{S}^{\subseteq}$ (and $\mathcal{S}^{\supseteq}$) contain the descendants (ancestor) of the elements in $\mathcal{S}$ in the security set tree. Also, let $\mathcal{S}^{\downarrow}$ be a set of permissions such that

i. $S^{\downarrow} \subseteq S$;
ii. for all $b \in S$ there exists $b' \in S^{\downarrow}$ such that $b' \supseteq b$; and
iii. there are no $b, b' \in S^{\downarrow}$ such that $b \subseteq b'$.

Definition 5.4 Given a graph $\mathcal{G}$, the conflict-clear permissions for a triple t over a policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$ is the set of permissions $\mathsf{CCP}(\mathcal{G}, \mathcal{P}, t) = (\mathcal{M}(\mathcal{G}, \mathcal{A}, t)^{\subseteq} \setminus \mathcal{M}(\mathcal{G}, \mathcal{F}, t)^{\supseteq})^{\downarrow}$. □

Intuitively, the conflict-clear permissions corresponds to the parts of the triple for which the policy allows access after removing all conflicts. Note that the set of conflict-clear permissions will always be one of the following: $\{\{\mathsf{s}\}\}$, $\{\{\mathsf{o}\}\}$, $\{\{\mathsf{s},\mathsf{p}\}\}$, $\{\{\mathsf{p},\mathsf{o}\}\}$, $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$, $\{\{\mathsf{s}\},\{\mathsf{o}\}\}$, $\{\{\mathsf{s}\},\{\mathsf{p},\mathsf{o}\}\}$, $\{\{\mathsf{s},\mathsf{p}\},\{\mathsf{o}\}\}$ or $\{\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\}\}$. With the CCP of a triple we know exactly what we can show.

Example 5.7 (example 5.4 continued) Consider the graph $\mathcal{G}_1$ and an access control policy $\mathcal{P}_3 = (\mathcal{A}_3, \mathcal{F}_3)$, $\mathcal{A}_3 = \{\rho_1, \rho_2, \rho_3\}$ and $\mathcal{F}_3 = \{\rho_5\}$.

From the access control policy $\mathcal{P}_3$ we get the following matching permissions:

	s	p	o	$\mathcal{M}(\mathcal{G}_1, \mathcal{A}_3, t)$	$\mathcal{M}(\mathcal{G}_1, \mathcal{F}_3, t)$
t_{13}	&a	foaf:firstName	*William*	$\mathcal{S}_1 = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$	–
t_{16}	&b	foaf:firstName	*Emma*	$\mathcal{S}_3 = \{\{\mathsf{p},\mathsf{o}\}\}$	–
t_{20}	&c	foaf:firstName	*Allen*	$\mathcal{S}_1 = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$	$\mathcal{S}_5 = \{\{\mathsf{o}\}\}$
t_{22}	&c	ex:area	*Physics*	$\mathcal{S}_2 = \{\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{s}\}\}$	–

Now we need to determine the CCP of each triple using the definition $\mathsf{CCP}(\mathcal{G}, \mathcal{P}, t) = (\mathcal{M}(\mathcal{G}, \mathcal{A}, t)^{\subseteq} \setminus \mathcal{M}(\mathcal{G}, \mathcal{F}, t)^{\supseteq})^{\downarrow}$.

The CCP of triples t_{13} and t_{16} will be the same as their security sets, $\mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, t_{13}) = \mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, (\mathsf{\&a}, \mathsf{foaf{:}firstName}, \mathsf{William})) = (\{\mathsf{s},\mathsf{p},\mathsf{o}\}^{\subseteq} \setminus \varnothing^{\supseteq})^{\downarrow} = (\{\{\mathsf{s},\mathsf{p},\mathsf{o}\},\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{s}\},\{\mathsf{o}\}\})^{\downarrow} = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ and $\mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, t_{16}) = \{\{\mathsf{p},\mathsf{o}\}\}$.

The only triple with conflicting permissions is t_{20} = (&c, foaf:firstName, *Allen*), and its conflict-clear permission is $\mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, t_{20}) = \mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, (\mathsf{\&c}, \mathsf{foaf{:}firstName}, \mathsf{Allen})) = (\{\mathsf{s},\mathsf{p},\mathsf{o}\}^{\subseteq} \setminus \{\mathsf{o}\}^{\supseteq})^{\downarrow} = (\{\{\mathsf{s},\mathsf{p},\mathsf{o}\},\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{s}\},\{\mathsf{o}\}\} \setminus \{\{\mathsf{s},\mathsf{p},\mathsf{o}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{o}\}\})^{\downarrow} = (\{\{\mathsf{s},\mathsf{p}\},\{\mathsf{s}\}\})^{\downarrow} = \{\{\mathsf{s},\mathsf{p}\}\}$.

To obtain the CCP of triple t_{22} we need to determine the security sets that subsume the rest, $\mathsf{CCP}(\mathcal{G}_1, \mathcal{P}_3, t_2) = (\{\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\},\{\mathsf{s}\}\})^{\downarrow} = \{\{\mathsf{s},\mathsf{p}\},\{\mathsf{p},\mathsf{o}\}\}$. The following table resumes the results.

□

	$\mathcal{M}_1 = \mathcal{M}(\mathcal{G}_1, \mathcal{A}_3, t)^{\subseteq}$	$\mathcal{M}_2 = \mathcal{M}(\mathcal{G}_1, \mathcal{F}_3, t)^{\supseteq}$	$\mathcal{M}_1 \setminus \mathcal{M}_2$	CCP
t_{13}	$\{\{s,p,o\},\{s,p\},\{p,o\},\{s\},\{o\}\}$			$\{\{s,p,o\}\}$
t_{16}	$\{\{p,o\},\{o\}\}$			$\{\{p,o\}\}$
t_{20}	$\{\{s,p,o\},\{s,p\},\{p,o\},\{s\},\{o\}\}$	$\{\{o\},\{p,o\},\{s,p,o\}\}$	$\{\{s,p\},\{s\}\}$	$\{\{s,p\}\}$
t_{22}	$\{\{s,p\},\{p,o\},\{s\},\{o\}\}$			$\{\{s,p\},\{p,o\}\}$

Definition 5.5 Given a triple t and a non-empty security pattern $b \subseteq \{s,p,o\}$, the anonymized triple $\mathsf{Anon}(t,b) = t'$ with $t'[a] = t[a]$ for $a \in b$ and $t'[a] = \_:\lambda$ otherwise, in which $\_:\lambda$ denotes a fresh blank node. □

Intuitively, the fresh blank node $\_:\lambda$ hides a value that the user is not permitted to see. Considering that the predicate of an anonymized triple can be a blank node, we are now working with generalized RDF triples. A *generalized RDF triple* is a triple in which its subject, predicate and object can be an IRI, a blank node or a literal [Gro14a].

Considering also that the only difference between the triples we produce to hide information with the standard triple is that the predicate can be a blank node, our anonymized triples could be transformed into standard triples by creating a URI to replace the predicates hiding information through blank nodes.

Definition 5.6 Given a graph $\mathcal{G}$ and a policy $\mathcal{P}$, the anonymized graph $\mathsf{Anon}(\mathcal{G},\mathcal{P}) = \{t' |$ $t \in \mathcal{G}$, $b \in \mathsf{CCP}(\mathcal{G},\mathcal{P},t)$ and $t' = \mathsf{Anon}(t,b)\}$. □

Example 5.8 (example 5.7 continued) Given that we already have the CCP of triples t_{13}, t_{16}, t_{20} and t_{22} we can generate $\mathsf{Anon}(\mathcal{G}_1,\mathcal{P}_3)$, graph composed by anonymized triples according to the rules of the policy $\mathcal{P}_3$. The graph $\mathsf{Anon}(\mathcal{G}_1,\mathcal{P}_3)$ can be seen in Figure 5.4. □

		s	p	o
$\mathsf{Anon}(t_{13},\{s,p,o\})$	⇒	&a	foaf:firstName	*William*
$\mathsf{Anon}(t_{16},\{p,o\})$	⇒	$\_:\lambda 1$	foaf:firstName	*Emma*
$\mathsf{Anon}(t_{20},\{s,p\})$	⇒	&c	foaf:firstName	$\_:\lambda 2$
$\mathsf{Anon}(t_{22},\{s,p\})$	⇒	&c	ex:area	$\_:\lambda 3$
$\mathsf{Anon}(t_{22},\{p,o\})$	⇒	$\_:\lambda 4$	ex:area	*Physics*

Figure 5.4: Graph $\mathsf{Anon}(\mathcal{G}_1,\mathcal{P}_3)$.

Now we need to define the answers to a query in the presence of an access control policy as the traditional answers but over the anonymized graph.

Definition 5.7 Given a graph $\mathcal{G}$, an access control policy $\mathcal{P}$ and a conjunctive query CQ, the answers to query CQ from $\mathcal{G}$ under $\mathcal{P}$ is the set:

$$\mathbf{Ans}^{\mathcal{P}}(\mathrm{CQ}, \mathcal{G}) = \mathbf{Ans}(\mathrm{CQ}, \mathsf{Anon}(\mathcal{G}, \mathcal{P}))$$ □

Example 5.9 If we execute on graph $\mathsf{Anon}(\mathcal{G}_1, \mathcal{P}_3)$ the query CQ_4 = `SELECT` $?x, ?z$ `WHERE` ($?x$, foaf:firstName, $?z$), we obtain $\mathbf{Ans}^{\mathcal{P}_3}(\mathrm{CQ}_4, \mathcal{G}_1) = \mathbf{Ans}(\mathrm{CQ}_4, \mathsf{Anon}(\mathcal{G}_1, \mathcal{P}_3)) = \{\{(?x,$ &a$), (?z, \textit{William})\}, \{(?x, _{:}\lambda 1), (?z, \textit{Emma})\}, \{(?x,$ &c$), (?z, _{:}\lambda 2)\}\}$, while for CQ_5 = `SELECT` $?z$ `WHERE` ($?x$, foaf:firstName, $?z$) we get $\mathbf{Ans}^{\mathcal{P}_3}(\mathrm{CQ}_5, \mathcal{G}_1) = \mathbf{Ans}(\mathrm{CQ}_5, \mathsf{Anon}(\mathcal{G}_1, \mathcal{P}_3)) = \{\{(?z, \textit{William})\}, \{(?z, \textit{Emma})\}, \{(?z, \_{:}\lambda 2)\}\}$. □

5.3 Justifications of Design Decisions

The choice of syntax and semantics for the access control policies was not trivial since it involved a lot of analysis to ensure that they were intuitive, easy to use and expressive while ensuring well behaved properties such as avoiding leak of information. In what follows we will discuss some of the issues that were considered at design time.

5.3.1 The Security Set Tree.

In order to provide a fine-grained access control, we opted for managing the access to parts of a triple. Initially one could think that we are talking about imposing restrictions directly over the subjects $\{\mathsf{s}\}$, predicates $\{\mathsf{p}\}$ and objects $\{\mathsf{o}\}$. However, that would be like controlling access to the URIs in the RDF graph based on their presence as subjects, predicates or objects. We would be neglecting the fact that triples are the smallest data unit of RDF, so it does not make sense to leave them out of the definition of the permissions.

To preserve the sense given to the URIs by a triple, we need to control the access to its subject, predicate, object and the relationships between them. By relationships we are referring to the facts that can be extracted from an entire triple, as well as a triple that is missing one of its parts. This is it, the subject is related to a predicate $\{\mathsf{s}, \mathsf{p}\}$, the predicate is related to an object $\{\mathsf{p}, \mathsf{o}\}$, the subject is related to an object $\{\mathsf{s}, \mathsf{o}\}$. Therefore, we need control over the security patterns $\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}$, $\{\mathsf{s}, \mathsf{p}\}$, $\{\mathsf{p}, \mathsf{o}\}$, $\{\mathsf{s}, \mathsf{o}\}$, $\{\mathsf{s}\}$, $\{\mathsf{p}\}$ and $\{\mathsf{o}\}$

Now if we compare this set with the security set shown in Definition 5.1 we can notice

that the patterns $\{\mathsf{s},\mathsf{o}\}$ and $\{\mathsf{p}\}$ are not present in it. We decided against considering them because no interesting ACPs could be generated with them. The set $\{\mathsf{s},\mathsf{o}\}$ was removed from the possible security sets because when a triple t is being granted access to $\{\{\mathsf{s},\mathsf{o}\}\}$ the value of p can be generally inferred. Allowing access only to the predicate of a triple would only let us ask if there exists a subject that has an object related to it by that predicate in the graph, query that does not really tell us much if it is not accompanied by an specific subject or object. Denying access to the predicate would be the same as forbidding access to $\{\mathsf{s},\mathsf{p}\}$, $\{\mathsf{p},\mathsf{o}\}$ and by extension to $\{\mathsf{s},\mathsf{p},\mathsf{o}\}$, so it can be subsumed by these patterns.

We also studied whether removing security sets $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$, from the security tree, would produce policies that are as expressive as the ones defined including them or not. For this to be possible, each policy defined with security sets from our current security tree must be able to be rewritten using ACPs without $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$. Considering the same semantics as before, the sets of possible CCP for a triple obtained from policies defined with and without $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$ are equal. Sets $\{\{\mathsf{s},\mathsf{p}\}\}$, $\{\{\mathsf{p},\mathsf{o}\}\}$, $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ are easily obtained by adding an ACP that allow access to them and $\{\{\mathsf{s},\mathsf{p}\}, \{\mathsf{p},\mathsf{o}\}\}$ can be obtained by having a triple affected by two ACPs, each one of them allowing access with one of the first two security patterns. Sets $\{\{\mathsf{s}\}\}$, $\{\{\mathsf{o}\}\}$, $\{\{\mathsf{s}\}, \{\mathsf{o}\}\}$ can be obtained if two ACPs grant and deny access to a triple and both have the same security set. Finally, sets $\{\{\mathsf{s}\}, \{\mathsf{p},\mathsf{o}\}\}$, $\{\{\mathsf{s},\mathsf{p}\}, \{\mathsf{o}\}\}$ can be obtained if a triple is affected by an ACP granting access to the entire triple and an ACP denying access to $\{\{\mathsf{s},\mathsf{p}\}\}$ and $\{\{\mathsf{p},\mathsf{o}\}\}$, respectively.

The problem lies in that, without these security patterns, we lose the capability of forbidding and granting the access to specific subjects and objects from a triple. For example, in order to hide the telephone number of the people in a system with triples of the form (ID, :phoneNum, *number*), we need an ACP that restrict the access to the pair $\{\mathsf{p},\mathsf{o}\}$ of those triples. If the user has access to those entire triples thanks to another permission, the resulting CCP will be $\{\{\mathsf{s},\mathsf{p}\}, \{\mathsf{o}\}\}$, thus the user will have access to the numbers even though he does not know to which person they belong.

If we wanted to maintain the same CCP of the initial policy, new rules would be needed for each triple in the graph. Another possibility is changing the semantics and consider a smaller security set tree that lacks the patterns $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$, however, it is more intuitive to try to forbid the access to the subject of a triple than forbidding the access to the pair subject-predicate of it, action that can be a little confusing, making people believe that the access to the predicate is being forbidden.

5.3.2 Triple Level Coherence

The way in which we defined the permissions and policies was inspired by the work in [FFMA10]. Because of this, when the security sets of all allowed ACPs in a policy are $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ and the ones of all forbidden ACPs are $\{\{\mathsf{s}\},\{\mathsf{o}\}\}$, we have a policy that should be equivalent to one expressed with their definition. Their policies are defined as $\mathcal{P} = (\mathbb{P},\mathbb{N},ds,cr)$, where $\mathbb{P}$ and $\mathbb{N}$ are the sets of permissions granting and denying access to triples respectively, and so $\mathbb{P}$ and $\mathbb{N}$ are equivalent to our sets $\mathcal{A}$ and $\mathcal{F}$; ds and cr represent the default permission and the conflict resolution rule, both of which deny the access to triples in our policies, situation represented in both cases with the sign "–". Therefore, if all the ACPs in the set $\mathcal{A}$ of a policy have $\mathcal{S} = \{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ and all the ACPs in the set $\mathcal{F}$ of that policy have $\mathcal{S} = \{\{\mathsf{s}\},\{\mathsf{o}\}\}$, that policy is equivalent to a policy $\mathcal{P} = (\mathcal{A},\mathcal{F},-,-)$ as defined in [FFMA10].

The accessible triples of a graph $\mathcal{G}$ for the policy we defined above, using their notation, is the set $[[(\mathcal{A},\mathcal{F},-,-)]]_{\mathcal{G}} = \mathcal{T}_{\mathcal{A}} \setminus \mathcal{T}_{\mathcal{F}}$, with $\mathcal{T}_{\mathcal{A}} = \bigcup_{\mathcal{R}\epsilon\mathcal{A}}[[\mathcal{R}]]_{\mathcal{G}}$, $\mathcal{T}_{\mathcal{F}} = \bigcup_{\mathcal{R}\epsilon\mathcal{F}}[[\mathcal{R}]]_{\mathcal{G}}$ and $\mathcal{R}$ is a permission. They defined $[[\mathcal{R}]]_{\mathcal{G}}$ as the triples in $\mathcal{G}$ that are mapped from the mappings obtained from $\langle\!\langle \mathcal{R} \rangle\!\rangle_{\mathcal{G}}$, thus $[[(\mathcal{A},\mathcal{F},-,-)]]_{\mathcal{G}}$ is the set of triples that are allowed thanks to permissions minus the set of triples that are forbidden thanks to permissions.

In contrast, we defined $\mathsf{Anon}(\mathcal{G},\mathcal{P})$ as the triples in $\mathcal{G}$ whose CCP is not empty. The CCP is defined in terms of the matching permissions of the triple, and the matching permissions are defined from the mappings of the ACPs for the set of allowed and forbidden permissions. As all security sets in $\mathcal{A}$ and $\mathcal{F}$ are $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ and $\{\{\mathsf{s}\},\{\mathsf{o}\}\}$, respectively, when we compute the CCP of the triples in $\mathcal{G}$, we will be determining if the triple is allowed by the ACPs in $\mathcal{A}$ and if it is forbidden by the ACPs in $\mathcal{F}$, and the possible results from the subtraction of these sets of matching permissions are that the triple is allowed or forbidden entirely.

Therefore, our policies are semantically equivalent to the ones in [FFMA10] when all security sets in $\mathcal{A}$ are $\{\{\mathsf{s},\mathsf{p},\mathsf{o}\}\}$ and all security sets $\mathcal{F}$ are $\{\{\mathsf{s}\},\{\mathsf{o}\}\}$.

5.3.3 Policy Consistency

After the policy has been defined, we need to check if it is possible to obtain or infer data that has been forbidden when a user is performing queries over an RDF store. We will say

that a policy is inconsistent when it allows a user to access information that he should not be able to by combining allowed information. In this section we show that our policies do not have security leaks.

Policy consistency has been studied for XML Documents [BCF07]. They defined that consistent policies are the ones for which it is not possible to simulate a forbidden update through a sequence of allowed updates. Intuitively, for our policies that control read access, we expect a policy to be consistent if it is not possible to obtain forbidden data by combining the answers obtained by a series of queries over the data for which the user has access.

Since SPARQL queries do not return triples but as many columns as variables in the `SELECT`, we use SQL to combine the answers.

Definition 5.8 A policy $\mathcal{P}$ is *inconsistent* if there exists SPARQL queries $Q_1, Q_2, \ldots, Q_n$ and a SQL query Q_s such that for all RDF graphs $\mathcal{G}$:

i. There exists triple $t_1 \in Result$ and $t_1 \notin \mathsf{Anon}(\mathcal{G}, \mathcal{P})$
ii. $Result \subseteq \mathcal{G}$

where $Result = \mathbf{Ans}_{SQL}(Q_s, (\mathbf{Ans}^{\mathcal{P}}(Q_1, \mathcal{G}) \cup \mathbf{Ans}^{\mathcal{P}}(Q_2, \mathcal{G}) \cup \cdots \cup \mathbf{Ans}^{\mathcal{P}}(Q_n, \mathcal{G})))$

□

This definition states that a policy will be inconsistent if the following conditions are met: the result of combining the answers of multiple SPARQL queries through a SQL query contains at least one triple that is not present in the anonymized graph, and the result set does not contain triples that are not part of the original graph, which means that at least one triple in the result set is a hidden triple from the original RDF graph. It is worth noting that the result from a query $\mathbf{Ans}^{\mathcal{P}}(Q_k, \mathcal{G})$, for k between 1 and n, is the result given by query $\mathbf{Ans}(Q_k, \mathsf{Anon}(\mathcal{G}, \mathcal{P}))$, which means that to perform the query, we first need to obtain the anonymized graph.

It is required that the result set does not contain nonexistent triples. If we allowed them, all policies could be inconsistent because performing a cartesian join between subjects, predicates and objects could be enough to get triples that could be hidden from the user, although most of the triples generated in this way would be triples that make no sense. Since the user cannot distinguish between the correct and incorrect triples, there is no real

triple leak.

We know that there are policies defined at triple level, like the ones defined in [FFMA10], that are consistent due to the way they are defined, so in order to ensure that all policies defined with our model are consistent, we start by analyzing the case when all defined permissions are applied over entire triples. When the access is controlled at triple level, the queries are computed over the graph without considering hidden triples, so that there is no way for the answers to contain forbidden data. Trying to infer new triples from the allowed data is the same as guessing.

If we allow access to triples that only have the subject or object visible, a user could know about their presence in the graph but he can assume nothing about the predicate and object/subject being hidden. To be able to notice the existence of subjects and objects he was not aware of but has the right to access them, a user would have to ask for all the triples that he can access that have blank nodes as predicates. Combining this set of triples with the rest of the triples the user can see will not leak hidden information. This is because mixing the subjects and objects that are alone with predicates obtained from other triples will produce triples that have no guaranty of corresponding to the hidden data.

Thereby, if we also allow users to see the others parts of a triple, by allowing access to the triples of the form $(\mathsf{s}, \mathsf{p}, \_{:}\lambda)$ and $(\_{:}\lambda, \mathsf{p}, \mathsf{o})$, a user could make assumptions about the relations between the triples that lack a subject and the ones that lack an object if they have the same predicate, and between the triples that lack a subject or object and the triples that only have visible their subject or object. Considering that all $\_{:}\lambda$ nodes will be fresh blank nodes, they would need first to determine what triples have a blank node as the subject or object and what triples have only a subject or object, to later combine them to create new triples. There will be some particular cases in which these new triples will be in the original graph, however there is no way for the user to know if they are indeed triples from the graph or false information, thus there is no data leak. If the triples $(\mathsf{s}, \_{:}\lambda, \_{:}\lambda)$, $(\_{:}\lambda, \_{:}\lambda, \mathsf{o})$ have predicates that are not present in the anonymized graph, the users cannot infer the hidden data.

The policies defined with our model are always consistent, even when the RDF graphs are of a size considerably small. This is thanks to the definition of $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ in terms of the CCP and, as stated before, the form of the security tree. The CCP ensures that each triple in $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ hides the necessary information, while the security tree ensures

that $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ cannot have triples of the form $(\mathsf{s}, \_{:}\lambda, \mathsf{o})$ whose predicate can be easy to guess depending on the meaning of the subject and object, for example, from the triple (Dave, $\_{:}\lambda$, Student) we can infer that the predicate is rdf:type or is_a, while in (Dave, $\_{:}\lambda$, *18*) the predicate could be age, approved_classes or day_of_birth, predicates that are not related but, depending on the type of data being stored on the graph, some of them could be discarded. Thus, when the size of the graph is considerably small, one could think that it is easy to create new triples and discard the triples that contain false information, however, the policy is still consistent because triples created this way are not security leaks, but educated assumptions.

5.3.4 The Need for Forbidden Permissions

To obtain the anonymized graph generated as result of a policy, we determine the set of CCP of each triple in the graph. If we consider that the CCP of a triple consists only of its allowed parts, we could think in rewriting the policies defined with allowed and forbidden permissions as policies that only have allowed permissions. However, this is not possible because policies without forbidden permissions are not as expressive as the policies that have them.

The main factor that prevent us from rewriting the policies is the fact that RDF graphs may change over time. When the original graph changes, the anonymized graph of each role will change according to its policy, adding triples from the original graph, wherein some of their parts could be hidden, and deleting triples from it.

The new triples added and deleted could make the ACPs act over a set of triples that is different from the one before. If we remove the possibility to state explicitly what triples need to be hidden, it is possible that some of the triples should be hidden for some roles but they are not because there is no way to state that they must not be visible.

Example 5.10 Considering the graph and ACPs from section 5.2.1, if we apply policy $\mathcal{P}_{ex6} = (\mathcal{A}_{ex6}, \mathcal{F}_{ex6})$, with $\mathcal{A}_{ex6} = \{\rho_{A4}\}$ and $\mathcal{F}_{ex6} = \{\rho_{F1}, \rho_{F2}\}$, to the graph $\mathcal{G}_{ex}$, we will get the anonymized graph $\mathcal{G}_{ex6}$ as shown in Figure 5.5.

ρ_{A4} = APPLY $\mathcal{S}_{A4}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z)$ $\qquad\qquad$ $\mathcal{S}_{A4} = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$

ρ_{F1} = APPLY $\mathcal{S}_{F1}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z), (?z, ?v, ?w)$ $\qquad\qquad$ $\mathcal{S}_{F1} = \{\{\mathsf{s}\}\}$

ρ_{F2} = APPLY $\mathcal{S}_{F2}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z), (?m, ?n, ?y)$ $\quad\quad$ $\mathcal{S}_{F2} = \{\{\mathsf{o}\}\}$

s	p	o
a	b	c
d	c	e
e	f	b

$\Rightarrow$

s	p	o
a	b	_:λ1
e	f	b

Figure 5.5: Graph $\mathcal{G}_{ex}$ (left) and its anonymized graph $\mathcal{G}_{ex6}$ (right)

The user will have access to one entire triple in the new graph and to the pair (subject, predicate) of the first triple of the original graph. The triple (d, c, e) will not be accessible because the access to $\{\mathsf{s}\}$ and $\{\mathsf{o}\}$ has been forbidden. Therefore, we could attempt to rewrite this policy as $\mathcal{P}_{new} = (\mathcal{A}_{new}, \mathcal{F}_{new})$, with $\mathcal{A}_{new} = \{\rho_{NA1}, \rho_{NA2}\}$ and $\mathcal{F}_{new} = \varnothing$.

ρ_{NA1} = APPLY $\mathcal{S}_{NA1}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z)$, $?x$ = a $\quad\quad$ $\mathcal{S}_{NA1} = \{\{\mathsf{s}, \mathsf{p}\}\}$

ρ_{NA2} = APPLY $\mathcal{S}_{NA2}$ SELECT $?x, ?y, ?z$ WHERE $(?x, ?y, ?z)$, $?y$ = f $\quad\quad$ $\mathcal{S}_{NA2} = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$

Now, if we add the triple (c, f, g) to the original graph, and name it $\mathcal{G}_{exNew}$, the following is going to happen: triple (a, b, c) needs to be entirely hidden in the anonymized graph, because it is now affected by ρ_{F1} and ρ_{F2}, and triple (c, f, g) needs to be added to it, because it is affected by ρ_{A4}. Even though (c, f, g) will be added without problems to the anonymized graph under policy $\mathcal{P}_{new}$, thanks to permission ρ_{NA2}, (a, b, _:λ1) will not be removed from the anonymized graph controlled by the rewritten policy. Therefore, it is not feasible to rewrite a policy without using forbidden permissions in this way. These graphs can be seen in Figure 5.6.

s	p	o
a	b	c
d	c	e
e	f	b
c	f	g

s	p	o
e	f	b
c	f	g

s	p	o
a	b	_:λ1
e	f	b
c	f	g

Figure 5.6: Graph $\mathcal{G}_{exNew}$ (left), Anon$(\mathcal{G}_{exNew}, \mathcal{P}_{ex6})$ (middle) and Anon$(\mathcal{G}_{exNew}, \mathcal{P}_{new})$ (right). □

To make rewriting the policies to have only allowed permissions, we would need a more expressive language. If we add to the CQs the possibility of using the MINUS statement, we could rewrite the ACPs from the $\mathcal{A}$ set into permissions that do not affect forbidden triples. Continuing the above example, if we allow access to the entire triples with an ACP, and

then forbid some parts of them, we are going to have to create the following permissions for our rewritten policy:

- A permission that allows access to entire triples, setting aside the forbidden triples. This will be achieved by using the conditions of all forbidden permissions with `MINUS` statements. For ρ_{Mi1}, the first `MINUS` will delete the triples that are affected by ρ_{F1}, and the second `MINUS` the ones that are affected by ρ_{F2}.
  ρ_{Mi1} = `APPLY` $\mathcal{S}_{Mi1}$ `SELECT` $?x, ?y, ?z$ `WHERE` $\{(?x, ?y, ?z)$ `MINUS` $\{(?x, ?y, ?z), (?z, ?v, ?w)\}$ `MINUS` $\{(?x, ?y, ?z), (?m, ?n, ?y)\}\}$ $\mathcal{S}_{Mi1} = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$
- For each forbidden permission in $\mathcal{F}$, if its security set is not $\{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$, we will allow access to the complement of its security set with a new permission, excluding the triples affected by forbidden permissions sharing this security set with `MINUS` statements. For example, the security set of the rewritten permission of ρ_{F1} will be $\{\mathsf{p}, \mathsf{o}\}$, `MINUS` the other forbidden permissions with security sets $\{\mathsf{o}\}$, $\{\mathsf{p}, \mathsf{o}\}$, because they could affect the same triples .
- For forbidden permission that affect triples with a security set $\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}$, two new permissions need to be created, one that affects $\{\mathsf{s}, \mathsf{p}\}$ and other that affects $\{\mathsf{p}, \mathsf{o}\}$, following the same rules as above.

Example 5.11 (example 5.10 continued) The policy $\mathcal{P}_{ex6}$ would be rewritten as $\mathcal{P}_{new2} = (\mathcal{A}_{new2}, \mathcal{F}_{new2})$, with $\mathcal{A}_{new2} = \{\rho_{Mi1}, \rho_{Mi2}, \rho_{Mi3}\}$ and $\mathcal{F}_{new2} = \varnothing$.

ρ_{Mi1} = `APPLY` $\mathcal{S}_{Mi1}$ `SELECT` $?x, ?y, ?z$ `WHERE` $\{(?x, ?y, ?z)$ `MINUS` $\{(?x, ?y, ?z), (?z, ?v, ?w)\}$ `MINUS` $\{(?x, ?y, ?z), (?m, ?n, ?y)\}\}$ $\mathcal{S}_{Mi1} = \{\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}\}$

ρ_{Mi2} = `APPLY` $\mathcal{S}_{Mi2}$ `SELECT` $?x, ?y, ?z$ `WHERE` $(?x, ?y, ?z), (?z, ?v, ?w)$ $\mathcal{S}_{Mi2} = \{\{\mathsf{p}, \mathsf{o}\}\}$

ρ_{Mi3} = `APPLY` $\mathcal{S}_{Mi3}$ `SELECT` $?x, ?y, ?z$ `WHERE` $(?x, ?y, ?z), (?m, ?n, ?y)$ $\mathcal{S}_{Mi3} = \{\{\mathsf{s}, \mathsf{p}\}\}$

□

A similar process has to be executed for each ACP in the $\mathcal{A}$ set. The difference is that we need to properly consider the security set of the ACP being rewritten and its relation with the security sets of the ACPs in $\mathcal{F}$. So, in the worst case, if all the ACPs in $\mathcal{A}$ allow access to the set $\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}$, they would have to be rewritten into five permissions each, one for each security pattern. To calculate the size of all the new ACPs we are going to denote by $|\mathcal{F}|_b$ the number of ACPs in $\mathcal{F}$ explicitly forbidding the security pattern b. The size of all the

new ACPs, for the $\{\mathsf{s}, \mathsf{p}, \mathsf{o}\}$ set, will grow by adding $|\mathcal{F}|$ `MINUS` statements. The size of the ACPs, for the $\{\mathsf{s}, \mathsf{p}\}$ set, will grow by adding $|\mathcal{F}|_{sp} + |\mathcal{F}|_{s}$ `MINUS` statements. The size of the ACPs, for the $\{\mathsf{p}, \mathsf{o}\}$ set, will grow by adding $|\mathcal{F}|_{po} + |\mathcal{F}|_{o}$ `MINUS` statements. The size of the ACPs, for the set $\{\mathsf{o}\}$, will grow by adding $|\mathcal{F}|_{o}$ `MINUS` statements, and for the $\{\mathsf{s}\}$ set, the size will grow by adding $|\mathcal{F}|_{s}$ `MINUS` statements. Thus, the size of the requests that need to be performed to compute each ACP will be $|\mathcal{F}| + |\mathcal{F}|_{sp} + 2 \times |\mathcal{F}|_{s} + |\mathcal{F}|_{po} + 2 \times |\mathcal{F}|_{o}$, which is at most $3 \times |\mathcal{F}|$.

Therefore, in this way, we can rewrite the original policy as a policy that only contains allowed permissions. Nonetheless, we can easily note that the process of rewriting a single permission from $\mathcal{A}$ gets more complex the bigger the set $\mathcal{F}$. Another problem would be that our new policy becomes more expensive to compute, because the quantity of comparisons that need to be performed depends of the size of the `WHERE` statement. And if we consider an enforcement mechanism that stores all anonymized graphs, there are no reasons to increment the time it takes to generate them, if the space used to store the graphs will be exactly the same.

Chapter 6

Policy Enforcement

In most settings, policy enforcement mechanism are based on either views or query rewriting. When deciding which approach is better suited the following variables need to be considered:

- The time it takes to compute the answers.
- Frequency with which the data is updated: materialized views would need to be recomputed. However, some incremental strategies can be considered.
- Existence and size of the rewriting.
- Space used by the materialized view.

We are adopting the views approach, by specifying an enforcement mechanism that uses directly the semantics as described before. This approach has the advantage of being able to solve queries faster than an implementation based on query rewriting, since the queries do not have to be preprocessed to be computed. Besides, with our approach, the forbidden triples are not used to obtain the answers. Thus, no time is wasted filtering triples from the answer set.

One of the disadvantages of materialized views is the extra amount of data since both the original graph and the materialized views for each role are stored. In contrast, mechanisms using the query rewriting approach would not require that much space for answering queries, since they perform a modified query on the original graph. To reduce the space needed by the views approach, subgraphs containing the triples that are shared by multiple roles could be defined and used, instead of repeating those triples in each graph.

A mechanism based on the query rewriting approach needs to rewrite the queries in a way that it does not consider the forbidden information. A simple query rewriting mechanism could add, to each query, a filter for each ACP in the set of forbidden permission of the policy. Then, it would need to add filters, that consider the set of allowed permissions, in order to separate the triples that are accessible from the ones forbidden by default. If

the process of computing each of these filters is like performing a query on the graph, the filtering process could be done in a reasonable time when the size of the policy is small, but as the size of the policy increases, computing a single rewritten query becomes a more demanding task because, in this case, the number of filters is directly proportional to the number of ACPs.

Thus, we consider a view based policy enforcement algorithm. Indeed, we will use the anonymized graph as the view. Hence, to enforce an access control policy, a new RDF Graph will be created for each defined role, from the original graph, to be used by users to perform their queries, accordingly with their roles. In Section 7 we study how to decrease the size of anonymized graphs in the presence of several roles.

6.1 Enforcement Algorithm

In order to compute the anonymized graph, we first need an algorithm to compute the set of allowed matching permissions $\mathcal{M}(\mathcal{G}, \mathcal{A}, t)$ and forbidden matching permissions $\mathcal{M}(\mathcal{G}, \mathcal{F}, t)$ of each triple. Given a graph $\mathcal{G}$ and a generic set of ACPs Π, Algorithm 1, called $\mathbf{ComputeM}$, creates a function r such that given a triple t, $r(t) = \mathcal{M}(\mathcal{G}, \Pi, t)$. For efficiency purposes, the matching permissions for all the triples in the graph are computed at the same time. Algorithm $\mathbf{ComputeM}$ starts by initializing the function r to return an empty set for all triples (line 1). Next, it computes the answers to all the queries in the ACPs in Π (lines 2-4) and adds the respective security set to $r(t)$ for every triple t in the result (lines 5-6).

Now, the anonymized graph $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ can be calculated using Algorithm 2, named $\mathbf{Anonymize}$. The algorithm starts by creating an empty graph $\mathcal{G}'$ to which we will add all the triples of the anonymized graph (line 1). Next, it computes functions $r_{\mathcal{A}}$ and $r_{\mathcal{F}}$ that

Algorithm 1 $\mathbf{ComputeM}$ $(\mathcal{G}, \Pi)$

Input: RDF Graph $\mathcal{G}$, set of permissions Π
Output: A function r such that given a triple t, $r(t) = \mathcal{M}(\mathcal{G}, \Pi, t)$

1: $r \leftarrow$ function that given a triple returns an empty set
2: **for all** (`APPLY` $\mathcal{S}$ `SELECT` (*?x*, *?y*, *?z*) `WHERE` $\mathcal{TP}, \mathcal{C}$) $\in \Pi$ **do**
3: $\quad \mathcal{Q} \leftarrow$ `SELECT` (*?x*, *?y*, *?z*) `WHERE` $\mathcal{TP}, \mathcal{C}$
4: $\quad \mathcal{T} \leftarrow \mathbf{Ans}(\mathcal{Q}, \mathcal{G})$
5: $\quad$ **for all** $t \in \mathcal{T}$ **do**
6: $\quad\quad r(t) \leftarrow r(t) \cup \mathcal{S}$
return r

Algorithm 2 $\mathbf{Anonymize}\ (\mathcal{G}, \mathcal{P})$

Input: RDF Graph $\mathcal{G}$, Policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$
Output: A graph $\mathcal{G}'$ that represents $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$

```
1:  G' ← ∅
2:  r_A ← ComputeM(G, A)
3:  r_F ← ComputeM(G, F)
4:  for all t ∈ G do
5:      if r_A(t) ≠ ∅ then
6:          S_A ← r_A(t)^⊆
7:          if r_F(t) ≠ ∅ then
8:              S_F ← r_F(t)^⊇
9:              S_A ← S_A ∖ S_F
10:         S_A ← S_A^↓
11:         for all b ∈ S_A do
12:             t' ← (_:λ, _:λ, _:λ)
13:             if s ∈ b then t'[s] ← t[s]
14:             if p ∈ b then t'[p] ← t[p]
15:             if o ∈ b then t'[o] ← t[o]
16:             G' ← G' ∪ {t'}
    return G'
```

for each triple return the set of security patterns that are respectively allowed and forbidden (lines 2-3). The CCP for each triple that has at least one allowed matching permission can be computed using $r_{\mathcal{A}}$ and $r_{\mathcal{F}}$ (lines 6-10). Once these permissions have been determined, an anonymized triple is computed for each security pattern (lines 11-15) and added to the anonymized graph (line 16).

Now that we are able to determine the anonymized graph of a graph $\mathcal{G}$ with respect to a policy $\mathcal{P}$, we can perform queries over $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$. As we are interested in the enforcement method for multiple queries, Algorithm 6, called **Enforcement**, will be used to compute the answers to a list of conjunctive queries $\mathcal{CQ} = [\mathrm{CQ}_1, \ldots, \mathrm{CQ}_n]$ that are performed on the protected graph. This algorithm starts by obtaining the anonymized graph $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ and creating an empty list of URIs L (lines 1-2), list to which we will add all the URIs that will be given as result. Then, it computes the answers to all the queries in the list $\mathcal{CQ}$ (lines 3-4) and adds the rest of the results to the list of URIs L (line 5).

Algorithm 3 $\mathbf{Enforcement}\ (\mathcal{G}, \mathcal{P}, \mathcal{CQ})$

Input: An RDF Graph $\mathcal{G}$, a policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$ and a list of conjunctive queries $\mathcal{CQ} = [\text{CQ}_1, \ldots, \text{CQ}_n]$
Output: A list of URIs L

```
G' ← Anonymize(G, P)
L ← empty list
for all CQ ∈ CQ do
    R ← Ans(CQ, G')
    L ← Add R at the end of the list
return L
```

6.2 Time Cost Analysis

We will study the cost of making m queries over a graph $\mathcal{G}$ with n triples and which is under a policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$, where $|\mathcal{A}|$ is a and $|\mathcal{F}|$ is f. Figure 6.1 contains a summary of the parameters used in this section. This cost is associated to the three algorithms presented above, and to be able to obtain it, we need to determine first the cost of performing any query.

Variables	Meaning
m	number of conjunctive queries in $\mathcal{CQ}$, i.e. $\|\mathcal{CQ}\|$
n	number of triples in a graph $\mathcal{G}$, i.e. $\|\mathcal{G}\|$
a	number of ACP in $\mathcal{A}$, i.e. $\|\mathcal{A}\|$
f	number of ACP in $\mathcal{F}$, i.e. $\|\mathcal{F}\|$
pat(CQ)	number of triple patterns in a query CQ
com(CQ)	number of comparisons in a query CQ
$p_{max}^{\mathcal{CQ}}$	maximum number of patterns in a query in the list of queries $\mathcal{CQ}$
$c_{max}^{\mathcal{CQ}}$	maximum number of comparisons in a query in the list of queries $\mathcal{CQ}$
$q_{\text{CQ}}^{\mathcal{G}}$	running time of $\mathbf{Ans}$ (CQ,$\mathcal{G}$)
p_{max}^{Π}	maximum number of patterns in an ACP in Π
c_{max}^{Π}	maximum number of comparisons in an ACP in Π
p_{max}	$max\{p_{max}^{\mathcal{CQ}}, p_{max}^{\Pi}\}$
c_{max}	$max\{c_{max}^{\mathcal{CQ}}, c_{max}^{\Pi}\}$
$mp_{\Pi}^{\mathcal{G}}$	running time of $\mathbf{ComputeM}\ (\mathcal{G}, \Pi)$
$h_{\mathcal{P}}^{\mathcal{G}}$	running time of $\mathbf{Anonymize}\ (\mathcal{G}, \mathcal{P})$

Figure 6.1: Summary of parameters.

6.2.1 $\mathbf{Ans}(\mathbf{CQ}, \mathcal{G})$

We want the cost of answering query CQ over $\mathcal{G}$ in terms of the size of the query and the graph. The size of the query is measured in terms of the numbers of triple patterns and comparisons in it, denoted pat(CQ) and com(CQ) respectively. The size of the graph is the number of triples in it, this is, $n = |\mathcal{G}|$.

The cost of determining if a triple matches a pattern is constant, so checking if a pattern matches n triples in a graph is $O(n)$. In the worst case, each pattern of the query would match the n triples and we would need to compare them all to check for the common variables in the triple patterns to determine the mapping that satisfy all of them. Thus, the cost of matching all the triple patterns is $O(n^{\mathsf{pat(CQ)}})$. For every mapping (which in the worst case are $O(n^{\mathsf{pat(CQ)}})$), we also need to check the satisfaction of the comparisons in the query. As a consequence, $\mathbf{Ans}(\mathbf{CQ}, \mathcal{G})$ runs in $O(\mathsf{com(CQ)} \times n^{\mathsf{pat(CQ)}})$. We will denote this running time by $q^{\mathcal{G}}_{\mathrm{CQ}}$.

6.2.2 $\mathbf{ComputeM}\ (\mathcal{G}, \Pi)$

Line 1 of Algorithm 1 is $O(n)$ because it initializes the function for every triple. The **for all** statement that follows will run $|\Pi|$ times. Line 3 runs in constant time and line 4 is $q^{\mathcal{G}}_{\mathcal{Q}}$. The **for all** statement in line 5 will be executed $|\mathcal{T}|$ which, in the worst case, is n. The cost of adding security sets to the function r is constant (line 6). If we denote by p^{Π}_{max} the maximum number of patterns in an ACP in Π, and by c^{Π}_{max} the maximum number of comparisons in an ACP in Π, we get that the cost of running $\mathbf{ComputeM}\ (\mathcal{G}, \Pi)$ is $O(|\Pi| \times c^{\Pi}_{max} \times n^{p^{\Pi}_{max}})$. We will denote this running time by $mp^{\mathcal{G}}_{\Pi}$. If we consider data complexity, $\mathbf{ComputeM}$ runs in polynomial time.

6.2.3 $\mathbf{Anonymize}\ (\mathcal{G}, \mathcal{P})$

Creating an empty graph takes a constant amount of time (line 1). Line 2 runs in $mp^{\mathcal{G}}_{\mathcal{A}} = O(a \times c^{\mathcal{A}}_{max} \times n^{p^{\mathcal{A}}_{max}})$ and line 3 runs in $mp^{\mathcal{G}}_{\mathcal{F}} = O(f \times c^{\mathcal{F}}_{max} \times n^{p^{\mathcal{F}}_{max}})$. The **for all** statement in line 4 will be executed n times. Lines 5 to 10 take a constant amount of time since $S_{\mathcal{A}}$ and $S_{\mathcal{F}}$ can contain at most five security sets. Lines 11 to 16 run in constant time as well, because here, $S_{\mathcal{A}}$ contains at most two b, creating a triple and replacing its subject,

predicate and/or object with the values from some t in $\mathcal{G}$, and adding the new triple to the graph $\mathcal{G}'$ take a constant amount of time. Therefore, the cost of running **Anonymize** $(\mathcal{G}, \mathcal{P})$ is $O(a \times c_{max}^{\mathcal{A}} \times n^{p_{max}^{\mathcal{A}}} + f \times c_{max}^{\mathcal{F}} \times n^{p_{max}^{\mathcal{F}}})$. We can simplify this formula considering the maximum number of patterns and comparisons in the policy $\mathcal{P}$, instead of the maximum number of patterns and comparisons in the sets $\mathcal{A}$ and $\mathcal{F}$. Thus, we obtain that the cost of running **Anonymize** $(\mathcal{G}, \mathcal{P})$ is $O((a + f) \times c_{max}^{\mathcal{P}} \times n^{p_{max}^{\mathcal{P}}})$. We will denote this running time by $h_{\mathcal{P}}^{\mathcal{G}}$.

6.2.4 Enforcement $(\mathcal{G}, \mathcal{P}, \mathcal{CQ})$

The running time of algorithm **Enforcement** depends on $h_{\mathcal{P}}^{\mathcal{G}}$ and the cost of performing every query in $\mathcal{CQ}$ over graph $\mathcal{G}$.

Line 1 is $h_{\mathcal{P}}^{\mathcal{G}} = O((a + f) \times c_{max}^{\mathcal{P}} \times n^{p_{max}^{\mathcal{P}}})$, because it computes the anonymized graph of $\mathcal{G}$. Creating an empty list takes a constant amount of time (line 2). The **for all** statement in line 3 will be executed m times. The running time of answering a query is $q_{\text{CQ}}^{\mathcal{G}'}$, and considering that, in the worst case $|\mathcal{G}'| = 2 \times |\mathcal{G}|$ (see Section 6.3), we have $q_{\text{CQ}}^{\mathcal{G}'}$ equals to $O(c_{max}^{\text{CQ}} \times (2n)^{p_{max}^{\text{CQ}}})$ (line 4). In line 5, adding the answers to the list takes a constant amount of time. So, the cost of running **Enforcement** $(\mathcal{G}, \mathcal{P}, \mathcal{CQ})$ is $O((a + f) \times c_{max}^{\mathcal{P}} \times n^{p_{max}^{\mathcal{P}}} + m \times c_{max}^{\text{CQ}} \times (2n)^{p_{max}^{\text{CQ}}})$.

We can simplify this considering the maximum number of patterns and comparisons between the conjunctive queries in $\mathcal{CQ}$ and the ACPs in the policy $\mathcal{P}$, with p_{max} and c_{max}, respectively. At the end, the cost of running **Enforcement** $(\mathcal{G}, \mathcal{P}, \mathcal{CQ})$ will be $O((a + f + m) \times c_{max} \times (2n)^{p_{max}})$, which is polynomial on the size of $\mathcal{G}$ under data complexity.

We calculated the cost of the enforcement algorithm for the case in which we have to anonymize the graph each time a set of questions is performed, however, we are supposed to store the anonymized graphs, and recompute them only when the original graph is modified. In this way, the cost of the enforcement algorithm is reduced, so when a user with a certain role performs a set of CQ over a graph that already has an anonymized graph for that role, we can ignore the cost of anonymizing it. Thus, the cost of enforcing the policy with algorithm **Enforcement**, for this case, will be $O(m \times c_{max} \times (2n)^{p_{max}})$.

6.3 Space analysis

For a number of roles r, each with its respective policy, and a graph $\mathcal{G}$ which contains n triples, the space that an anonymized graph will use is $2 \times |\mathcal{G}| = 2 \times n$, this is because each triple from graph $\mathcal{G}$ can be present in $\mathsf{Anon}(\mathcal{G})$ and could use the space of two triples, if the security set applied to it is one of the following: $\{\{\mathsf{s}\}, \{\mathsf{o}\}\}$, $\{\{\mathsf{s}\}, \{\mathsf{p}, \mathsf{o}\}\}$, $\{\{\mathsf{s}, \mathsf{p}\}, \{\mathsf{o}\}\}$ or $\{\{\mathsf{s}, \mathsf{p}\}, \{\mathsf{p}, \mathsf{o}\}\}$. So the maximum total space that will be used to store all the anonymized graphs, in the worst case, is twice the size of the original graph multiplied by the number of roles, i.e., $O(r \times 2 \times |\mathcal{G}|) = O(2 \times r \times n) = O(r \times n)$.

Chapter 7

Management of Multiple Roles

In this section we study the problem of dealing with multiple roles that might have different policies and, therefore, different anonymized graphs. Considering that every access control policy is defined for a specific role, and that our policy enforcement algorithm relies in creating and storing an additional graph for each policy, there exists the possibility that any triple could be present in multiple anonymized graphs. This clear duplicity of information results in too much storage space wasted. Thus, we need to consider ways in which we can reduce the quantity of wasted space.

In order to efficiently store all the anonymized versions of a graph, we could determine subgraphs which contain the triples that are shared between the anonymized graphs of different roles, so when a user needs his data, the system will answer his queries from the anonymized graph obtained from composing all the subgraphs that are part of it.

We want to be able to store all the anonymized graphs in a compact way. To do this, we will first treat all blank nodes created in the process of anonymizing the graphs as a constant λ, this is a triple (a,b,_:λ1) which is added when anonymizing is equal to (a,b,λ). A problem with this replacemente is that if the anomymized graph has, for example, (a,b,_:λ1) and (a,b,_:λ2), we would get the same triple (a,b,λ). To avoid loosing the information that the anonymized graph has two such triples, we will use a multiset (bag) semantics to represent the graph with blank nodes replaced by λ. Also, to simplify presentation we will assume next that the graph $\mathcal{G}$ has no blank nodes. The definitions can be extended by distinguishing the blank nodes added from anonymization from the ones present in the original graph.

Definition 7.1 Given a graph $\mathcal{G}$, and a policy $\mathcal{P}$, the *multiset anonymized graph*, denoted $\mathsf{Anon}^{ms}(\mathcal{G}, \mathcal{P})$, is multiset obtained from $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ by replacing each triple in $\mathsf{Anon}(\mathcal{G}, \mathcal{P})$ by its version with blank nodes replaced by λ. □

Now, as a first step to represent the anonymized graph in a compact way we want to represent the data shared by several of them.

Definition 7.2 Given an RDF Graph $\mathcal{G}$, a set of policies Γ, and a subset $\Gamma' \subseteq \Gamma$, the anonymized graph for Γ' is a multiset $\mathsf{Anon}(\mathcal{G}, \Gamma') = \bigcap_{\mathcal{P} \in \Gamma'} \mathsf{Anon}^{ms}(\mathcal{G}, \mathcal{P})$. □

If we want to efficiently store the anonymized graphs for subsets of Γ we need to get rid of the redundancy between multisets $\mathsf{Anon}(\mathcal{G}, \Gamma')$ and $\mathsf{Anon}(\mathcal{G}, \Gamma'')$ when $\Gamma' \subseteq \Gamma''$. For example, if $t \in \mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_1\})$ and $t \in \mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\})$ it is enough to store it in $\mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\})$ to reduce space. Thus, we define the reduced anonymized graph which can be recursively defined as:

Definition 7.3 Given a graph $\mathcal{G}$, a set of policies Γ, and a subset $\Gamma' \subseteq \Gamma$, the *reduced anonymized graph* can be recursively defined as:

$$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma') = \begin{cases} \mathsf{Anon}(\mathcal{G}, \Gamma') & \text{for } \Gamma' = \Gamma \\ \mathsf{Anon}(\mathcal{G}, \Gamma') \setminus (\biguplus_{\Gamma' \subsetneq \Gamma''} \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma'')) & \text{else} \end{cases}$$

where $\biguplus$ is the multiset sum. □

Note that then we can compute the anonymized graph for each policy $\mathcal{P} \in \Gamma$ as $\mathsf{Anon}(\mathcal{G}, \mathcal{P}) = \biguplus_{\Gamma' \subseteq \Gamma, \mathcal{P} \in \Gamma'} \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma')$.

Example 7.1 Consider a graph $\mathcal{G}$ and $\Gamma = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$. We will determine the subgraphs $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2\})$ and $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_3\})$. The graph $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\})$will only contain triples that are shared by all the anonymized graphs, i.e. $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1)$, $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2)$ and $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_3)$. Graph $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\})$ will contain triples that are shared exclusively by the graphs $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1)$ and $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2)$. Graph $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\})$ will contain triples that are shared exclusively by $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2)$ and $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_3)$, while graph $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_3\})$ will store the triples shared exclusively by $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1)$ and $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_3)$. Last, graphs $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2\})$ and $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_3\})$ will include the triples that are exclusive to each anonymized graph, i.e. $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1)$, $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2)$ and $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_3)$ respectively. This is:

$$\begin{aligned}
&\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) = \mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) = \mathsf{Anon}(\mathcal{G}, \mathcal{P}_1) \cap \mathsf{Anon}(\mathcal{G}, \mathcal{P}_2) \cap \mathsf{Anon}(\mathcal{G}, \mathcal{P}_3), \\
&\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\}) = \mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\}) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) \\
&\qquad\qquad = \{\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1) \cap \mathsf{Anon}(\mathcal{G}, \mathcal{P}_2)\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}), \\
&\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\}) = \mathsf{Anon}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\}) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) \\
&\qquad\qquad = \{\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2) \cap \mathsf{Anon}(\mathcal{G}, \mathcal{P}_3)\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}),
\end{aligned}$$

$$\begin{aligned}
\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_3\}) &= \mathsf{Anon}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_3\}) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\}) \\
&= \{\mathsf{Anon}(\mathcal{G},\mathcal{P}_1) \cap \mathsf{Anon}(\mathcal{G},\mathcal{P}_3)\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\}), \\
\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1\}) &= \{\{\mathsf{Anon}(\mathcal{G},\mathcal{P}_1) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\})\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2\})\} \setminus \\
&\quad \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_3\}), \\
\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_2\}) &= \{\{\mathsf{Anon}(\mathcal{G},\mathcal{P}_2) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\})\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2\})\} \setminus \\
&\quad \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_2,\mathcal{P}_3\}), \\
\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_3\}) &= \{\{\mathsf{Anon}(\mathcal{G},\mathcal{P}_3) \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\})\} \setminus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_2,\mathcal{P}_3\})\} \setminus \\
&\quad \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_3\}).
\end{aligned}$$

□

Example 7.2 (example 7.1 continued) Consider graphs $\mathsf{Anon}(\mathcal{G},\mathcal{P}_1)$, $\mathsf{Anon}(\mathcal{G},\mathcal{P}_2)$ and $\mathsf{Anon}(\mathcal{G},\mathcal{P}_3)$ as defined below. We will determine all the necessary subgraphs to compute $\mathsf{Anon}(\mathcal{G},\mathcal{P}_1)$, $\mathsf{Anon}(\mathcal{G},\mathcal{P}_2)$, and $\mathsf{Anon}(\mathcal{G},\mathcal{P}_3)$ using space efficiently.

$\mathsf{Anon}(\mathcal{G},\mathcal{P}_1)$ $\quad\Rightarrow\quad$ $\mathsf{Anon}^{ms}(\mathcal{G},\mathcal{P}_1)$

s	p	o
_:λ1	rdf:type	ex:Student
_:λ2	rdf:type	ex:Student
_:λ3	rdf:type	ex:Student
&c	foaf:firstName	*Allen*

s	p	o
λ	rdf:type	ex:Student
λ	rdf:type	ex:Student
λ	rdf:type	ex:Student
&c	foaf:firstName	*Allen*

$\mathsf{Anon}(\mathcal{G},\mathcal{P}_2)$ $\quad\Rightarrow\quad$ $\mathsf{Anon}^{ms}(\mathcal{G},\mathcal{P}_2)$

s	p	o
&c	rdf:type	ex:Teacher
_:λ4	rdf:type	ex:Student
_:λ5	rdf:type	ex:Student

s	p	o
&c	rdf:type	ex:Teacher
λ	rdf:type	ex:Student
λ	rdf:type	ex:Student

$\mathsf{Anon}(\mathcal{G},\mathcal{P}_3)$ $\quad\Rightarrow\quad$ $\mathsf{Anon}^{ms}(\mathcal{G},\mathcal{P}_3)$

s	p	o
_:λ6	rdf:type	ex:Student
&a	foaf:firstName	*William*
&b	foaf:firstName	*Emma*

s	p	o
λ	rdf:type	ex:Student
&a	foaf:firstName	*William*
&b	foaf:firstName	*Emma*

Following the rules states previously, we will obtain $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2,\mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_2\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_2,\mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1,\mathcal{P}_3\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_1\})$, $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_2\})$ and $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G},\{\mathcal{P}_3\})$.

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\})$

s	p	o
λ	rdf:type	ex:Student

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\})$

s	p	o
λ	rdf:type	ex:Student

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1\})$

s	p	o
λ	rdf:type	ex:Student
&c	foaf:firstName	*Allen*

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2\})$

s	p	o
&c	rdf:type	ex:Teacher

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_3\})$

s	p	o
&a	foaf:firstName	*William*
&b	foaf:firstName	*Emma*

$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\})$ and $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_3\})$ are empty. If a user belongs to a role that is under a policy $\mathcal{P}_1$, queries will be computed over the graph $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_1) = \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1\})$. For a user under policy $\mathcal{P}_2$, his queries will be computed over the graph $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_2) = \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2\})$. Last, the queries of a user under policy $\mathcal{P}_3$ will be computed over the graph $\mathsf{Anon}(\mathcal{G}, \mathcal{P}_3) = \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_1, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_2, \mathcal{P}_3\}) \uplus \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \{\mathcal{P}_3\})$. □

We will now modify the enforcement algorithms presented in Chapter 6 to deal with multiple roles.

7.1 Enforcement Algorithms for Multiple Roles

Now that we proposed a way to store the anonymized graphs more efficiently, we need a method to create these graphs. Initially, we could directly use our enforcement algorithm to obtain the corresponding anonymized graphs, and then compute the subgraphs required to reduce the space used by the data. However, it would be less problematic if we could obtain these subgraphs without computing the anonymized graphs, and we would require less space too.

A brute force algorithm could first compute all anonymized graphs to compute from them

the reduced anonymized graphs. However, this will be very inefficient. Instead, we compute for each triple in the graph the CCP of each policy and then determine to which anonymized graphs it would belong. Indeed, the first step to compute these reduced anonymized graphs is to compute the set of allowed and forbidden matching permissions by using the algorithm **ComputeM**, in the same way it is done for our first enforcement algorithm. This process will be performed for both sets $\mathcal{A}$ and $\mathcal{F}$ composing each policy $\mathcal{P}$.

Given a graph $\mathcal{G}$ and the set of policies Γ, the next step is to determine which policies in Γ affect each one of the triples in $\mathcal{G}$. This can be done with algorithm 4, called **ComputePol**. It starts by initializing the function $\mathfrak{L}$, that will return an empty set for each triple in $\mathcal{G}$ and each b in Σ (line 1). Then, it will compute for each policy the functions $r_{\mathcal{A}}$ and $r_{\mathcal{F}}$, that will return the set of security patterns that are allowed and forbidden for each triple in $\mathcal{G}$ (line 3-4). Each triple that has at least one allowed matching permission gets its CCP computed using $r_{\mathcal{A}}$ and $r_{\mathcal{F}}$, in the same way as with our previous algorithm. However, instead of storing an anonymized triple, we add the policy affecting the triple to the set $\mathfrak{L}$ (lines 6-12).

Now that we have the set of policies affecting each triple, we can anonymize the graphs with algorithm 5, called **AnonymizeAll**. This algorithm first obtains the set of policies affecting each triple (line 1) and initialize the multisets where the reduced anonymized graphs will be stored (line 2). Then, it computes the anonymized triple for all triples in $\mathcal{G}$ and security patterns in Σ (lines 3-9), and adds them to the respective multiset of the reduced anonymized graph (line 10).

Now that we are able to determine the reduced anonymized graphs of a graph $\mathcal{G}$ with respect to a set of policies Γ, a user under a policy $\mathcal{P}$ can perform queries over $\mathcal{G}$. Again, we are interested in enforcing the policies for a user performing multiple queries. Algorithm 6, called **RoleEnforcement**, will be used to compute the answers for the user's queries. This algorithm starts by obtaining the reduced anonymized graphs and creating an empty graph to store the triples allowed for the user (lines 1-2). Next, it determines the multisets that correspond to the policy affecting the user and add their triples to graph $\mathcal{G}'$, replacing each λ by a fresh blank node (lines 3-6). Finally, it creates an empty list of URIs L, computes the answers to all the conjunctive queries in the list $\mathcal{CQ} = [\mathrm{CQ}_1, \ldots, \mathrm{CQ}_n]$ and adds the results to the list L (lines 7-10).

Algorithm 4 $\mathbf{ComputePol}\ (\mathcal{G}, \Gamma)$

Input: RDF Graph $\mathcal{G}$, set of policies Γ
Output: A function $\mathfrak{L}$ such that given a triple t and a security pattern b, $\mathfrak{L}(t, b)$ returns the set of policies affecting t for security pattern b.

$\mathfrak{L}(t, b) \leftarrow \varnothing$ for every $t \in \mathcal{G}$ and $b \in \Sigma$
for all $\mathcal{P} \in \Gamma$ **do**
  $r_{\mathcal{A}} \leftarrow \mathbf{ComputeM}(\mathcal{G}, \mathcal{A}_{\mathcal{P}})$
  $r_{\mathcal{F}} \leftarrow \mathbf{ComputeM}(\mathcal{G}, \mathcal{F}_{\mathcal{P}})$
  for all $t \in \mathcal{G}$ **do**
    if $r_{\mathcal{A}}(t) \neq \varnothing$ **then**
      $S_{\mathcal{A}} \leftarrow r_{\mathcal{A}}(t)^{\subseteq}$
      if $r_{\mathcal{F}}(t) \neq \varnothing$ **then**
        $S_{\mathcal{F}} \leftarrow r_{\mathcal{F}}(t)^{\supseteq}$
        $S_{\mathcal{A}} \leftarrow S_{\mathcal{A}} \setminus S_{\mathcal{F}}$
      for all $b \in S_{\mathcal{A}}^{\downarrow}$ **do**
        $\mathfrak{L}(t, b) \leftarrow \mathfrak{L}(t, b) \cup \{\mathcal{P}\}$
return $\mathfrak{L}$

Algorithm 5 $\mathbf{AnonymizeAll}\ (\mathcal{G}, \Gamma)$

Input: RDF Graph $\mathcal{G}$, set of policies Γ
Output: The set of reduced anonymized graphs

$\mathfrak{L} \leftarrow \mathbf{ComputePol}\ (\mathcal{G}, \Gamma)$
$\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma') \leftarrow$ empty multiset for every $\Gamma' \subseteq \Gamma$
for all $t \in \mathcal{G}$ **do**
  for all $b \in \Sigma$ **do**
    if $\mathfrak{L}(t, b) \neq \varnothing$ **then**
      $t' \leftarrow (\lambda, \lambda, \lambda)$
      if $\mathsf{s} \in b$ **then** $t'[\mathsf{s}] \leftarrow t[\mathsf{s}]$
      if $\mathsf{p} \in b$ **then** $t'[\mathsf{p}] \leftarrow t[\mathsf{p}]$
      if $\mathsf{o} \in b$ **then** $t'[\mathsf{o}] \leftarrow t[\mathsf{o}]$
      $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma') \leftarrow \mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma') \uplus \{t'\}$ where $\Gamma' = \mathfrak{L}(t, b)$
return $\{\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma') \mid \Gamma' \subseteq \Gamma\}$

7.2 Space Analysis

For a graph $\mathcal{G}$, which contains n triples, and a number r of role policies contained in a set Γ, the space that the reduced anonymized graphs will use is $5 \times |\mathcal{G}| = 5 \times n$. This is because each triple from graph $\mathcal{G}$ can be present in multiple $\mathsf{Anon}^{\downarrow\Gamma}(\mathcal{G}, \Gamma')$ at the same time, in all these forms: $(\mathsf{s}, \mathsf{p}, \mathsf{o})$, $(\mathsf{s}, \mathsf{p}, \lambda)$, $(\lambda, \mathsf{p}, \mathsf{o})$, $(\mathsf{s}, \lambda, \lambda)$ and $(\lambda, \lambda, \mathsf{o})$. So the maximum total space that will be used to store all the reduced anonymized graphs, in the worst case, is 5 times the size of the original graph. To this number, we have to add the size of the original

Algorithm 6 $\mathbf{RoleEnforcement}\ (\mathcal{G}, \Gamma, \mathcal{P}, \mathcal{CQ})$

Input: RDF Graph $\mathcal{G}$, set of policies Γ, the user's policy $\mathcal{P}$, with $\mathcal{P} \in \Gamma$, and a list of conjunctive queries $\mathcal{CQ} = [\text{CQ}_1, \ldots, \text{CQ}_n]$.
Output: A list of URIs L

```
G ← AnonymizeAll(G, Γ)
G' ← ∅
for all Anon^{↓Γ}(G, Γ') ∈ G do
    if P ∈ Γ' then
        G' ← G' ⊎ Anon^{↓Γ}(G, Γ')
G' ← G' with all ocurrences of λ replaced by a fresh blank node
L ← empty list
for all CQ ∈ CQ do
    R ← Ans(CQ, G')
    L ← Add R at the end of the list
return L
```

graph, which will be stored to recompute the reduced anonymized graphs in case that the original graph is modified or if the policies are modified. Last, we need twice the size of the original graph in order to temporarily store the graph that will be generated to answer user's queries, i.e., the total space that will be needed to manage an RDF database with this method is $O(5 \times |\mathcal{G}| + |\mathcal{G}| + 2 \times |\mathcal{G}|) = O(5 \times n + n + 2 \times n) = O(8n) = O(n)$.

In section 6.3 we showed that given a policy the space used by its anonymized graph is $O(2 \times n)$. Thus, for a policy with r roles, if we store each anonymized graph separately, we require $O(2 \times r \times n)$ space. Compared with the technique provided in this section, both algorithm will use a similar amount of space when the number of policies is small. However, as the number of policies increases, our new enforcement algorithm is considerably better at managing the space used.

7.3 Time Cost Analysis

We will study the cost of computing when a user, under a policy $\mathcal{P} = (\mathcal{A}, \mathcal{F})$ with $\mathcal{P} \in \Gamma$, wants to perform m queries over a graph $\mathcal{G}$ with n triples, where $|\mathcal{A}|$ is a, $|\mathcal{F}|$ is f and $|\Gamma|$ is r. Figure 7.1 contains a summary of the parameters used in this section. We are going to continue using simplifications like in the previous chapter, so we are going to consider that c^{Γ}_{max} and p^{Γ}_{max} are the maximum number of comparisons and patterns, respectively, of any conjunctive query CQ in $\mathcal{CQ}$ and any ACP in Γ.

As we already know the cost of performing any query on a graph and the cost of the algorithm **ComputeM**, in order to be able to obtain the cost of performing m queries, we need to determine the cost associated to the three algorithms presented in this chapter.

Variables	Meaning
m	number of conjunctive queries in $\mathcal{CQ}$
n	number of triples in a graph $\mathcal{G}$
r	number of policies in Γ
a_{max}	number of ACP in $\mathcal{A}$, for all $\mathcal{A}$ in Γ
f_{max}	number of ACP in $\mathcal{F}$, for all $\mathcal{F}$ in Γ
p^{Γ}_{max}	maximum number of patterns between the queries in $\mathcal{CQ}$ and the ACPs in Γ
c^{Γ}_{max}	maximum number of comparisons between the queries in $\mathcal{CQ}$ and the ACPs in Γ
$q^{\mathcal{G}}_{\text{CQ}}$	running time of **Ans** (CQ,$\mathcal{G}$)
$mp^{\mathcal{G}}_{\Pi}$	running time of **ComputeM** $(\mathcal{G}, \Pi)$
$cp^{\mathcal{G}}_{\Gamma}$	running time of **ComputePol** $(\mathcal{G}, \Gamma)$
$ha^{\mathcal{G}}_{\Gamma}$	running time of **AnonymizeAll** $(\mathcal{G}, \Gamma)$

Figure 7.1: Summary of parameters.

7.3.1 ComputePol $(\mathcal{G}, \Gamma)$

Line 1 of Algorithm 4 is $O(n)$ because it initializes the function for every triple in $\mathcal{G}$ and $|\Sigma| = 5$. The **for all** statement that follows will run r times. To simplify the calculation process, we will consider a_{max} as the maximum number of ACP in $\mathcal{A}$, for all $\mathcal{A}$ in Γ, and f_{max} will be the maximum number of ACP in $\mathcal{F}$, for all $\mathcal{F}$ in Γ. Now, Line 3 will run in $mp^{\mathcal{G}}_{\mathcal{A}_{\mathcal{P}}} = O(a_{max} \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}})$ and line 4 will run in $mp^{\mathcal{G}}_{\mathcal{F}_{\mathcal{P}}} = O(f_{max} \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}})$. The **for all** statement in line 5 will be executed n times. Lines 6 to 12 take a constant amount of time since $S_{\mathcal{A}}$ and $S_{\mathcal{F}}$ can contain at most five security sets and $S^{\downarrow}_{\mathcal{A}}$ contains at most two b. Therefore, the cost of running **ComputePol** $(\mathcal{G}, \Gamma)$ is $O(n + r \times (a_{max} \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}} + f_{max} \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}} + n)) = O(r \times (a_{max} + f_{max}) \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}})$. We will denote this running time by $cp^{\mathcal{G}}_{\Gamma}$.

7.3.2 AnonymizeAll $(\mathcal{G}, \Gamma)$

Line 1 runs in $cp_{\Gamma}^{\mathcal{G}} = O(r \times (a_{max} + f_{max}) \times c_{max}^{\Gamma} \times n^{p_{max}^{\Gamma}})$. Creating an empty multiset takes a constant amount of time, so line 2 runs in $O(2^r)$. The **for all** statement in line 3 will be executed n times. Lines 4 to 10 take a constant amount of time since Σ contains five b and creating a triple and replacing its subject, predicate and/or object with the values from a t in $\mathcal{G}$, and adding the new triple to its corresponding reduced graph are operations that take a constant amount of time. Therefore, the cost of running **AnonymizeAll** $(\mathcal{G}, \Gamma)$ is $O(r \times (a_{max} + f_{max}) \times c_{max}^{\Gamma} \times n^{p_{max}^{\Gamma}} + 2^r + n) = O(r \times (a_{max} + f_{max}) \times c_{max}^{\Gamma} \times n^{p_{max}^{\Gamma}} + 2^r)$. We will denote this running time by $ha_{\Gamma}^{\mathcal{G}}$.

7.3.3 RoleEnforcement $(\mathcal{G}, \Gamma, \mathcal{P}, \mathcal{CQ})$

The running time of algorithm **RoleEnforcement** depends on the cost of running algorithm **AnonymizeAll** $ha_{\Gamma}^{\mathcal{G}}$, the cost of creating the anonymized graph $\mathcal{G}'$ from the reduced anonymized graphs related to the policy $\mathcal{P}$, and the cost of performing $|\mathcal{CQ}| = m$ queries on the graph $\mathcal{G}'$.

The first part of the time cost analysis for this algorithm comprises lines 1 and 2. Line 1 is $ha_{\Gamma}^{\mathcal{G}} = O(r \times (a_{max} + f_{max}) \times c_{max}^{\Gamma} \times n^{p_{max}^{\Gamma}} + 2^r)$, since it computes the reduced anonymized graphs of $\mathcal{G}$, and line 2 takes a constant amount of time to create an empty graph.

The second part comprises lines 3 to 6. The **for all** statement in line 3 will be executed 2^r times. Line 4 takes a constant amount of time. Line 5 can run a maximum of $O(2 \times n)$ times, because the maximum number of triples that could be added to the anonymized graph is $2 \times n$. Finally, Line 6 runs in $O(2 \times n)$. Therefore, the process of creating the anonymized graph $\mathcal{G}'$ takes $O(2^r + 2 \times n + 2 \times n) = O(2^r + 4 \times n)$.

The last part comprises the cost of performing the queries. Line 7 takes a constant amount of time to create an empty list. The **for all** statement in line 8 will be executed m times. The running time of answering a query is $q_{\text{CQ}}^{\mathcal{G}'}$, and considering that, in the worst case, $|\mathcal{G}'| = 2 \times |\mathcal{G}|$ (see Section 6.3), we have $q_{\text{CQ}}^{\mathcal{G}'}$ equals to $O(c_{max}^{\Gamma} \times (2n)^{p_{max}^{\Gamma}})$ (line 9). In line 10, adding the answers to the list takes a constant amount of time. So, the cost of performing queries over an anonymized graph is $O(m \times c_{max}^{\Gamma} \times (2n)^{p_{max}^{\Gamma}})$

Thereby, the cost of running **RoleEnforcement** $(\mathcal{G}, \Gamma, \mathcal{P}, \mathcal{CQ})$ is $O(r \times (a_{max} + f_{max}) \times c^{\Gamma}_{max} \times n^{p^{\Gamma}_{max}} + 2^r) + O(2^r + 4 \times n) + O(m \times c^{\Gamma}_{max} \times (2n)^{p^{\Gamma}_{max}})$.

As we already know, anonymizing the graph is not an operation that will be performed each time a user needs to perform queries, but only when the original graph is modified. This is because the reduced anonymized graphs will be stored, allowing us to ignore that part of the cost of algorithm **RoleEnforcement**. In the end, the cost of enforcing the policy for a user under a certain role with this algorithm is $O(2^r + 4 \times n) + O(m \times c^{\Gamma}_{max} \times (2n)^{p^{\Gamma}_{max}})$ = $O(2^r + 4 \times n + m \times c^{\Gamma}_{max} \times (2n)^{p^{\Gamma}_{max}})$.

If we compare this result to the one obtained in section 6.2.4, $O(m \times c_{max} \times (2n)^{p_{max}})$, we can see that the method presented in this section takes more time to compute queries. This difference in cost is due to the process of creating the anonymized graph from the reduced anonymized graphs.

Deciding at glance which of the two methods for controlling the access to RDF graphs is the best choice will depend on the number of roles being enforced and the number of triples of the original graph. However, it is more common that the number of triples of a graph is far greater than the number of policies, making irrelevant the cost of creating the anonymized graph compared to the cost of answering queries. Considering this, and the fact that admins could add more policies in the future, implementing an enforcement algorithm for multiple roles is recommended over the algorithm described in chapter 6.

Chapter 8

Conclusion

Companies and institutions that want to publish data in the semantic web could need to put restrictions over the access to their data. Therefore, as RDF is the standard format to publish data in the semantic web, there is a need for access control mechanisms that provide a fine-grained access control for RDF graphs. Current work in RDF access control can enforce restrictions over data units as smallest as an RDF triple. However, with these approaches, we cannot grant nor deny the access to parts of the triples.

In this thesis, we presented the syntax and semantics of a fine-grained access control policy for RDF graphs. The policies defined with our model handle the access to the subject, predicate and object of a triple, filling the gap left by current approaches. These policies are based on sets of allowed and forbidden permissions that specify which triples are affected by the restriction. We presented an algorithm for the enforcement of the policies that is derived from the formal definition of the semantics, and studied its time cost and space cost. Considering that for this algorithm all access control policies are defined for a specific role, we handled the problem of dealing with multiple policies and their respective anonymized graphs by proposing an algorithm that improved the previous one in terms of the space it needs for storing the data. This means that given a set of roles, their respective access control policies and an RDF graph $\mathcal{G}$, it stores all the anonymized versions obtained from each policy applied to graph $\mathcal{G}$ without storing multiple copies of the triples that these graphs have in common. We analyzed its time cost to discover that it takes more time to compute queries than our first algorithm.

In the future, we will consider studying how RDFS affects our policy and enforcement methods, and propose alternatives to ensure that our policy and enforcement algorithms can handle RDFS properly. Another issue that can be addressed is controlling the updates that can be performed in an RDF graph, considering the SPARQL 1.1 update language [PGP13]. The last issue to consider for future work is the incremental maintenance of anonymized graphs, which means avoiding the need to compute a new anonymized graph

for each role when update operations, such as insert or delete a triple, are performed on the original graph. We dedicated part of our study to this last topic, however, the results we obtained with the algorithms we created were not useful. This is, in the worst case, they needed to recompute all anonymized graphs.

Bibliography

[ACH+07] Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2007.

[BCEPM08] Robert Bunge, Sam Chung, Barbara Endicott-Popovsky, and Don McLane. An operational framework for service oriented architecture network security. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 312–312. IEEE, 2008.

[BCF07] Loreto Bravo, James Cheney, and Irini Fundulaki. Repairing inconsistent XML write-access control policies. *CoRR*, abs/0708.2076, 2007.

[BCFM00] Elisa Bertino, Silvana Castano, Elena Ferrari, and Marco Mesiti. Specifying and enforcing access control policies for xml document sources. *World Wide Web*, 3(3):139–151, 2000.

[BL09] Tim Berners-Lee. *Linked Data*, 18 June, 2009. Web. Accessed 10 November, 2014. `http://www.w3.org/DesignIssues/LinkedData.html`.

[CFMS94] Silvana Castano, Maria Grazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1994.

[CVDG12] Luca Costabello, Serena Villata, Nicolas Delaforge, and Fabien Gandon. Ubiquitous access control for sparql endpoints: lessons learned and future challenges. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 487–488. ACM, 2012.

[DA06] Sebastian Dietzold and Sören Auer. Access control on rdf triple stores from a semantic wiki perspective. In *ESWC Workshop on Scripting for the Semantic Web*. Citeseer, 2006.

[DBM14] R.V. Guha Dan Brickley and Brian McBride. *RDF Schema 1.1*, 25 February, 2014. Web. Accessed 10 November, 2014, `http://www.w3.org/TR/rdf-schema/`.

[DuC11] Bob DuCharme. *Learning SPARQL*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2011.

[FFMA10] Giorgos Flouris, Irini Fundulaki, Maria Michou, and Grigoris Antoniou. Controlling access to RDF graphs. In *Proceedings of the Third future internet conference on Future internet*, FIS'10, pages 107–117, Berlin, Heidelberg, 2010. Springer-Verlag.

[FJK+08] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. Rowlbac: Representing role based access control in owl. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 73–82, New York, NY, USA, 2008. ACM.

[FKC07] David F. Ferraiolo, Richard D. Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control, Second Edition*. Artech House, Inc., Norwood, MA, USA, 2007.

[Gro14a] RDF Working Group. *RDF 1.1 Concepts and Abstract Syntax*, 25 February 2014. Web. Accessed 15 April, 2015, `http://www.w3.org/TR/rdf11-concepts/#section-generalized-rdf`.

[Gro14b] RDF Working Group. *Resource Description Framework (RDF)*, 25 February, 2014. Web. Accessed 10 November, 2014, `http://www.w3.org/RDF/`.

[JF06] Amit Jain and Csilla Farkas. Secure resource description framework: An access control model. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 121–129, New York, NY, USA, 2006. ACM.

[KJP08] Jaehoon Kim, Kangsoo Jung, and Seog Park. An introduction to authorization conflict problem in RDF access control. In Ignac Lovrek, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (2)*, volume 5178 of *Lecture Notes in Computer Science*, pages 583–592. Springer, 2008.

[PGP13] A. Passant P. Gearon and A. Polleres. *SPARQL 1.1 Update*, 21 March, 2013. Web. Accessed 10 November, 2014, `http://www.w3.org/TR/sparql11-update/`.

[PS99] Joon S Park and Ravi Sandhu. Rbac on the web by smart certificates. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 1–9. ACM, 1999.

[PS08] E. Prud'hommeaux and A. Seaborne. *SPARQL Query Language for RDF*, 15 January, 2008. Web. Accessed 10 November, 2014, `http://www.w3.org/TR/rdf-sparql-query/`.

[PSA01] Joon S Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security (TISSEC)*, 4(1):37–71, 2001.

[RFJ05] Pavan Reddivari, Tim Finin, and Anupam Joshi. Policy based access control for an RDF store. In Lalana Kagal, Tim Finin, and Jim Hendler, editors, *Policy Management for the Web*, pages 78–81, 2005.

[Sam02] Pierangela Samarati. Regulating access to web-published data. In *ERCIM News, 49*, page 10, 2002.

[SBJ96] Pierangela Samarati, Elisa Bertino, and Sushil Jajodia. An authorization model for a distributed hypertext system. *Knowledge and Data Engineering, IEEE Transactions on*, 8(4):555–562, 1996.

[SHP13] A. Seaborne S. Harris and E. Prud'hommeaux. *SPARQL 1.1 Query Language*, 21 March, 2013. Web. Accessed 10 November, 2014, `http://www.w3.org/TR/sparql11-query/#SparqlOps`.

[SS94] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, Sept 1994.

[wLO14] *W3C Linking Open Data Project*, 24 September, 2014. Web. Accessed 10 November, 2014, `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`.

[wNE12] *Nesstar Publisher*, 2012. Web. Accessed 10 November, 2014, `http://www.nesstar.com`.

[YT05] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.