

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Access Control in Linked Data Archives

Tiago Gonçalves da Silva



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Sérgio Nunes

March 30, 2023

Access Control in Linked Data Archives

Tiago Gonçalves da Silva

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Cristina Ribeiro

Main Examiner: Prof. José Carlos Ramalho

Supervisor: Prof. Sérgio Nunes

March 30, 2023

Abstract

The ever-growing amount of data that exists in today's age pushes for solutions that effectively give us access and let us interconnect information to gain knowledge about a given subject. It is within this context that, in the last decade, the development of systems supported on Linked Data and the values of the Semantic Web has gained some momentum. The knowledge graph is one example of a structure that follows these principles whose use has also experienced growth lately. Built of nodes, edges, and labels, a knowledge graph is a network that represents real-world entities, such as concepts, objects, events, or persons, and characterizes the relationships between them. DBpedia, Wikidata, and Google's knowledge graph are three examples of many instances of knowledge graph-based implementations. Like them, most use cases from until a few years ago usually fall into two categories. They either contain open data and are available to anyone or, instead, hold private information and limit the access through a high-level authentication and authorization method. More recently, the set of use cases for knowledge graphs has expanded. In some new use cases, there is the need for systems that hold both data intended to be shared across all users and sensitive data with access limitations. This means that, in such cases, users are only granted access to part of the stored data. The need for systems with such requirements has emerged in many contexts. In the archival sector, the field of focus of this work, the existence of public records that are free of access to anyone and records that remain confidential is relatively frequent. In cases like these, for systems to meet the mentioned access control needs, they must implement a strategy that permits the specification of permissions and subsequently uses them to restrict user access to the data it holds.

Access Control in applications of the archival domain, supported by linked data and knowledge graphs is the focus of this work. Here, we developed an access control system for the Spring Boot framework, that allows the definition of access policies to control the access and execution of update operations over the data stored in a knowledge graph. Our platform, AC4LDA, was designed to fulfill the access control needs of the archival domain, which were identified within the context of the EPISA project. To successfully achieve the requirements of the context of archives, we followed a strategy based on the construction of temporary graphs to enforce the specified access policies. The policies' representation, and evaluation and enforcement process were implemented using Semantic Web technologies in RDF and SPARQL, and rely on an RDF-S vocabulary that extends a previously published vocabulary. The development the AC4LDA also included the implementation of a web application to aid the specification of access policies. Finally, faced with the nonexistence of a standard for testing and validating access control mechanisms for linked data, we modeled a procedure to evaluate the correctness of the implemented strategy and a mechanism to evaluate the performance of the system using the tools provided by a SPARQL Performance Benchmark, SP2Bench. The results of these processes showed that, despite its effectiveness, the AC4LDA struggles with the increase of the size of the protected set of data.

Keywords: Access Control, Linked Data, Knowledge Graph

Resumo

A crescente quantidade de dados que existe nos dias de hoje leva a que sejam desenvolvidas soluções que nos permitam aceder e interligar informação de forma eficaz, para obter conhecimento acerca de um determinado assunto. É neste contexto que, na última década, o desenvolvimento de sistemas suportados em Dados Ligados e nos valores da Web Semântica tem ganho algum ímpeto. O grafo de conhecimento é um exemplo de uma estrutura que segue estes princípios, cuja a utilização também tem crescido. Construído através de nós, arestas e etiquetas, um grafo de conhecimento é uma rede que representa entidades do mundo real tais como conceitos, objetos, eventos, pessoas; e permite caracterizar as relações existentes entre eles. DBpedia, Wikidata e o grafo de conhecimento da Google são três exemplos entre inúmeros casos de implementação de grafos de conhecimento. Como estes, a maioria dos casos de uso podem colocar-se em duas categorias: ou contêm dados abertos e disponíveis para qualquer acesso; ou possuem dados privados cujo acesso é limitado através de um método de autenticação e autorização de alto nível. Mais recentemente, o conjunto de casos de uso de grafos de conhecimento expandiu-se. Para alguns novos casos há a necessidade da existência de sistemas capazes de guardar dados partilhados por todos os seus utilizadores e dados sensíveis com restrições de acesso. Nestes casos, é concedido aos utilizadores acesso a apenas uma parte da totalidade dos dados guardados. A necessidade de sistemas com tais requisitos tem emergido em diversos contextos. No setor arquivista, a área em este trabalho se foca, a coexistência de registos pública de acesso livre para qualquer pessoa e registos confidenciais é relativamente frequente. Então é necessário implementar uma estratégia que permite a especificação de permissões e as utilize para restringir o acesso aos dados que detém.

O controlo de acesso em aplicações do contexto arquivista, suportadas por dados ligados é o foco deste trabalho. Desenvolvemos um sistema de controlo de acesso para a framework Spring Boot, que permite a definição de políticas de acesso para controlar o acesso e a execução de operações de atualização sobre dados guardados num grafo de conhecimento. A nossa plataforma, AC4LDA, foi desenhada para cumprir as necessidades de controlo de acesso do domínio arquivista, colecionadas no contexto do projeto EPISA. Para atingir os requisitos do contexto arquivista, seguimos uma estratégia baseada na construção de grafos temporários para fazer cumprir as políticas de acesso especificadas. A representação destas restrições e os respetivos processos de avaliação e aplicação foram implementados usando tecnologias da Web Semântica como RDF e SPARQL, e dependem de um vocabulário RDF-S que expandimos. O desenvolvimento da AC4LDA incluiu também a implementação de uma aplicação web para auxiliar a especificação de políticas de acesso. Por fim, dada a inexistência de procedimentos padrão para testar e validar mecanismos de controlo de acesso para dados ligados, desenvolvemos um procedimento para avaliar as estratégias implementadas e um processo para avaliar o desempenho do sistema a partir das ferramentas fornecidas pelo SP2Bench, uma coleção de referência para avaliação de desempenho da linguagem SPARQL. Resultados mostram que, apesar de eficaz, a AC4LDA possui problemas relacionados com escalabilidade, uma vez que o aumento do volume de dados a

proteger provoca um aumento nos tempos de processamento.

Keywords: Controlo de Acesso, Dados Ligados, Grafo de Conhecimento

Acknowledgements

Thank you to my supervisor, Sérgio Nunes, for the patience and orientation, over the course of this work. The path was not always clear, and your guidance steered me to a successful outcome. I'm glad, and fortunate the EPISA project brought us together.

Also, a word of gratitude to the members and partners of the EPISA project who directly or indirectly participated in the development of this work.

Thank you to my family for the love and support you gave me. Even without realizing it, you all had a very important contribution to this work just by being yourselves and providing me with a sometimes much-needed distraction during difficult times.

Finally, thank you to Mariana for the constant and close support, encouragement, and motivational speeches. You sharing this path with me is something that I'll always cherish.

To all of you, I am endlessly grateful.

Tiago Silva

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Main Goals	3
1.4	Summary	4
2	Background on Linked Data and Access Control	5
2.1	The Semantic Web and Linked Data	5
2.1.1	Semantic Web Technologies	7
2.2	Access Control	14
2.2.1	Concepts and definition	14
2.2.2	Access Control in Software Systems	15
3	Access Control Over Linked Data	17
3.1	The Access Control Framework	18
3.1.1	Access Control Model	18
3.1.2	Policy Specification	20
3.1.3	Policy Enforcement	21
3.2	Existing Technologies and Solutions	22
3.3	Related Research Work	27
3.3.1	Methodology	27
3.3.2	Outline of Analyzed Research Works	29
3.3.3	Access Control for RDF and Linked Data Platforms	32
3.4	Discussion	42
4	Access Control Requirements for Linked Data and the Archival Context	46
4.1	Access Control Requirements for Linked Data	47
4.2	Access Control Requirements in the Context of Archives	48
4.2.1	The Archival Domain	48
5	Access Control for the Archival Domain	53
5.1	A Solution for the Archival Domain	53
5.1.1	Access Control Model	53
5.1.2	Policy Specification	54
5.1.3	Policy Enforcement	57
5.2	The AC4LDA Platform	63
5.2.1	The AC4LDA Package	64
5.2.2	The AC4LDA Admin Interface	73

6	Evaluation	82
6.1	Methodology	82
6.2	Results	84
6.2.1	Quality Level	84
6.2.2	Performance Level	89
6.3	Discussion	91
7	Conclusions	95
7.1	Conclusions and Main Contributions	95
7.2	Further Work	96
	References	98
A	Algorithms for Access Control Enforcement	108
A.1	Creation of temporary views	108
A.2	Enforcement of Access Control on Insert Operations	109
A.3	Enforcement of Access Control on Delete Operations	110
B	Docker Configuration	113
B.1	Docker	113
B.2	Apache Jena Fuseki	113
B.3	AC4LDA Admin	115
B.4	docker-compose.yml	116
C	AC4LDA REST API Reference	118
D	SHACL Constraints	129
E	Performance Test Results	134

List of Figures

2.1	The Semantic Web technology stack [1].	7
2.2	Example of an RDF graph [2].	8
2.3	Pattern Matching in a Sample RDF Graph [3].	10
2.4	Example of the result of inference based on symmetric, reverse, and transitive properties; in green is represented the information deduced.	15
3.1	RDF storage providers ranked by popularity according to solid IT.	22
3.2	Structure of RDF triplestore.	23
3.3	Mapping of designations of RDF triplestore providers.	24
4.1	Organization of groups and subgroups in the Archival domain.	49
4.2	Entities represented in the CIDOC-CRM ontology [4].	50
4.3	Informal example of the transformation of information from a hierarchical-based model to a graph-based model [5].	51
5.1	Summary of S4AC vocabulary [6, 7].	56
5.2	Summary of PRISSMA vocabulary [6, 8].	57
5.3	Addition of Role concept to PRISSMA vocabulary.	58
5.4	Addition of Access Policy Type concept to S4AC vocabulary.	59
5.5	Summary of the AD4LDA vocabulary.	61
5.6	Sample of RDF graph containing a request context.	62
5.7	Architecture of the AC4LDA Platform.	64
5.8	Architecture of the AC4LDA Package.	65
5.9	Sequence of operations in the AC4LDA Package.	67
5.10	Layout of the AC4LDA Admin GUI.	75
5.11	Sitemap of the AC4LDA Admin GUI.	76
5.12	Interface of the AC4LDA Admin GUI for viewing and deleting existing Roles.	77
5.13	Interface of the AC4LDA Admin GUI for editing existing Roles.	77
5.14	Interface of the AC4LDA Admin GUI for viewing and deleting existing Users.	77
5.15	Interface of the AC4LDA Admin GUI for creating Users.	77
5.16	Interface of the AC4LDA Admin GUI for viewing and deleting existing Access Policies.	81
5.17	Interface of the AC4LDA Admin GUI for creating Access Policies.	81
6.1	Association of Users and Roles for testing the evaluation of context.	85
6.2	Representation of entries of the DBLP database [9].	88
6.3	Projection and average response time by the number of policies in different dataset sizes.	92

6.4	Projection and average response time at different dataset sizes for a different number of access policies.	93
-----	--	----

List of Tables

2.1	Main constructs of RDF Schema [2].	9
2.2	Result of the execution of the query from Listing 2.3 against the graph of Figure 2.3.	11
2.3	Result of the execution of the query from Listing 2.4 against the graph of Figure 2.3.	11
2.4	Result of the execution of the query from Listing 2.5 against the graph of Figure 2.3.	12
3.1	Summary of access control features of RDF stores.	27
3.2	Relevant works in application Access Control by topic of focus.	29
3.3	Summary of features in works that implement access control for RDF and linked data systems.	44
3.3	Summary of features in works that implement access control for RDF and linked data systems (<i>cont.</i>).	45
5.1	Data properties added to the S4AC and PRISSMA vocabularies.	60
5.2	Restrictions applied to each of the AC4LDA vocabulary concept classes.	78
5.3	AC4LDA REST API Endpoints.	79
5.4	AC4LDA Admin GUI Routes.	80
6.1	Compliance with the Generic access control requirements.	84
6.2	Compliance with the archive-specific access control requirements.	84
6.3	Access Policies for testing Evaluation of Context.	86
6.4	Access Policies for testing Enforcement phase.	87
6.5	Operation executed for testing Enforcement phase.	88
6.6	Average execution times for tests (ms).	90
E.1	Execution times for tests (ms).	134

Listings

2.1	Example of a triple pattern.	10
2.2	Example of a set of triple patterns.	10
2.3	Sample SELECT operation.	11
2.4	Sample ASK operation.	11
2.5	Sample CONSTRUCT operation.	12
2.6	Sample INSERT operation.	12
2.7	Sample INSERT DATA operation.	12
2.8	Sample DELETE operation.	13
2.9	Sample DELETE DATA operation.	13
2.10	Example of definition of symetric, inverse, and transitive propoerties in OWL. . .	14
3.1	Structure and example of policy representation in a high-level syntax by Abel et al. [10].	33
3.2	Structure and example of policy language in Gabillon and Letouzey [11].	35
3.3	Notation and example of an annotated RDF triple.	36
3.4	Structure Quad tuple representing elements of an access policy [12].	37
3.5	Sample RDF triple representing a person's social security number and the en- crypted version of the same triple.	41
5.1	Sample ask context SPARQL queries.	58
5.2	Construction of a SPARQL SELECT query.	67
5.3	SPARQL SELECT query to retrive basic information about an access policy. . .	68
5.4	Definition of the methods for SPARQL operations provided the AC4LDA Java API.	70
5.5	Definition of the methods for authentication provided the AC4LDA Java API. . .	71
5.6	Maven dependency of the AC4LDA package.	72
5.7	Definition of AC4LDA Annotation.	72
5.8	Usage of AC4LDA Annotation.	73
5.9	Smaple configuration file for the AC4LDA Package.	73
6.1	Selection of a 1,000 triple graph using one and two SPARQL CONSTRUCT queries.	90
A.1	Algorithm for the creation of temporary views.	108
A.2	Algorithm for the enforcement of access control on insert operations.	109
A.3	Algorithm for the enforcement of access control on delete operations.	110
B.1	Dockerfile for the Apache Jena Fuseki	113
B.2	Dockerfile for the AC4LDA Admin GUI	115
B.3	docker-compose.yml file for starting Apache Jena Fuseki and the Admin GUI . .	116
C.1	OpenAPI specification for the AC4LDA REST API	118
D.1	Prefixes used in SHACL Constraints	129
D.2	Access Policy SHACL Constraints	129
D.3	Access Condition Set SHACL Constraints	130
D.4	Access Condition SHACL Constraints	130

D.5	User SHACL Constraints	131
D.6	Role SHACL Constraints	132
D.7	Environment SHACL Constraints	132
D.8	Context SHACL Constraints	133

Abbreviations

WWW	World Wide Web
URI	Uniform Resource Identifier
URI	Internationalized Resource Identifier
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
SPARQL	SPARQL Protocol and RDF Query Language
OWL	Web Ontology Language
AC	Access Control
ACL	Access Control List
ACF	Access Control Framework
ACM	Access Control Model
MAC	Mandatory Access Control
DAC	Discretionary Access Control
RBAC	Role-based Access Control
ABAC	Attribute-based Access Control
CBAC	Context-based Access Control
DBMS	Database Management System

Chapter 1

Introduction

1.1 Context

The World Wide Web (WWW) is one of the most important instruments available to today's society. Aside from playing a central role in people's lives; companies, businesses, and organizations heavily rely on it to run their operations. Nowadays, through the WWW, we can shop for anything from almost anywhere, consult our banking accounts, pay bills, access government tax platforms, get entertainment, interact in social networks, and access a vast space of shared information. Because of this network, we almost effortlessly have access to heaps of services and an unprecedented amount of information that continuously keeps on growing.

Nevertheless, according to Tim Berners-Lee, the inventor of the WWW, his creation has not yet reached its full potential [13] despite the already massive influence it currently holds. The WWW has gradually evolved and has suffered some changes since its origin in 1989. It started as a network of static pages connected through hyperlinks, later coined Web 1.0 [14]. As stated by Berners-Lee, the Web 1.0 was the “read-only” Web [15] as it only provided the transmission of information from its creators to its consumers. This first version of the WWW gradually progressed to the Web widely implemented today, the Web 2.0, a Web of services and of greater interactivity in comparison to its earlier version. This was the second generation of the Web, which transformed it into a “read-write” Web. As Choudhury [14] states:

“This was an era of user-generated content and huge social media interaction, blogging, video sharing, chatting, hosted services, web applications, voice over IP, emails, instant messages, social bookmarking, podcasting, picture sharing, weblogs (...) and all kinds of online interactivity became possible.”

These strides were vital to turning the WWW into the huge success it is today. However, as early as 2001, Berners-Lee proposed an extension to this version of the Web, a formalization of his original dream for the WWW, which, as he stated, consisted of a “universal space of information” where users should be able to make links to absolutely any piece of information [13]. With this in mind, Berners-Lee presented the concept known as “The Semantic Web”. The Semantic Web's goal is to give meaning to data in such a way that both humans and machines can understand and

explore. Ultimately, the Semantic Web is about turning the current document-oriented Web into a Web of Data utilizing already established technologies such as XML and the Resource Description Framework(RDF) [14, 16].

This version of the Web was later coined as Web 3.0, just as the concept of Linked Data was presented in 2006, again by Berners-Lee. Linked Data is one of the essential notions to put the Semantic Web into practice [17]. It defines a set of principles to guide the publication of structured data on the Web and linking that same data with additional data. Despite not being yet fully realized, the Web 3.0 principles have opened the door to a web of connected, shared, and unified information that promises to reduce the effort to search for data and improve the retrieved information's quality, quantity, and relevancy.

One particular field that can significantly benefit from the realization of the Semantic Web is the Archival domain. For years, Archives have played a significant role in our world. They usually comprise long-established institutions that hold extensive collections of historical records and represent the memory of nations, institutions, organizations, and much more. Archives are reliable primary sources of information and provide firsthand facts, data, and evidence from letters, reports, notes, memos, and photographs.

Nowadays, many archives are heavily represented in digital format [18]. With so much information, the archival domain benefits from implementing systems following the Semantic Web's values as the linkage between records and entities they mention ensures complementing with information that may not have. Generally speaking, linked data archives have the potential to build a broader knowledge network and provide access to a more significant set of facts compared to the existing archival systems by enriching archives with external information. At the same time, they contribute to the realization of the Web of Data.

1.2 Motivation

Over the last few years, the number of linked data datasets and linked data applications has grown significantly. Many of the published datasets belong to domains such as geography, government, life sciences, and linguistics, and thus following the Linked Data principles is becoming a common practice [19]. This growth results from the increasing interest in systems that both humans and machines can understand and explore, due to the advantages that such systems enable. Following the introduction of the Linked Data concept and its principles by Tim Berners-Lee, Linked Data became the basis to new Semantic Web concepts and structures like knowledge graphs, which various well-known systems and applications currently use [20].

In 2007, DBpedia was published and was one of the first systems based on this type of structure [21]. Similarly, Wikidata followed the same steps when it came out a few years later. These two systems are instances of the same use case and intend to achieve the same end: to publish and provide access to structured data about real-world entities. In 2012, Google presented its knowledge graph as a new component of its search engine [22]. Since then, Google's search engine has featured a section of information in its results page that uses a knowledge graph to enhance the

results obtained by the search performed. Contrarily to DBpedia and Wikidata, which give access to the data held by their knowledge graph, Google's tool uses its to map the search term with an entity of the knowledge graph and subsequently explore the connections and relations with other entities. Apart from these cases, other large companies like Amazon [23] and Microsoft [24] have also announced the usage of knowledge graphs in their systems.

Like in DBpedia, Wikidata, and Google's applications, the data in most linked data-based systems is equally available to any user. Regardless of any managing methods internal to the providers of knowledge graphs and linked data systems, from the user's perspective, in most implementations the data held by the system can be accessed in the same way by all users that is, there is no differentiation in the access to the data. More recently, however, linked data systems have started to evolve to more dynamic systems, something similar to what was witnessed in the transition from Web 1.0 to Web 2.0. While initially, the use cases for linked data provided access to static content like in the first version of the Web, lately have surged use cases that push for an infrastructure where users are provided with newer forms of interacting and generating new content. As Kirrane et al. [25] describes, a dynamic read/write infrastructure capable of supporting data integration of public and private data.

In Web 2.0, the possibility of having both private and public information plays a big part in the world's current dependence on the WWW, as many services would not be feasible without it. Now, the recently extended set of use cases for Web 3.0 systems pushes for solutions with similar characteristics. The implementation of dynamic linked data systems requires proper security mechanisms that control the user's access to information and enable systems in these environments to hold private data. In other words, systems must implement a strategy that permits the specification of permissions and subsequently uses them to restrict user access to the data it holds.

Particularly in the archival domain, access control mechanisms are crucial to the implementation of a linked data infrastructure as the collection of records that archives hold is usually only partially shared. In such cases, some documents are not available to the public user, and it could be that some may not even be available to some people from inside the institution. Furthermore, not all members within an Archive organization have permission to create or delete records or curate all the ones they can read. Hence, an access control module is a mandatory component for a linked data-based archive management system.

1.3 Main Goals

With the increase of interest in linked data systems and their evolution towards becoming a dynamic distributed data source, the set of use cases of such systems has expanded. With these new use cases, new requirements for linked data systems have surged. As discussed above, this is the case for the Archival domain, where implementing a linked data-based system requires methodologies that perform access control on the data the application holds. The results of this work aim to fulfill this necessity, so its main goals are the following:

1. Study existent solutions and technologies for performing access control on linked data systems
2. Design and implement a linked data access control solution suitable for the Archival Domain
3. Evaluate the implemented solution

1.4 Summary

This chapter covered the characterization of the context in which this work is inserted. We explored the Web's evolution from its origin to the present, the characteristics of each of its phases, and looked at the next steps. Additionally, we reflected on the archival domain, how the recent advancements to the Web are favorable in this case, and what benefits they would induce. Afterward, we inspected the problem the work aims to solve and the relevancy within its context. At last, given the problem to be tackled, we summarized the work's main goals and expected results.

The remainder of this work is divided into six additional chapters. In the next chapter, we present a set of concepts pertaining to the background of Linked Data and Access Control, which are crucial for the remainder of the work. Chapter 3, Access Control and Linked Data, which documents a study we performed of the state of the art, existing solutions, and technologies to deal with similar problems. Chapter 4 dedicated to the requirements for access control in linked data systems both at a generic level and in the context of Archives. Chapter 5 where we describe the implemented solution after considering the requirements elicited in the previous chapter and the solutions currently available analysed in Chapter 3. Afterward, in Chapter 6, the solution presented is validated and evaluated through different strategies at different levels. Finally, in Chapter 7, we draw conclusions regarding the accomplished work and define paths for future work.

Chapter 2

Background on Linked Data and Access Control

2.1 The Semantic Web and Linked Data

The Semantic Web is a concept proposed by the creator of the WWW, Tim Berners-Lee, and represents his vision for the future of the Web. The World Wide Web has undergone significant changes since it was first created in 1989. It started as a network of static pages connected through hyperlinks, and was referred to as the "read-only" Web because it only provided the transmission of information from its creators to its consumers. However, over time, the WWW evolved to become the dynamic and interactive Web that we know today. This second generation of the Web, unlike its initial version, which was primarily static, comprises an interactive space where users actively participate in the generation of content. Additionally, this second version of the Web or, as it was later coined, the Web 2.0, offers a platform where users can engage with each other and with content in various ways.

Despite these strides, Berners-Lee believed the potential still was not fully unlocked so, in 2001, he proposed an extension to this version of the Web, a formalization of his original dream for the WWW, which, as he stated, consisted of a "universal space of information" where users should be able to make links to absolutely any piece of information [13]. A proposal to extend the current "Web of Documents" with a "Web of Data" [26]. With this in mind, Berners-Lee presented the concept known as "The Semantic Web". The main objective of the Semantic Web is to represent information in such a way that it can be easily understood and explored not only by humans but also by machines.

As stated by Berners-Lee et al. [16]:

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in co-operation."

The Semantic Web aims build on top of the Web 2.0 to improve it, by transforming it into a Web of Data using a stack of existing technologies like the Resource Description Framework (RDF) [14, 16], RDF-S, OWL, and SPARQL [27], further described in Subsection 2.1.1.

Later, in 2006, Tim Berners-Lee presented another term: Linked Data [17]. Linked Data is a proceeding for the realization of the Semantic Web. The Semantic Web is not only about publishing data on the Web, but also about doing it in such a way that it is connected to data from other sources and thus builds a large network of information. Linked Data, together with the Semantic Web presents a set of design principles and a set of specific technologies that aim to make easier the access and management of data, by contributing to data description and data integration, which enables a web of connected, shared, and unified information that promises to reduce the effort to search for data and improve the retrieved information's quality, quantity, and relevancy.

Linked Data Principles

Along with the creation and definition of the term “Linked Data”, Tim Berners-Lee also proposed a set of four principles for the publication and usage of Linked Data [17]. Even though these principles are often called and considered rules, Berners-Lee states that not following them does not destroy anything. However, it incapacitates use cases to fully benefit from the advantages of the Semantic Web and Linked Data.

The four principles are as follows:

1. Use Uniform Resource Identifiers (URIs) as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. So that they can discover more things.

The first principle is one that is already followed by the standard technologies of the Semantic Web, and it is relative to using URIs to identify things. RDF specifies that resources should be identified by Internationalized Resource Identifiers (IRIs) and URIs are a subset of the former. The difference between them is that the former uses the Universal Coded Character Set while the latter is restricted to the usage of ASCII characters. Nevertheless, since some Web technologies may not be prepared to use the extended set of characters, the usage of URIs provides more guarantees in terms of support [28].

The second principle, while it reiterates the usage of URIs, it specifies that they should be identifiers that follow the HTTP protocol, one of the most widespread protocols in the WWW. This is recommended so that it is possible to look up a URI and retrieve information from the result of this process. The third principle builds on the indications of the second and points out

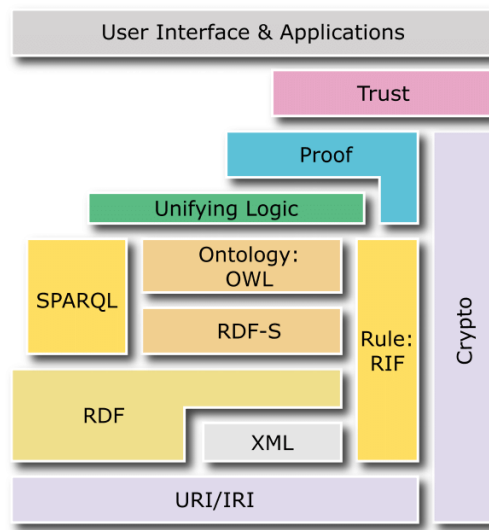


Figure 2.1: The Semantic Web technology stack [1].

that when HTTP URIs are effectively looked up, they should be resolved and retrieve data in a standard format, ideally a format part of the Semantic Web technology stack such as RDF, or in the form of a SPARQL endpoint. Finally, the fourth rule regards linking data to external sources. Similar to how it is done in the current “Web of documents”, where pages have links to other pages that have related information, in the Web of data this is also required. These links are essential as they are the ones that ultimately build the large network structure of information that enable the Semantic Web to be completely realized.

2.1.1 Semantic Web Technologies

The Semantic Web is a concept that relies on a set of standards and technologies, as depicted in Figure 2.1. At its core are RDF, the RDF Schema, SPARQL, and OWL, standards that, as we have just seen, are also essential in the context of the principles of Linked Data. In the rest of this section, we briefly describe their purpose and discuss aspects pertaining to each one of them, given their importance not only to the concepts of the Semantic Web and Linked Data but also to the development of this work.

2.1.1.1 The Resource Description Framework: RDF and RDF Schema

The Resource Description Framework, or RDF, is a framework designed for the purpose of representing interchangeable information about resources on the Web, in a universal way and understandable by both humans and machines. A resource is an element that denotes something in the world. Therefore, it can represent anything from physical things like objects, places, or a person to abstract and conceptual objects like numbers or strings. The RDF’s basic building blocks are RDF statements called triples that describe the relationship between two of these resources. A triple is composed of three entities, a subject, a predicate, and an object. The subject is a resource that

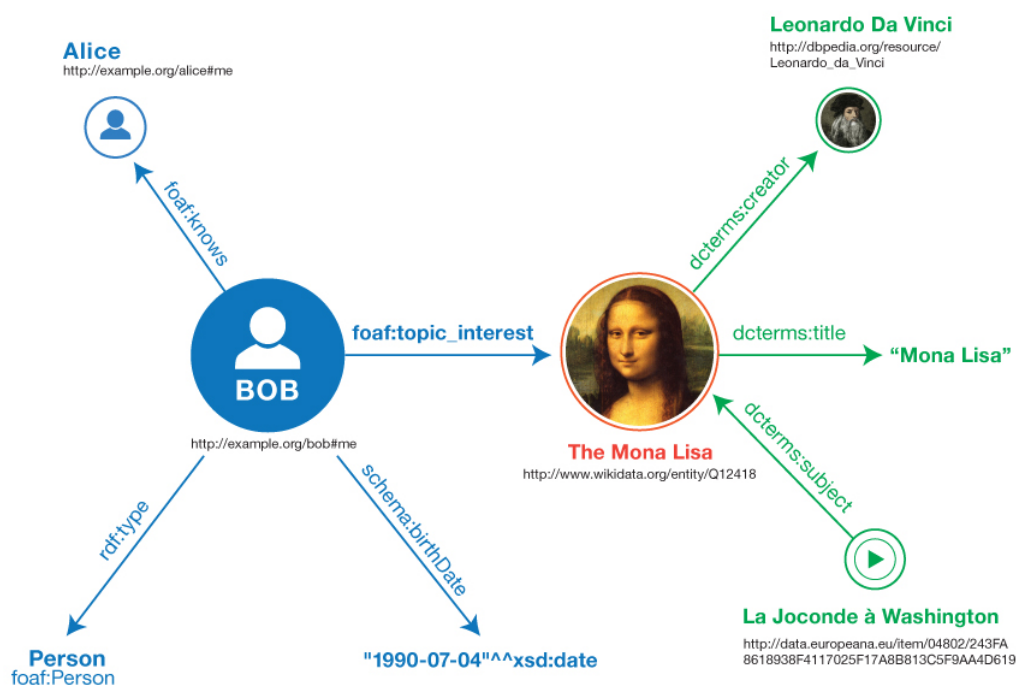


Figure 2.2: Example of an RDF graph [2].

generally is represented by an IRI, and the object is a resource that can either be an IRI or a literal, used for representing values such as strings, numbers, and dates. The predicate, also represented by an IRI, describes the directed and binary relationship between the triple’s subject and object.

Due to this structure, in its essence, a triple is parallel to a directed graph of two linked nodes, which means that a set of triples constitutes a larger graph. Such graphs are named RDF Graphs and can be visualized as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link [29]. Being a powerful and expressive framework for representing data, RDF is used for building knowledge graphs – richly interlinked, interoperable, and flexible information structures.

RDF Schema (RDF-S), also a standard of the W3C, is an extension to RDF that enables the description of meta-data about resources and properties, providing the framework with additional structure and data modeling characteristics. As we described above, RDF provides a way to make statements but does not make any assumptions. For that purpose, there are RDF vocabularies, a collection of IRIs to use in RDF graphs to enrich resources with semantic and contextual information. RDF Schema is one of such vocabularies and allows the definition of concepts like classes, subclasses, properties, sub-properties, hierarchies of these elements, and domain and range restrictions for properties. Additionally, it offers an easy path for the creation of other vocabularies through the usage and extension of its properties [2, 30]. In Figure 2.2, we can see the usage of the RDF-S property “type” along with FOAF [31], a famous vocabulary that derives from the specification of RDF-S.

In Table 2.1, we present a summary of the main modeling constructs provided by RDF Schema.

Construct	Syntactic form	Description
Resource (a class)	R rdf:type rdfs:Resource	R (a resource) is an RDF Resource. All other classes are subclasses of this class
Class (a class)	C rdf:type rdfs:Class	C (a resource) is an RDF class
Property (a class)	P rdf:type rdf:Property	P (a resource) is an RDF property
type (a property)	I rdf:type C	I (a resource) is an instance of C (a class)
subClassOf (a property)	C1 rdfs:subClassOf C2	C1 (a class) is a subclass of C2 (a class)
subPropertyOf (a property)	P1 rdfs:subPropertyOf P2	P1 (a property) is a sub-property of P2 (a property)
domain (a property)	P rdfs:domain C	domain of P (a property) is C (a class)
range (a property)	P rdfs:range C	range of P (a property) is C (a class)

Table 2.1: Main constructs of RDF Schema [2].

2.1.1.2 SPARQL

The SPARQL Protocol and RDF Query Language or, as it is more commonly known, SPARQL is a language for querying and updating RDF data sources [32]. After the emergence of the Resource Description Framework, there was the need for a specification for retrieving information from collections of data represented in such a way. With this, among various research works in the Semantic Web and Linked Data communities, querying languages such as SeRQL and RQL were proposed, which aimed to provide a syntax for performing read operations over RDF sources [33].

SPARQL was introduced in 2007 as an updated version of a previously developed RDF query language, RQL, and was considered a W3C Candidate Recommendation to ultimately become a Recommendation in the following year. Since then, SPARQL has become among the community the standard language for querying triple stores, and the number of existing query languages for RDF has settled. In 2013, SPARQL was extended with update operations that allow the manipulation of RDF data in a language named SPARQL Protocol and RDF Update Languages (SPARUL) [34]. The conjunction of both the query and update languages and the set of other features introduced was named SPARQL 1.1, nevertheless, like its predecessor, the language is often referred to as simply SPARQL. The SPARQL language, like most existing query languages for RDF, uses graph pattern matching, formed by variables, URIs, or RDF Literals, for exploring RDF graphs. There are two kinds of patterns available, triple patterns and graph patterns. The simplest kind of pattern available is the triple pattern, presented in listings 2.1 and 2.2, which contain

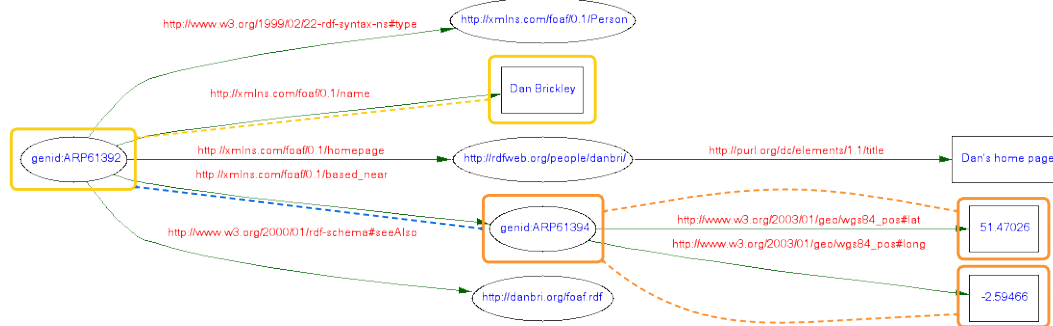


Figure 2.3: Pattern Matching in a Sample RDF Graph [3].

excerpts of SPARQL. Listing 2.1 presents a pattern composed of a variable, a URI, and an RDF literal and, in the example presented in Figure 2.3, matches with the data highlighted in yellow. Listing 2.2 presents three different patterns, the first, identical to the one in Listing 2.1, a second, which uses the variable in the previous pattern to match the information linked by the predicate `<http://xmlns.com/foaf/0.1/based_near>`, and a final one that uses the matched information in the variable “?location”, and ultimately matches the information highlighted in orange in Figure 2.3.

```

1 ...
2 ?person <http://xmlns.com/foaf/0.1/name> "Dan Brickley"
3 ...

```

Listing 2.1: Example of a triple pattern.

```

1 ...
2 ?person <http://xmlns.com/foaf/0.1/name> "Dan Brickley" .
3 ?person <http://xmlns.com/foaf/0.1/based_near> ?location .
4 ?location ?predicate ?value
5 ...

```

Listing 2.2: Example of a set of triple patterns.

SPARQL has a considerable set of types of query and update forms; for querying SPARQL provides SELECT, CONSTRUCT, ASK and DESCRIBE queries; to manipulate information, SPARUL provides operations for inserting and deleting data from RDF graphs with INSERT, DELETE, INSERT DATA, DELETE DATA operations and, additionally, LOAD and CLEAR. Aside from this, SPARUL also specifies operations that can be used for graph management using CREATE, DROP, COPY, MOVE, and ADD. Below, we present a brief description of the most used types of operations throughout this work.

SELECT

The SELECT operation, similarly to other querying languages both from the inside and outside of the Linked Data domain, is used to read information from triplestores. Like in other operations of the SPARQL language, it is possible to add a clause that defines the graph against which the query should be executed. Whereas if it is not set, the query runs against the given dataset's default graph.

```

1 SELECT ?person
2 WHERE {
3     ?person <http://xmlns.com/foaf/0.1/name> "Dan Brickley"
4 }

```

Listing 2.3: Sample SELECT operation.

?person
genid:ARP61392

Table 2.2: Result of the execution of the query from Listing 2.3 against the graph of Figure 2.3.

ASK

The ASK operation is a read-only operation that, instead of fetching and retrieving data like SELECT, simply uses the patterns in its query to assess whether they do or do not exist in the data source being queried. Thus, the result of an ASK operation is a boolean value that indicates if the pattern is present (true) or not present (false).

```

1 ASK WHERE {
2     ?person <http://xmlns.com/foaf/0.1/name> "Dan Brickley"
3 }

```

Listing 2.4: Sample ASK operation.

True

Table 2.3: Result of the execution of the query from Listing 2.4 against the graph of Figure 2.3.

CONSTRUCT

The CONSTRUCT operation is another type of read operation that, similarly to the operations viewed so far, uses pattern matching. However, while the SELECT operation returns the resulting values of the pattern-matching process associated with the variables listed in the first part of

the request, the CONSTRUCT operation returns a model containing the triples matched in the same process. By default, if no clause is provided, the operation returns all of the triples that were matched. It is also possible to restrict the patterns used to build the model through a clause containing triple patterns, as is presented next.

```

1 CONSTRUCT WHERE {
2   ?person <http://xmlns.com/foaf/0.1/name> "Dan Brickley"
3 }

```

Listing 2.5: Sample CONSTRUCT operation.

?s	?p	?o
genid:ARP61392	<http://xmlns.com/foaf/0.1/name>	"Dan Brickley"

Table 2.4: Result of the execution of the query from Listing 2.5 against the graph of Figure 2.3.

INSERT / INSERT DATA

The INSERT and INSERT DATA operations are operations that allow the creation of new triples in a triple store. While the two operations have the same purpose, they have slightly different structures and functioning. The INSERT operation allows the specification of triple and graph patterns in a while clause, similarly to the operations covered so far. On the contrary, the INSERT DATA has only one clause, that can be used only to specify triples which means that, in this case, variables are not available. As an example, in listings 2.6 and 2.7, we present two operations that produce the same result.

```

1 INSERT {
2   ?res <http://example.com/refers_to> genid:ARP61392
3 } WHERE {
4   ?res ?p ?o .
5   FILTER (?res = genid:ARP61392 || http://rdfweb.org/people/danbri/)
6 }

```

Listing 2.6: Sample INSERT operation.

```

1 INSERT DATA {
2   genid:ARP61392 <http://example.com/refers_to> genid:ARP61392 .
3   <http://rdfweb.org/people/danbri/> <http://example.com/refers_to> genid:
4     ARP61392
5 }

```

Listing 2.7: Sample INSERT DATA operation.

An important feature of “insert” operations is that they are, by nature, idempotent operations, meaning that the execution of two consecutive operations that add exactly equal triples or sets of triples will result in just one instance of that triple or set of triples in the destination RDF graph. If we execute consecutively the operations presented in listings 2.6 and 2.7, the RDF Graph from Figure 2.3 would have the same content as if we executed only one of them.

DELETE / DELETE DATA

While the previous set of operations deals with the addition of triples, the DELETE and DELETE DATA operations do the exact opposite: they remove from a triple store the set of triples that they specify. Additionally, the two operations have the same differences as the INSERT and INSERT DATA operations. With the DELETE operation, it is possible to use variables and pattern matching something that does not happen with the DELETE DATA operation. Below, in listings 2.8 and 2.9, we present two operations that have the same effect, if they were executed against the RDF Graph from Figure 2.3.

```

1 DELETE {
2   ?near ?p ?o
3 } WHERE {
4   genid:ARP61392 <http://xmlns.com/foaf/0.1/based_near> ?near
5 }
```

Listing 2.8: Sample DELETE operation.

```

1 DELETE DATA {
2   genid:ARP61392 <"http://www.w3.org/2003/01/geo/wgs84_pos#lat"> "51.47026"
3   genid:ARP61392 <"http://www.w3.org/2003/01/geo/wgs84_pos#long"> "-2.59466"
4 }
```

Listing 2.9: Sample DELETE DATA operation.

2.1.1.3 OWL

The Web Ontology Language (OWL), formally OWL 2, is a language designed by the W3C Working Group which extends the specification of an initial version of the same language [35]. Similarly to SPARQL, OWL is a W3C Recommendation and is part of W3C’s Semantic Web technology stack. The OWL language was designed to represent semantically rich knowledge about any given domain by modeling concepts and relations between concepts. Its fine-grained representation of the domain produces a network of meta-information about the data individuals that can be understood by machines, making possible the inference of knowledge and deduction of new information about the said individuals. This can be accomplished by OWL semantic reasoners, which are part of most of nowadays RDF store services and applications. OWL is serializable to a number of

RDF syntaxes, allowing its specification to be stored in triplestores. Like the RDF Schema, OWL permits the definition of classes and subclasses of concepts and the relations between them but goes further than that and allows the definition of meta-information regarding these entities [36].

Among the various features that OWL provides on top of RDF Schema are features like, for example, the definition of transitive, inverse, or symmetric characteristics of properties, which are particularly interesting for the matter of inference. The definition of these aspects provides the reasoner with information that lets it deduce implicit information based on explicit statements. For example, considering the examples presented in the language documentation [37, 38], the definition of these characteristics in properties that describe genealogy relations, the subsequent addition of a set of triples, and execution of a reasoner that explores these features results in the inference that can be seen in Figure 2.4. The definition of these features is presented in Listing 2.10.

```

1 <owl:SymmetricProperty rdf:about="hasSpouse"/>
2
3 <owl:ObjectProperty rdf:about="hasParent">
4   <owl:inverseOf rdf:resource="hasChild"/>
5 </owl:ObjectProperty>
6
7 <owl:TransitiveProperty rdf:about="hasAncestor"/>

```

Listing 2.10: Example of definition of symmetric, inverse, and transitive propoerties in OWL.

2.2 Access Control

2.2.1 Concepts and definition

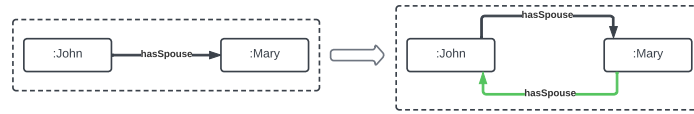
Access Control (AC) is present in all kinds of computer systems and applications. It is what lets us use computer systems for non-public purposes and is a mandatory condition for computer systems to be present in every sector, both in public and confidential scenarios. As Samarati and di Vimercati [39] state,

“Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied.”

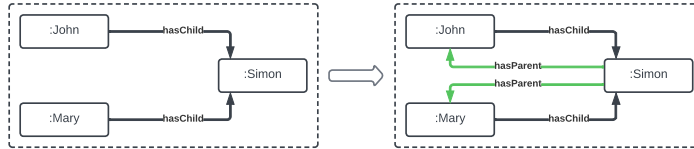
The OWASP Foundation¹ defines it as the same as the concept of “Authorization” and claims that it is responsible for two aspects, listed next [40]. The OWASP defines subjects as either users, devices, or processes and objects as the resources being protected.

1. “to govern the decisions and processes of determining, documenting and managing the subjects that should be granted access and the objects to which they should be granted access”;

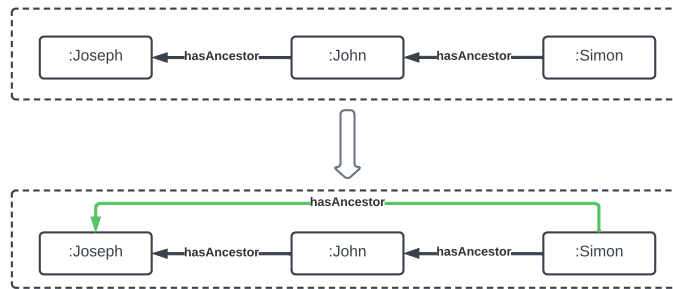
¹<https://owasp.org/>



(a) Inference based on a symmetric property.



(b) Inference based on a reverse property.



(c) Inference based on a transitive property.

Figure 2.4: Example of the result of inference based on symmetric, reverse, and transitive properties; in green is represented the information deduced.

2. “to govern the methods and conditions of enforcement by which subjects are allowed to or restricted from connecting with, viewing, consuming, entering into or making use of identified information objects”

Therefore, authorization is the stage whose process determines whether the access asked in the request should be granted or denied based on the data it tries to access, the information about the requester, and, sometimes, the information about the request itself. Given that, for the process of authorization to work, it is required to have precise knowledge about the subject making the request, most of the time, an authentication phase precedes the authorization phase. The authentication phase is responsible for identifying the subjects and providing that information to subsequent tasks that decide whether to grant or revoke a determined request from that subject.

2.2.2 Access Control in Software Systems

Access Control has a tremendous impact on the functioning of today’s software systems. In all kinds of systems, finding a barrier between users and the system’s data or functionalities is frequent. Particularly on the Web, the role of access control is significant since that applications are

accessible from almost anywhere in the world, as long as there is a device with an internet connection. So, given that systems that run on the internet do not require a physical presence to be accessed, their access control mechanisms have to be exceptionally competent.

There are many different scenarios and shapes for the specification of access control. In some cases, such as banking or enterprise systems, it is a central authority that regulates the authorization process. In the banking sector, access to a client's bank account is usually restricted to only the owner of that account. In enterprises and organizations, software systems are usually closed to the public but available for their personnel, even though sometimes there are cases with a mixture of both. Still, within the organization, different levels of access may exist to limit or permit access to resources based on the role or function of a given employee. In other cases, it is the users of the system that set access control definitions. In the case of social networks, it is frequent to see that a user, the owner of his profile, is able to determine whether that profile is public, visible to anyone that can navigate on the Web, to other users, owners of a profile in that social network, or even make their profile visible to a small subset of these users. While these are all common cases from use cases of the current Web, the Web 3.0 also has use cases of systems with access control mechanisms. Wikidata² and Geonames³, for example, are accessible by everyone on the Web, but the rights to contribute and edit the information there present are reserved for authenticated users.

Through these examples, we can see that the influence of Access Control in the current and in the next iteration of the Web is evident. The AC mechanisms provided are critical to the proper functioning of each example described. Through these examples, we can perceive that not only AC is essential in today's software systems, but it will remain that way in the future. As we have already discussed in the previous chapter, as more use cases for developing linked data applications show up, the implementation of access control mechanisms will continue to be necessary. Given that the implementations of new use cases of Linked Data share the same access restrictions as some of the examples seen, the need for access control in this context is clear.

²Wikidata, <https://www.wikidata.org>

³Geonames, <https://www.geonames.org/>

Chapter 3

Access Control Over Linked Data

In the last few years, with the growing interest in the Linked Data field and the increasing number of use cases supported by this concept, the need for access control in linked data systems has become more prominent. Despite some efforts by the community to develop standards, for now, there is still no obvious complete solution for performing access control in linked data systems. Some steps have been taken in that direction with the likes of, among others, the eXtensible Access Control Markup Language (XACML) [41], the ODRL [42] and the WebAccessControl (WAC) [43], three communities supported by OASIS ¹ for the former and W3C ² for the later two, that focus on the expression of access control policies [25]. Additionally, the appearance of new use cases in the context of Linked Data and the Semantic Web with the need for access control mechanisms has prompted the W3C to form a Working Group [44] to discuss and present a recommendation for aspects of such nature, but it has not been established a standard and, in terms of technical implementation, there is no software capable of answering to all needs.

Our goal was to assess the state of the art of access control for linked data, by exploring the components that are part of access control systems, particularly in the context of linked data, discovering the approaches followed in the available technologies and solutions, and appraising how the completed research work extends or improves these approaches. Therefore, this chapter details an analysis of what composes an access control system for linked data, and reviews the existing available solutions and the related work already developed in the field. Thus, it is composed of four sections. In the first, we describe the main components of an access control solution, and their respective characteristics. The second and third provide a review of the access control mechanisms present in existing technologies and in the literature. And, finally, the last section comprises an outline of our findings.

¹OASIS, <https://www.oasis-open.org/>

²W3C, <https://www.w3.org/>

3.1 The Access Control Framework

An access control system is mainly composed of the three aspects presented below. The composition of these three units forms, as Kirrane et al. [25] define, the Access Control Framework.

Access Control Model Defines the structure of the access control process. In other words, how the access control rights are mapped from the subjects to the objects being secured.

Policy Specification Defines how the policies and rules for performing access control are specified, namely the language in which they are described.

Policy Enforcement Defines how the access control policies and rules are enforced to ensure that the access to objects is protected.

The purpose of this section is to present some alternatives for each of these parts as it is where the existing solutions usually deviate from each other.

3.1.1 Access Control Model

The Access Control Model (ACM) comprises a formal representation of the structure and behaviour that a given access control system must follow [39]. Here, we briefly present the ones that have been used to model access control in the context of linked data and RDF data [25]. Aside from models like the Mandatory Access Control, Discretionary Access Control, Role Based Access Control, and View Based Access Control; that are established and have been used in multiple contexts, we present more recent approaches, like the Attribute Based Access Control and Context Based Access Control models.

Mandatory Access Control

The functioning of Mandatory Access Control (MAC) lies in the definition of an ordered set of security levels by a central authority. Additionally, all subjects and objects held by the system being secured must be assigned with a security level. Then, the order infused in the definition of the level will be used for deciding whether to grant access to a determined object by a subject. For subjects, the security level defines that it can access objects assigned with the same or a lower level. In contrast, for objects, the security level indicates that the object can be accessed by subjects assigned with the same or a higher level [45]. MAC was originally developed for military applications, and therefore, it is best suited to closed environments, where there is a clear, unbreakable, and obligatory hierarchy [46].

Discretionary Access Control

Despite relying on a central authority like MAC, Discretionary Access Control (DAC) revolves around the concept of ownership and the process of delegation. In DAC, the central authority assigns an owner and possibly additional subjects to resources. This results in each subject having

sets of access rights relative to a group of resources. However, these policies can be overridden by subjects who may pass their sets of access rights on to other subjects [25, 47]. DAC is known for being the Access Control Model used by the file systems of most computer operating systems.

Role Based Access Control

Role Based Access Control (RBAC) aggregates into groups subjects that should have access to exactly the same set of objects. The groups of subjects represent roles, a specific function within an organization, that are then granted access to resources through the definition of policies [46, 48]. RBAC is mainly used when there are multiple subjects with the same access rights to the same objects. In these cases, the effort required to define access control policies to all of the systems' subjects is lower when compared with previously discussed Access Control Models in MAC and DAC. Additionally, RBAC may also help in cases where the hierarchy is less strict, and the access to resources cannot be modeled into an ordered set of levels as MAC requires [25].

View Based Access Control

View Based Access Control (VBAC) is used in relational databases to simultaneously grant access to one or more relations, tuples, attributes, or values. Nonetheless, some approaches in the literature detail its use for RDF, making it a prospect for a linked data system [11, 49]. VBAC needs the definition of views through policies that will define sections or subsets of objects from within the system being secured. Ultimately, each subject is assigned to a view or a group of policies that delineates one and then is able to access its entire content.

Attribute Based Access Control

Attribute Based Access Control (ABAC) is an access control model that decides whether to grant or deny access to resources based on attributes of the subject requesting access and the object to be accessed. Instead of evaluating the subject and object as whole individual entities like in the previously seen access control models, ABAC uses subject and object attributes that enable a meticulous access control process. By taking into consideration multiple aspects defined by the various attributes it examines and through the large set of combinations of such attributes, it allows the definition of rules with a high level of precision [25, 50].

Context Based Access Control

Like ABAC, Context Based Access Control (CBAC) also diverges from evaluating the subject and object as unitary items in access control models like MAC or DAC. CBAC uses properties of the subject, object, and the environment to grant or deny access to resources [25]. These properties are relative to the request's context which Abowd et al. [51] define as "information that can be used to characterize the situation of an entity" and define aspects like the time, location or any kind of data regarding the conditions from which the request is made.

3.1.2 Policy Specification

The representation of policies is a critical aspect of an access control system as it defines the syntax and semantics of the configuration of rules. Ultimately, the policy representation exhaustively expresses what users have access to a given object and, consequently, the objects a user has access to. The language of representation and description of these policies can be anything provided that there is a strategy for subsequently analyzing and applying the access constraints specified in the policies to the secured data. Nonetheless, the combination of these two components can limit the implementation of the desired Access Control Model or the inclusion of characteristics in the resulting system. Below are enumerated languages used in the literature for policy specification. Listed are both generic languages whose primary purpose is not the representation access control policies and languages specially designed for this task.

RDF-S and Ontology languages (DAML, DAML+OIL, OWL) RDF-S and existing ontology languages have been a recurrent choice for expressing access control policies. These languages have been used to either express policies directly or used as a base for designing new policy languages. Among these languages are DARPA Agent Markup Language³ (DAML), DAML+OIL [52] and the Web Ontology Language (OWL) [35]. These languages were designed for the Semantic Web and enable the definition of knowledge about entities and the relationship between them, one of the main aspects of the Semantic Web concept.

Semantic Web Rule Language (SWRL) The Semantic Web Rule Language (SWRL) [53] is a language that conjugates the properties of two sublanguages OWL with RuleML and provides a notation to representing implication rules aiming to bring description logic to the context of the Semantic Web. These rules are composed of two parts, the antecedent and the consequent, which comprise conditions. Each rule behaves as a logic implication meaning that, if the conditions expressed in the antecedent are verified, then the conditions presented in the consequent will also be verified. Because it is based on OWL, SWRL rules can be expressed entirely in an RDF graph, additionally rules can be evaluated by OWL reasoners, present in most RDF store services.

XACML XACML or, the eXtensible Access Control Markup Language, is a generic access control language that combines the specification of two XML-based sublanguages for regulating access to the assets of a resource server [41]. The languages it describes are a policy language that allows the definition of access policies and conditions to access a specific resource, and a request/response language that describes a structure for queries to be executed against the restrictions described with the policy language in order to formulate a decision of granting or revoking the access to the requested resource [54].

XACML is an OASIS standard and has been included in standards from other groups, whereas in the Linked Data field, it has been extended to cater to specific use cases and used in the implementation of access control practices for linked data systems [55].

³DAML, <http://www.daml.org/index.html>

Other representations Aside from the languages already mentioned, there are also specific languages developed solely for the purpose of representing Access Restrictions. Cases like the language presented in Stojanov et al. [56], inspired by the syntax of SPARQL, comprise languages that, similarly to XACML, are specialized for the definition of Access Policies. The application of this strategy brings a language that caters to a specific domain and constitutes a personalized language for a given context.

3.1.3 Policy Enforcement

Policy Enforcement is the process responsible for ensuring that the policies specified in a determined policy language in the configuration of a given system are, in fact, realized. This process is responsible for providing a decision of either allowing or denying the execution of an action or a set of actions present in the request by evaluating the details of the aforementioned request. This procedure can be decomposed into two steps, the first being the determination of the group of subjects an access policy refers, or the set of policies with which the requesting subject is matched, and the second which consists of the calculation of the allowed actions in that requesting context, using the set of policies obtained in the previous phase. Nevertheless, sometimes the stages are tightly coupled and are inseparable. For example, when the policies are specified through the use of Access Control Lists and the units of data to be protected are at the level of the graph, the policy enforcement comprises only the interpretation of the ACLs, and further execution of the action requested only in the allowed graphs. If the units being secured are already partitioned in the smallest form for which is desired, the definition of intricate Access Policies (i.e. Control access to RDF graphs, datasets, and servers) and the calculation of allowed data may not be needed.

With this, we can see that the process's intricacy increases with the rise of the complexity of both the Access Control Model being implemented and the strategy used for the specification of policies. Aside from this, the way how the secured data is arranged, and the granularity of the desired Access Policies also affect the complexity of this process. If the data is partitioned across different graphs or datasets and there is a scenario where it is desired to implement access protection at one of these levels, then the data is already segregated with the required granularity. However, if the intended protection is at a more fine-grained level, more specifically, allowing access to a set of triples of an RDF graph, some additional operations may be required, given that the data allowed to be accessed is mixed with data that should not be accessible. For scenarios like this, there are two main lines of development: either the context where the request is executed is changed, and the request is performed in an environment containing only allowed data, or the request itself is modified in such a way that only allowed data is retrieved.

The enforcement of access control can be performed in multiple ways that constitute either pre-filtering the data, prior to the execution of requests or filtering the request itself, transforming it so that it accesses only allowed data. One would say that another possible approach would be to post-filter the retrieved data and remove the disallowed data; however, this is not attainable because after the data is retrieved, it loses its structure defined by a vocabulary or ontology, and that is important information, crucial for deciding whether that data should be accessible or not.

Rank			DBMS	Database Model	Score		
Dec 2022	Nov 2022	Dec 2021			Dec 2022	Nov 2022	Dec 2021
1.	1.	1.	MarkLogic	Multi-model 	8.82	-0.93	-0.12
2.	2.	2.	Virtuoso 	Multi-model 	5.95	+0.11	+0.88
3.	3.	 4.	Apache Jena - TDB	RDF	4.27	+0.64	+1.56
4.	4.	 5.	Amazon Neptune	Multi-model 	2.89	-0.19	+0.33
5.	5.	 3.	GraphDB 	Multi-model 	2.47	+0.01	-0.41
6.	6.	6.	Stardog 	Multi-model 	1.66	-0.10	-0.27
7.	7.	7.	AllegroGraph 	Multi-model 	1.42	-0.02	+0.20
8.	8.	8.	Blazegraph	Multi-model 	1.14	-0.07	+0.18
9.	9.	9.	RDF4J	RDF	1.09	-0.02	+0.42

Figure 3.1: RDF storage providers ranked by popularity according to solid IT⁵ [57].

Two methods of enforcing policies that comprise implementations of the alternatives presented above are view filtering and query rewriting [25].

View filtering This strategy comprises forming a view with only allowed data where the controlled access to data is guaranteed by default.

Query Rewriting For scenarios where the request contains any kind of querying strategy, a frequent approach is to apply modifications to the request in such a way that the resulting query only retrieves or affects allowed data.

3.2 Existing Technologies and Solutions

Among the services that offer solutions for storing RDF data, there are multiple approaches for implementing security measures and access control mechanisms. In this section, we analyze the access control mechanisms and strategies of a set of services that provide the means to store RDF data. Included in the set of reviewed services are Apache Jena and Eclipse RDF4J which are applications that are specialized in the storage of RDF triples, and the likes of GraphDB, Stardog, Virtuoso, and Amazon Neptune, which comprise multi-model graph database systems capable of handling, among other graph structures, RDF data. These services are a selection of those present in a list provided by solid IT⁴, depicted in Figure 3.1, which ranks the database management systems (DBMS) that provide solutions for storing RDF according to their popularity. RDF storing services are usually structured as Figure 3.2 depicts. The triplestore can possess a set of datasets that in turn may hold several different graphs. These three units are effectively separated and triplestores provide different access control mechanisms at their levels. Since their designations in each provider often differ, in Figure 3.3 we present the mapping of those designations to the ones we adopted.

⁴solidIT, <https://solid-it.at/>

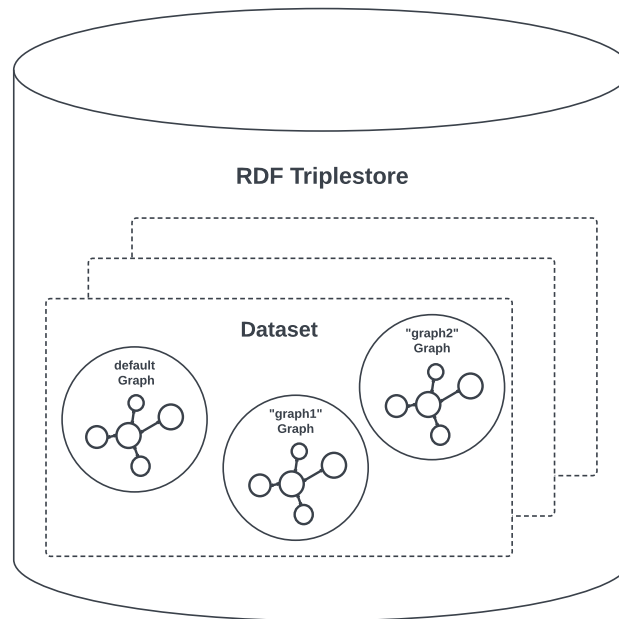


Figure 3.2: Structure of RDF triplestore.

Apache Jena Fuseki

Apache Jena Fuseki [58] offers access control mechanisms at different levels. The broader level is the server level which sets which users can send requests to the server [59]. Users that are not in the server’s ACL will not be able to perform any kind of operation. At a lower level, there is the dataset level which manages the users that are able to access and update the data stored in a specific dataset of the triple store.

Next is the level of the dataset’s endpoints. For each dataset, Apache Jena Fuseki offers various endpoints that cater to the Graph Store and SPARQL Protocols. To each one of these types, it provides endpoints for read-only operations and endpoints for update operations. For example, for a dataset named “default”, the Jena Fuseki server contains the following endpoints: for the Graph Store Protocol (Read) `/default/` and `/default/get`, for the Graph Store Protocol `/default/` and `/default/data`, for SPARQL Query `/default/`, `/default/query`, `/default/sparql` and, at last, for SPARQL Update `/default/` and `/default/update`. By providing access control over this set of endpoints, Jena Fuseki essentially restricts which actions users can execute over a dataset’s data. At last, at the graph level, it is also possible to define access restrictions, making it the application’s most fine-grained kind of access control.

For specifying the permissions across the different access control levels, Apache Jena Fuseki follows an approach that resembles a MAC model as it relies on ACLs that can be defined in the application’s RDF configuration file. There, it is required to define the user accounts and subsequently associate them to the server’s, the datasets’, the endpoints’, or the graphs’ allowed-users list.

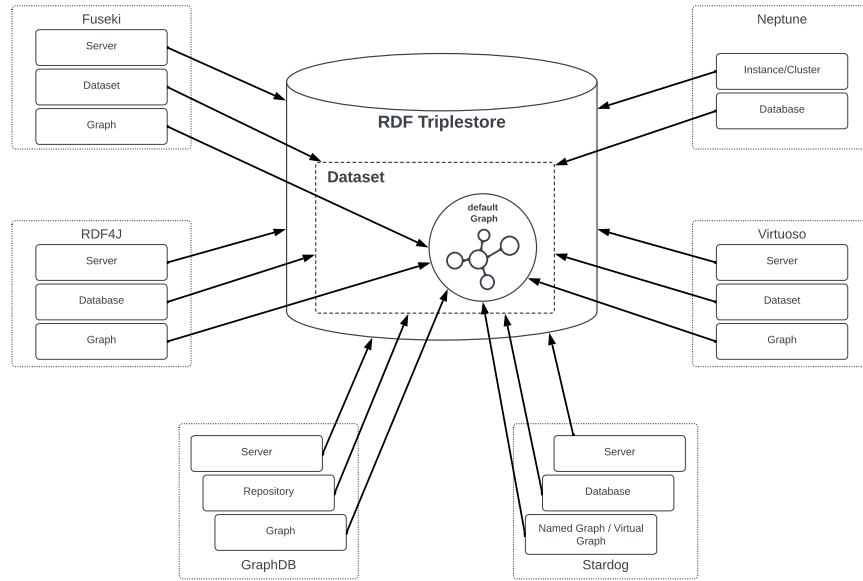


Figure 3.3: Mapping of designations of RDF triplestore providers.

Eclipse RDF4J

Eclipse RDF4J⁶ is a Java framework that originates from a fork of Sesame⁷, a project that was one of the first providers of RDF storage and responsible for many developments in the field such as the presentation of SeRQL, which, like SPARQL, is an RDF query language. RDF4J aids the processing and handling of RDF data through its multiple APIs and, over the module RDF4J Server, offers an RDF store solution along with integration with third-party RDF stores such as GraphDB, Stardog, Amazon Neptune, and Virtuoso. When using RDF4J Server, it is possible to set up the users' access rights and security mechanisms. RDF4J follows a strategy where users have access rights to resources, roles are an aggregation of access rights and users can be associated to roles [60]. Similarly to Fuseki, RDF4J requires the definition of users, roles, and access rights over its configuration files. In these files, the access rights to the resources are defined with constraints composed of URL patterns and HTTP methods and their association with roles and users. This results in an RBAC model at the level of the server, database, endpoint, and graph for read, write and delete operations, due to the flexibility given by the combination of both components of access rights constraints.

GraphDB

Ontotext's GraphDB⁸ follows a different approach, instead of having to assign users to the different levels of security like Apache Jena Fuseki, it implements an RBAC model to administer its

⁶RDF4J, <https://rdf4j.org>

⁷Sesame, <https://sourceforge.net/projects/sesame/>

⁸GraphDB, <https://graphdb.ontotext.com/>

users' access restrictions [61]. GraphDB implements its AC model according to the Hierarchical RBAC from the NIST standard or otherwise known as RBAC1 [62]. This standard describes a hierarchy of roles, where each one agglomerates a set of permissions. Given its hierarchical structure, some roles may inherit the permissions of other roles according to their position in the hierarchy, i.e. it contains its permission and the permissions of all its parent roles. GraphDB's model comprises a collection of built-in roles and their hierarchy, along with their respective set permissions. Users have attributed one or more roles and get their corresponding access rights. Concerning these access rights, GraphDB defines access and operation restrictions at the level of the server and at the level of the repository.

Stardog

Stardog⁹, similarly to GraphDB, implements an RBAC model. This model is based on a standard, simpler version of RBAC where it is not considered a hierarchy between the different roles. Stardog's documentation summarizes their strategy as follows: users have a set of permissions over resources, permissions can be grouped into roles, and roles can be assigned to users [63].

The list of resources over which Stardog allows the definition of permissions is long; here, we only mention the ones that are pertinent for the protection of data: database, named graph, virtual graphs, sensitive properties, and stored queries. For each resource, Stardog has a group of actions that can be used to create valid permissions. These are "read", "write", "create", "delete" whose functions are self-explanatory, "grant" and "revoke" that permit granting and revoking new permissions over resources for other users or roles, and "execute" which allows the execution of operations against resources, not used for the types of resources mentioned above. Finally, the action "all" that aggregates all of the actions.

With the conjunction of both these concepts, Stardog enables several different types of permissions at different levels. At the level of the database, equivalent to Jena Fuseki's dataset, it allows the definition of policies for the actions "read", "write", "create", "delete", "grant" and "revoke". For virtual graphs, which are graphs containing mapped data from other types of databases like relational, NoSQL, and JSON databases, it allows the same set of actions with the exception of the "write" action, still to be implemented in the future. As for named graphs, it only allows "read", "write", "grant" and "revoke" [64]. At last, sensitive properties can only be associated with a read action, they have to be created and maintained with Stardog's environment configuration tools mentioned below [65]. Nevertheless, Stardog's sensitive properties are the most fine-grained strategy for protecting data seen among the analyzed services, it is the smallest data unit with the possibility of being protected. With them, we are able to secure a set of triples of RDF data that is surrounded by other assertions. This can be illustrated by the example provided by the service's documentation. Considering a database holding information about a person and a property linking that person to his social security number, it is possible to add that property to a sensitive property

⁹Stardog, <https://www.stardog.com/>

list. Then, after defining what users or roles have permission to read it, all queries not performed by the permitted users or roles are not able to retrieve the information stored with that property.

Stardog's security policies, the attribution of permissions to roles and users and the association of users to roles, can be managed through the Admin CLI, the available API, and the Stardog Studio GUI [63].

Virtuoso

Virtuoso¹⁰ is a multi-model DBMS possessor of RDF store capabilities. On this subject, Virtuoso's documentation is not very clear and specific about its security measures. Still, it is clear that the implemented mechanisms available by default are not as sophisticated as what other services like GraphDB or Stardog offer. Virtuoso's approach appears similar to the one implemented by Apache Jena Fuseki and in part to some of the roles that come built-in with GraphDB. In Virtuoso's system, users can have roles that allow them to perform read or update operations at the level of the server, which amounts to a fusion of Apache Jena Fuseki's server, dataset, and endpoint access control levels [66]. Aside from this, users can also own RDF graphs and specify the remaining users' read and write privileges on that graph, a feature equivalent to the concessions given to some of GraphDB's default roles.

Supported by its multi-model system, Virtuoso allows the mapping of other models to the RDF store model and, through this connection, the security features of the original model are enabled for the RDF model view. For example, if an RDF graph is the virtualization of SQL data, then the security controls of the relational model can be applied. Their specification, however, falls out of the scope of this work, as it intends to gauge only the security methodologies for RDF data stored in an RDF store-modeled database system or the strategies of similar graph-based database systems. Additionally, the documentation points out that it is possible to have a more sophisticated ontology-based security model but, such a structure is only available after consultancy with the Virtuoso team.

Amazon Neptune

Amazon Neptune¹¹, like most of the services analyzed so far, is a graph database engine capable of hosting databases of multiple models. As Neptune's documentation is not clear about which AC mechanisms are applicable to the RDF model and which are applicable to other models, we assumed that every mechanism presented can be used to secure RDF data. Still, it presents AC mechanisms at different degrees. With administrative policies, Neptune allows the control of access to the server and the database, together with other types of policies that are not part of the focus of this work [67]. With data-protection policies, Neptune offers methods for setting security rules for accessing the data held by the databases, that ensure the safety of the stored data. These policies restrict what types of queries a user or group is allowed to execute. Depending on

¹⁰Virtuoso, <http://docs.openlinksw.com/virtuoso>

¹¹Amazon Neptune, <https://aws.amazon.com/neptune/>

RDF store	AC Model	Granularity				Configuration
		Server	Dataset	Graph	Triple	
Fuseki	MAC	✓	✓	✓		RDF file
RDF4J	RBAC	✓	✓	✓		XML file
GraphDB	RBAC	✓	✓	✓		?
Stardog	RBAC	✓	✓	✓	✓	CLI and GUI
Virtuoso	DAC, RBAC	✓	✓	✓	?	CLI
Neptune	RBAC	✓	✓	-		GUI

Table 3.1: Summary of access control features of RDF stores.

the query language, and its underlying database model, as is Cypher for property graphs, Grem-lin for OLTP graphs, and SPARQL for RDF graphs, Neptune maps language statements to the actions “ReadDataViaQuery”, “WriteDataViaQuery”, and “DeleteDataViaQuery” to later analyze received queries and require permissions to execute the statement contained in them [68]. Therefore, similarly to other services Neptune’s policies are able to protect data at the level of the server, database, and the actions performed when accessing a database.

3.3 Related Research Work

Despite the existence of many research works with the intent of performing access control by adopting Semantic Web technologies, not all of them are dedicated to protecting linked data systems or RDF data collections. For example, Kodali et al. [69] developed an authorization model to cater to multimedia digital libraries using ontologies and Semantic Web technologies, while Cirio et al. [70] produced a generic access control system that implements both role and attribute-based ACM. At the same time, among current works in the literature, there are also multiple instances relative to the application of access control to linked data. Some focus on a particular aspect of access control or a specific component of the Access Control Framework, like the implementation of a specific Access Control Model for linked data, the representation of policies by adopting a determined language, or the implementation of strategies for access control enforcement. Others provide complete solutions and detail the implementation of access control for RDF and linked data systems. This section focuses on examining a collection of research works and inspecting the adopted strategies for implementing the different components of the Access Control Framework.

3.3.1 Methodology

To find relevant works we followed a strategy that resulted in a selection of works mainly from the following four sources:

1. the works reviewed in the survey completed by Kirrane et al. [25]

2. a search for works that cite Kirrane et al. [25]’s survey or the most cited works in the area of access control for linked data it reviews, such as Dietzold and Auer [49] and Abel et al. [10].
3. a structured search procedure by a combination of keywords and terms (“Access Control”, “Authorization”, “Protection”, “Privacy”, “Linked Data”, “Semantic Web”, “RDF”, “Web of Data”)
4. a search for more recent works and contributions by the authors of the works found in the previous sources

The set of analyzed works is a collection of research works we considered to present valuable contributions to the subject of performing access control over linked data systems. The survey by Kirrane et al. [25] reviews a large group of works although not all of them focus only on enforcing access control over linked data. The authors also present works that use Semantic Web technologies to provide access control in other contexts, which were left out of this analysis. Still, we considered it a good starting point due to the large number of works it details and its number of citations. Regarding the two later sources, we carried out search activities using Google Scholar, the main source of research works, and also Springer, Elsevier Scopus, Web of Science, Science Direct, IEEE Xplore Digital Library, and ACM digital library for the reason of being some of the most acclaimed research databases.

From these sources resulted a group of more than research works that we briefly analysed and filtered by subjecting them to a selection process that excluded the ones that did not treat the specific case of applying access control specifically for linked data. We considered the ones that in some way contribute to finding or describing solutions to apply access control to RDF data or platforms of linked data. Works that used Semantic Web mechanisms and technologies to cater for other contexts such as to different graph based structures were thus left out of our survey.

In Table 3.2, are presented the works that we analyzed arranged by their primary area of focus. In this table, we can see that there is a larger group of works that describe full solutions rather than describing just a part of the Access Control Framework, in which we concentrated the efforts of our analysis. While the works that are present in the first two rows of Table 3.2 still provide relevant contributions to the subject, they either were later used and extended or they comprise approaches very similar to the ones followed by works that describe the implementation of full AC solutions. In the case of Sacco and Passant [71] and Villata et al. [72], the achievements of these works was followed by Sacco et al. [73] and Costabello et al. [74], where their results were used to provide a method for the specification of Access Policies. In the case of Fernández et al. [75], a similar approach was followed by Benaribi et al. [76], as the authors equally pursue the enforcement of access control over RDF data using encryption techniques. Finally and in a similar way, the results of Li and Cheung [77] and Flouris et al. [78] are parallel with the results produced by Kirrane et al. [12] and by Dietzold and Auer [49] and Muhleisen et al. [79], respectively, which we detail below. What follows is a survey of the works that detail access control approaches

for RDF and linked data platforms. In Table 3.3, we present a comprehensive overview further described in the following subsection, that summarizes the characteristics of each of them.

Topic of focus	References
Implementation of Access Control Models and Policy Representation	Sacco and Passant [71], Villata et al. [72],
Policy Enforcement	Fernández et al. [75], Li and Cheung [77], Flouris et al. [78]
Access control for RDF and Linked Data Platforms	Hollenbach et al. [80], Reddivari et al. [81], Abel et al. [10], Dietzold and Auer [49], Gabillon and Letouzey [11], Franzoni et al. [82], Lopes et al. [83], Kirrane et al. [12], Kirrane [84] Muhleisen et al. [79], Sacco et al. [73], Costabello et al. [74], Unbehauen et al. [85], Stojanov et al. [56], Benaribi et al. [76]

Table 3.2: Relevant works in application Access Control by topic of focus.

3.3.2 Outline of Analyzed Research Works

After the analysis we performed over the final list of works, we summarized their main characteristics in a comprehensive table, Table 3.2. Through our analysis, we understood where most works diverge from each other. Not only do they follow different strategies for the implementation of the three components of the ACF, but also in three other features, the protected actions, the granularity of protection, and the strategy for conflict resolution. By inspecting the contents of Table 3.2, we can notice that each work is very singular as there is a large set of combinations of strategies followed for the different aspects. This not only validates the nonexistence of a standard that we mentioned earlier in this chapter but also makes it challenging to aggregate them into different groups. So, we outline their approaches in each of the most critical features where they differ.

Access Control Model

As we pointed out above, the ACM defines the structure of the access control process, for which multiple strategies can be applied. Most works implement CBAC models, thus providing superior precision in defining access policies, particularly in the specification of the policy's subject. The works of Abel et al. [10], Franzoni et al. [82], Sacco et al. [73], Costabello et al. [6], Unbehauen et al. [85], and Stojanov et al. [56] all implement a CBAC model or empower their systems with features of this model. Dietzold and Auer [49], despite more formally implementing a VBAC model since the implemented system organizes the data in views, also considers features of CBAC by allowing the specification of contextual information and its usage for policy evaluation. Moreover, they show features of a DAC model since their strategy allows the attribution of an owner to a view and a subsequent definition of permissions for the other users. Similarly, Gabillon and

Letouzey [11] present a VBAC-based solution while other cases implement different models such as MAC, DAC, and RBAC. Hollenbach et al. [80] follow a MAC approach, Lopes et al. [83] describe the implementation of an RBAC model, and Kirrane et al. [12] detail the application of DAC features to the RDF data model. Finally, the remaining works of Reddivari et al. [81], Muhleisen et al. [79], and Benaribi et al. [76] developed systems that do not implement a specific ACM as they describe only the definition of subjects and objects; and a simple association between them.

Policy Representation

Relatively to policy representation, the collection of approaches was also diverse. Among these alternatives, aside from the authors that used RDF and RDF-S for representing policies, there were not provided much detail about the actual policy representation and further interpretation process. Dietzold and Auer [49] and Muhleisen et al. [79] used SWRL rules but do not specify how they are evaluated, although in the work of the latter the author mention that any reasoning program can be used and use Pellet to assess their approach. Franzoni et al. [82] use XACML, and mention that their implementation follows XACML's conceptual architecture, while the works of Reddivari et al. [81], Stojanov et al. [56] and Benaribi et al. [76] described self-implemented languages to specify access policies. Hollenbach et al. [80], Abel et al. [10], and Kirrane et al. [12] only formalize a conceptual representation of policies and do not define a concrete strategy for their specification. However, Abel et al. [10] noted that their conceptual policies can be mapped to other existing AC policy languages. Finally, Sacco et al. [73], Costabello et al. [74], and Unbehauen et al. [85] make use of RDF and RDF-S vocabularies and use SPARQL to interpret policies, whereas Lopes et al. [83] use annotations on RDF triples and also a query language for annotated RDF graphs Zimmermann et al. [86] to achieve the same effect.

Policy Enforcement

Regarding enforcing the access policies, the works reviewed mainly follow one of two strategies we have detailed earlier: view filtering and query rewriting. Here, the number of times each strategy was used in the set of works we analyzed is evenly distributed. Dietzold and Auer [49], Gabillon and Letouzey [11], Sacco et al. [73], and Stojanov et al. [56] elected to follow an enforcement strategy based on a view filtering approach; and Abel et al. [10], Franzoni et al. [82], Kirrane [84], Costabello et al. [74], Unbehauen et al. [85], and Benaribi et al. [76] follow a strategy based on query rewriting. However, despite being possible to aggregate these works into two groups, their underlying procedures are distinct as there are multiple approaches for implementing both strategies, which we describe in the next subsection. Outside of these two groups are the works by Hollenbach et al. [80], Reddivari et al. [81], Lopes et al. [83], and Kirrane et al. [12] that do not follow any of both strategies and do not fit in either of these two categories.

Protected Actions

Regarding the set of protected actions in the systems developed in each of the analyzed research works there was also disparity. In this case, either the developed systems protected only read actions or protected both read and update actions. Read actions are associated with any operation whose is to retrieve information from the protected RDF source while update actions are associated with operations that make modifications to that same data. From the set of analyzed works, Sacco et al. [73] and Benaribi et al. [76] only consider the protection of read actions, Muhleisen et al. [79] consider both read and update operations but do not detail the specific of implementation of the latter, and Hollenbach et al. [80], Reddivari et al. [81], Dietzold and Auer [49], Gabillon and Letouzey [11], Franzoni et al. [82], Costabello et al. [74], and Stojanov et al. [56] fully implement read and update operations while the remaining works, developed by Abel et al. [10], Lopes et al. [83], Kirrane et al. [12], and Unbehauen et al. [85], do not mention what type of protected actions their solutions support.

Protection Granularity

Another aspect in which the implemented solutions from the analyzed works differ is in the granularity of protection. This aspect pertains to the type of object to which an access policy refers to. Among the analyzed works there are four levels of protection: the level of the RDF file, equivalent to the dataset level in the protection provided by RDF stores; the RDF graph level, that can refer to the existing graphs within a dataset; the RDF triple level, that allows the specification of RDF triples from inside a graph; and the RDF quad level, which is similar to the triple level but allows the definition of the graph where the triple is inserted. However, regarding the latter two levels of protection there is one additional detail, some works allow only the specification of a single triple or quad, while other allow their specification through patterns, making it possible to select a larger quantity of triples or patterns. Hollenbach et al. [80] provide the least fine protection among the group of analysed works since it is the only one that secures data only at the level of RDF files. Costabello et al. [74] and Sacco and Passant [71] provide access control at the level of the RDF graph, enabling the protection of a datasets' graphs. Stojanov et al. [56], provides protection of the level of the RDF quad, while the remaining works provide protection at the level of the RDF triple.

Conflict Resolution

At last, in terms of conflict resolution, one of two things occurred: either a conflict resolution strategy was implemented, or a strategy for this aspect was not required due to the impossibility of conflicts occurring among policies. When a strategy was implemented, instead of implementing complex strategies such as the one implemented by Stojanov et al. [56], who developed a process based on priority values to specify the order of evaluation of the access policies, most works followed a defaulting strategy. A defaulting strategy is characterized by the anticipated definition of a behaviour to follow upon the rise of a conflict. For example, Abel et al. [10] elected to give a

higher priority to negative policies, and in case there are no specified policies over a given object, deny the access to that object.

3.3.3 Access Control for RDF and Linked Data Platforms

After in the beginning of this chapter we and in the previous subsection providing an overview of the analyzed works and the strategies chosen for a number of features in which they differ, in this subsection we provide a detailed characterization of the work accomplished in each of the analyzed research works.

Hollenbach et al. [80] describe a process that enables access control over RDF documents published on the Semantic Web. The authors envision an environment fully committed to the values of the Semantic Web, where we cannot rely on centralized authorities for handling authentication and authorization. Therefore, they implement an authentication module based on a decentralized authentication protocol, FOAF+SSL, whose certificates use FOAF files. The authorization is handled by evaluating user-written RDF files in N3 format that describes the allowed actions each requesting agent has over a given RDF file stored in the system through an Access Control List. The module evaluates whether the agent has permission to perform any of the three operations described by the authors: “read”, “write”, and “control”. For authorizing a request, the module uses the information obtained in the FOAF files received during the certificate exchanging phase to make a decision. In short, the authors followed an approach that enables the definition of access control mechanisms capable of restricting the execution of a set of operations over RDF files, stored in a web server. The authors of other research works opted to follow a different path and implement mechanisms that provide protection at the level of the RDF data stored in triplestore services, instead of at the level of RDF files.

In Reddivari et al. [81] the authors document the implementation of RAP, a framework based on an early version of the Jena framework, whose purpose is to enable the storage of RDF data while allowing the specification of policies to restrict the operations that accessing users are permitted to run. The work starts by describing the actions an RDF store usually presents and arranges them into four types: insert, delete, update, and query; at the same time, they describe part of the available predicates that can be used to define policies. Such predicates are codified in a Prolog-like notation and are converted to rules in a notation supported by Jena to be later evaluated by its reasoner. Using a set of predicates such as “permit” and “prohibit”, it is possible to define detailed policies by constraining them with triple and user-specific conditions. Additionally, RAP allows the definition of policies using metadata about the triples like triple ownership and creation details, information that is stored outside of the RDF store, in persistent Java hash-maps. The data protected by each rule is selected through RDF triple patterns, meaning that rules may be relative to a single triple or to a set of triples. Finally, toward the end of the article, the authors describe the architecture of the RAP framework and the components it is made of. After presenting the details regarding the role of the RDF store in the system and the contents that it stores, the authors

provide the implementation specifics of the RAP RETE Policy engine which is based on the Jena framework's reasoner. Furthermore, they describe the algorithms for the enforcement of access control in each of actions supported by the framework, and finally the Web service that serves as an interface for the execution of operations.

Abel et al. [10] describe the implementation of a system that enables access control in RDF Stores while considering information pertaining to the context in which the request is made, thus implementing a CBAC model. The authors implement query rewriting techniques to enforce the access control policies to the requests received. Such techniques involve rewriting methods for the Sesame RDF Query Language (SeRQL); nevertheless the authors claim that the same methods can be applied to other query languages like SPARQL. For the matter of policy representation, the authors mention that it was not put much effort and state that no specific language was chosen or implemented to represent policies. Instead, the authors present access policies in a high-level syntax, that we exhibit in Listing 3.1, and say that it can be mapped to existing policy languages.

```

1 allow/deny access to triples <RDF graph pattern / RDF triple> IF <RDF graph pattern
  / RDF triple / contextual condition> (AND)
2
3
4 allow access to triples (l3s:alice, foaf:phone, Z)
5
6 deny access to triples (X, foaf:phone, Z) IF (X, foaf:currentProject, l3s:reverse)
  AND Requester = 'RecommenderService'
```

Listing 3.1: Structure and example of policy representation in a high-level syntax by Abel et al. [10].

This syntax allows the definition of rules that grant or deny access to a specific triple or to a set of triples that match a given graph pattern. The decision of whether to permit or revoke access to data is made through the result provided by the implementation of an algorithm that evaluates both the request received, and the existing specified policies.

The authors' implementation is capable of handling read requests for queries of both SELECT and CONSTRUCT typology. The implemented enforcing method involves binding to the queries received clauses to ensure that only allowed data is returned. For policies of type 'deny' the clauses are added to a MINUS section, so that results that match the patterns of triples specified are removed. On the other hand, for policies of type "allow" the clauses are added to the WHERE section. At last, for conflict resolution, the authors configured the system to assume a default behavior of denying access to unprotected triples. Aside from that, it gives "deny" policies a higher priority relatively to "allow" policies therefore, the steps for deciding the permission of access to data are:

1. If there is a deny policy, then deny access;

2. If there is an allow policy, then allow access;
3. Or else, deny access;

Dietzold and Auer [49] A different approach was followed by Dietzold and Auer [49]. The authors describe the implementation of a system that enables access control for RDF triple stores following a VBAC model, as they build a virtual model for the queries to be run against, ensuring that only allowed data is returned. Initially, they discuss some requirements for access control in the context of RDF triple stores and state that access control for such cases should be available in multiple granularity levels. Aside from controlling access to a specific resource or triple, they defend that it should also be possible to control the access to both triples that have the same subject, and instances of a given class. The authors describe their system as the combination of 4 main components, a query engine, a rule processor, a vocabulary for policy specification, and an access control processor. The query engine is used for running queries and building the needed virtual models, the rule processor is used for interpreting the policies represented as SWRL rules which also utilize a vocabulary, more specifically, the LACS vocabulary. At last, the access control processor is the central component and is responsible for managing the remaining parts of the system components. Access Policies or, as the authors name them, “rules” comprise filters, composed of a representation of the data controlled by a given policy, which ultimately build/construct a smaller model of only allowed data, and SWRL rules that specify the conditions required to activate that filter. The system supports the definition of rules that control three different types of action: reading, adding, and deleting sets of triples. Following a VBAC approach means that the execution of these actions is as follows:

- For read-type actions, the filters are applied to the existing RDF model, the one containing all of the system’s RDF data, to build a virtual model to run the queries.
- For writing and removing information, a model is created, containing the triples to be inserted or deleted. Then, the filters of the rules are applied to this model, the triples that subsist this filtration process are the ones that are effectively inserted or deleted, depending on the type of operation being run.

Regarding the priority of rules and conflict resolution, the authors don’t mention any particular technique, even though there is a property in the LACS vocabulary that allows the association of a priority number to rules.

Gabillon and Letouzey [11], like Dietzold and Auer [49], believe it is possible to organize RDF data in separate views with CONSTRUCT queries. The authors propose a strategy parallel to the one generally implemented in relational database management systems accessible through SQL: the creation of views that are owned by the DBMS users, who control who is allowed to access them.

Therefore, the authors propose the association of views with users through the implementation of ACLs (Access Control Lists). They describe a language based on first-order logic which

follows the structure present in Listing 3.2, where subjects are users or processes, actions are the four SPARQL query types, SELECT, ASK, DESCRIBE, and CONSTRUCT, and objects are RDF graphs or previously constructed views.

```

1 <condition> -> Permit(<subject>, <action>, <object>)
2
3 Time(CLOCK,t) ∧ (t > 8) ∧ (t < 20) -> Permit(s,ASK,foafview.txt)

```

Listing 3.2: Structure and example of policy language in Gabillon and Letouzey [11].

The data that should be part of these views are stored in text files, where each file contains a SPARQL query of type CONSTRUCT, that returns a model with the view's containing triples. Despite being a view-oriented approach and following the structure of ACLs, and given that are developed strategies to evaluate conditions regarding the subject and the request, the authors' implementation shows features of a CBAC model. Additionally, it shows features of DAC attending that there is a hierarchy and ownership of graphs and views and subsequent policy creation rights associated with these entities. As a conflict resolution technique, the system adopts the strategy of defaulting to a determined behavior, already seen in Abel et al. [10].

Franzoni et al. [82] Another work that presents some similarities with the work of Abel et al. [10] is the work of Franzoni et al. [82]. Like in [Abel et al.](#), the authors implement query rewriting techniques for SeRQL for enforcing AC. Here, the authors describe the development of the Fine-Grained Access Control Model (FGAC Model) which aims to target a use case from the digital library field. The FGAC model is composed of two main components, the Access Control module, and the FGAC module. The Access Control module is responsible for analyzing access requests and determining if they should or not be granted and the FGAC module is responsible for effectively enforcing the policies and ensuring that the data retrieved is only allowed data. Regarding policy representation, in this work's solution, access policies, like the user access requests, are specified in XACML and are analyzed by the Access Control module which is implemented according to XACML's conceptual architecture, however with some modifications. The text files storing the FGAC policies are kept in the file system, stored in text files, and are capable of controlling access at the level of the triple, through the individuals' classes.

Lopes et al. [83] In contrast to what we have seen in other works so far is the strategy used by Lopes et al. [83]. In this work, the authors propose a new query language that is an extension to SPARQL named AnQL. This language depends on Annotated RDF Graphs which are the resulting work of a previous article, elaborated by Zimmermann et al. [86]. The main goal is to have an Annotated RDF graph secured by the AC policies defined in a logic programming environment, to then convert the received SPARQL queries to AnQL and securely retrieve the requested information. In practice, an annotated triple is a quad where the first three elements are the elements of an RDF triple and the last is the annotation. This can be represented with the notation in Listing

3.3. As an example, the authors provide the annotated triple presented at the end of Listing 3.3 that uses an unknown vocabulary and may represent that Joe Bloggs works for Westport Cars as a part-time employee.

```
(:subject, :predicate, :object) : annotation
(:joeBloggs, :worksFor, :westportCars) : 0.5
```

Listing 3.3: Notation and example of an annotated RDF triple.

The authors describe how the annotations can be used to model AC lists and to specify access control restrictions for concepts like users, roles and groups. They add that it is possible to use RDF-S inference rules, like using `rdfs:subclass` to infer restrictions to additional triples and mention that the definition of custom rules, using custom properties, is also allowed. Regarding create and update operations, they mention that these were still not available in SPARQL at the time of writing so they were not considered and were left for future work. In following work [87], they attain actions of write and delete, as they use the built infrastructure implemented in Prolog, with the help of SWI-Prolog's Semantic Web library to cater to an enterprise use case where data from a relational data model, along with its AC policies, needs to be migrated to the RDF model.

Kirrane et al. [12] and Kirrane [84] A different approach and techniques were pursued by Kirrane et al. [12], where the authors describe a process to add DAC features to the RDF data model. Initially, it is presented the functioning of DAC in the relational model and tree-based models to describe afterward how RDF's graph-based model differs from the previous two and how DAC can be applied to it.

Following this, the authors detail what they consider to be the two types of data-oriented authorizations: graph-based and schema-based authorizations. Later, they add how each type of authorization can be used to select resources to be protected and how authorizations can be derived using each type's concepts. For graph-based authorizations, the authors propose three alternatives: to derive authorizations to all resources connected to a given resource through any kind of property; to derive authorizations along a particular property path, recursively; and to implement graph search algorithms that traverse the graph and derive authorizations to specific nodes, this last is, however, left for future work. As far as schema-based authorizations, the authors suggest five rules that rely on the vocabulary of RDF Schema. The first and second rules refer to when an authorization is assigned to a Class or Property and how that authorization can be derived to all instances of both concepts through the use of the `rdf:type` property. Rule 3 details how authorization can be derived to all the properties associated with an instance of a class when it is the case that authorization is assigned to that instance, which can be attained using the properties `rdf:type` and `rdf:domain`. At last, rules 4 and 5 detail how the properties `rdfs:subclass` and `rdfs:subproperty` enable the derivation of authorizations specified to a class or property to all of

their subclasses and subproperties, respectively. Additionally, the authors mention that it is possible to take advantage of these propagation strategies to formulate more expressive restrictions by applying them to richer vocabularies or even OWL ontologies. They detail the different access rights that can be associated with RDF. Such rights are the outline of the SPARQL language as they are comprised of the different operations the language permits. The authors specify the following commands as possible action restrictions: for “read” operations the commands SELECT, CONSTRUCT, ASK, DESCRIBE; for “update” operations INSERT, DELETE and DROP and for graph management, even though no examples of how to use them in the description and further enforcement of AC policies are provided, the commands CREATE, COPY, MOVE and ADD. Additionally, they specify an access policy as a quad, structured as is presented in Listing 3.4, where each fraction represents the subject who requests permissions, the access right of that policy, the decision taken when the policy is activated, either grant or deny, and lastly the resource(s) being protected. The representation and enforcement strategies for these access policies are not formally presented.

```
1 <Subject, Access, Sign, Resource>
```

Listing 3.4: Structure Quad tuple representing elements of an access policy [12].

Nonetheless, the authors suggest the addition of commands to SPARQL to deliver an access control solution that mirrors the behavior of a DAC model. The commands suggested are GRANT and REVOKE, and their behavior is said to be similar to the one that can be seen in relational and tree-based DBMS. Along with this proposal, the authors provide the syntax and describe how valid commands of this type are structured.

Finally, the authors describe their approach to the resolution of conflicts among policies. The strategy follows three premises, explicit policies override implicit ones; most specific along a path takes precedence; denial takes precedence. When no policies are specified the authors mention that they adopt a defaulting behavior that can lean towards a closed policy, by denying access by default or towards an open policy, that allows access by default.

While in Kirrane et al. [12] the authors do not specify a concrete strategy for the enforcement of access control, later in further work, Kirrane [84] describes the process of implementation of query rewriting for a series of SPARQL operations. Here, Kirrane provides the rationale for rewriting the querying operations of SELECT, ASK, CONSTRUCT and DESCRIBE as well as some of the different sub-operations that can be used within them (filters, aggregate and property path operations). Additionally, Kirrane also provides rewriting techniques for the update operations of INSERT/DELETE (DATA), CLEAR, DROP, ADD, LOAD, COPY, and MOVE. At last, the author describes an evaluation strategy based on the Berlin SPARQL Benchmark [88, 89], that aims to gauge the correctness of the developed techniques and compare their performance with the strategy of view filtering. Regarding the correctness, Kirrane concludes that the rewriting strategies both for querying and update operations are effective according to a criteria for verifying the correctness of SPARQL, based on a previously published criteria for relational databases. Relatively

to the matter of performance, the author establishes that for aggregating functions of querying operations and operations of INSERT/DELETE (DATA) query rewriting is advantageous, whereas in regard to filter suboperations and graph-related SPARQL operations (CLEAR, DROP, ADD, LOAD, COPY, and MOVE) query rewriting displays equal or worse results when compared to view filtering.

Muhleisen et al. [79] In contrast with the rationale of both Lopes et al. [83] and Kirrane et al. [12] is the work of Muhleisen et al. [79], who follow a view filtering approach as they describe the process of implementation of Policy-enabled Linked Data Server (PeLDS). The authors start by defining the concept of Access Control policies as a set of rules, specified in the SWRL language, whose evaluation dictates whether a user can access a determined data object. The authors distinguish the different types of Access Control Models, which they consider to be only DAC, MAC, and RBAC, and review the different characteristics between each one of them. Next, they evaluate three abstraction levels of an RDF Graph, the syntactical level, the data model level, and the semantic level. They consider the RDF Graph in its serialized form to be the syntactical level; the graph's decomposition into triples the data model level; and the concepts and relationships associated with RDF triples the semantic level. From these three levels, only the latter two were considered worthy of implementing AC methodologies since that the abstraction level of serialized RDF graphs has been shown to be ineffective by Jain and Farkas [90].

Finally, the authors describe the architecture and functioning of the PeLDS. They start by describing the supported policy language, PsSF, a notation based on SWRL, which enables the specification of access to triples, resources, and, through ontology specification, concepts of the domain. Upon receiving a request, the PeLDS binds the context values of the current user to the variables and evaluates the AC policies using an OWL reasoner. The system only supports positive access rules meaning that conflicting decisions after the policies' evaluation never occur, since that it is not possible. Therefore, the system either allows access to data because some policy grants that action or, in any other case, the access is denied. The enforcing strategy is building a virtual graph of allowed information upon receiving requests. Even though the system currently supports read and update actions, the method used to control the execution of the latter is not detailed.

Sacco et al. [73] While in most of the works seen so far make use of SWRL rules and other rule-based languages for specifying policies, Sacco et al. [73] adopt the use of RDF-S vocabularies and OWL ontologies. In this work, the authors describe how the Privacy Preference Ontology (PPO), an ontology for describing access restrictions for linked data, developed in previous work by Sacco and Passant [71], can be used in a system that enforces the access permissions the language describes which they name Privacy Preference Manager (PPM). The PPO allows the specification of reading and writing actions in policies, and offers AC for resources, triples, and named graphs. Since the PPM is built to cater to the social web to aid the sharing of FOAF profile information, the write action has not been put into use yet, given that permitting other

users to modify one's profile is senseless. In the PPM, the access rules are enforced by a data-filtering algorithm that evaluates the policy constraints using SPARQL ASK queries to build a subset of the protected data containing only information that the requesting user is allowed to see. In further work, Sacco and Breslin [91] extend and improve the PPO and the framework proposed in this work. In regards to the ontology, the authors added the concepts of conditional operators to provide richer combinations of access conditions, a new type of policy to enable the definition of both positive and negative access policies, support for priority values to specify the order in which policies are evaluated and a new set of possible targets a policy can be applied to. Additionally, the authors introduce the Privacy Preference Manager Ontology (PPMO), an ontology that provides semantics to describe the which users can create access policies, diverging from the previous version of the system that assumed that only the data owners could perform such task. Finally, regarding the PPM, the authors add onto the previous version so that it considers the improvements made at the level of the PPO.

In Costabello et al. [74] , the authors opted to implement a strategy very close to the one implemented in Sacco et al. [73]. In this work, the authors describe the process of implementing Shi3ld, an access control solution for SPARQL-enabled RDF stores that provisions support for context-aware environments and helps the querying of linked data sources from mobile environments. Aside from this, the authors state that Shi3ld has been implemented in such a way that it is stackable in any SPARQL-enabled linked data platform. The authors continue and build on the work accomplished by Villata et al. [72] and Costabello et al. [6]. In Villata et al. [72], the authors analyze how the specified policies can be evaluated through the use of SPARQL ASK queries. Afterward, in Costabello et al. [6], the focus is the querying of RDF stores from mobile environments. The same vocabulary and evaluation strategy is used to build a system that applies AC in SPARQL queries received by an RDF store, a system that is later matured and named Shi3ld in this work. Thus, this work focuses on using the AC model, policy language, and the enforcement of such policies in the received SPARQL queries packed in user requests. Ultimately, Shi3ld uses query rewriting to enforce the AC rules to user requests. To determine the graphs to which the requesting user has rights to read or perform “insert” or “delete” operations, the system uses the saved SPARQL ASK queries of each access policy. These ASK queries are run against the context that is built for each request, and the answer returned after they are executed indicates whether the requester has access to the graph(s) protected by that policy. After this, the system rewrites the received SPARQL query and modifies it to make it query only the allowed graphs. This is accomplished by appending the clauses “FROM” or “From Named” in queries of type SELECT and “USING” or “Using Named” for queries of type Update, more specifically Insert, Insert Data, Delete, and Delete Data.

In Unbehauen et al. [85] , the authors present their approach to a system that provides access control mechanisms in SPARQL-enabled applications while following close the work accomplished in Costabello et al. [74], with the Shi3ld platform. In this work, the authors build on a

portion of the vocabulary used by Shi3ld, S4AC, to present an extended and modified vocabulary capable of representing user session data. Aside from this, their system diverges from Shi3ld's strategy of using SPARQL ASK queries to determine the accessible graphs and instead uses a single, more complex select query to reach the same goal. The authors justify this decision with the presumed overhead added by the individual ASK queries to request realization in an environment where there are a large number of policies and access conditions.

The authors then test three strategies for query rewriting to enforce access control on received SPARQL queries that are able to enforce access control at the level of the graphs by restricting what graphs are used upon the execution of a query. For the first strategy, the authors follow the same procedure as Costabello et al. [74] and use the "FROM" and "FROM NAMED" SPARQL operators, while for the second and third strategies they make use of the "EXISTS" and "FILTER" operators. The difference between the latter two is that in one case the accessible graphs are computed on initialization and the in other they are determined when a request is received. Additionally, the authors test a fourth method using Virtuoso's access control features where they translate and configure the access policies into the triplestore. Afterward, the authors perform an evaluation of the four strategies by simulating three scenarios using data produced by the Berlin SPARQL Benchmark [88, 89]. For this, the authors modified the data generation tool of the benchmark to make it generate three datasets of 10 million triples with different structures, and ran the benchmark using each implemented strategy.

Stojanov et al. [56] As for the work completed by Stojanov et al. [56], the authors describe the implementation of the Linked Data Authorization Platform (LDA Platform), a platform that offers protection for RDF stores. The authors implement their own policy language based on the structure of the SPARQL language or, more specifically, an extension to the syntax of SPARQL to make the access policies more readable and understandable. A policy in LDA language is defined as a 6-tuple that contains the following elements:

1. the permission of the policy, since it supports the definition of positive and negative policies through the keywords allow and deny
2. the action of the policy, among the set (read, insert, delete, modify) where "modify" is the conjunction of both insert and modify actions
3. a quad-pattern to select data to be protected
4. clauses to select data from the Intent, a term coined by the authors to describe the context of the request
5. clauses to restrict the data selected by the quad-pattern in (3)
6. a priority value used to resolve conflicts between policies

Afterward, the authors describe the enforcement of AC for each of the supported operations. This work follows a data-filtering methodology meaning that queries are executed against a filtered set

of data. Thus, all operations depend on a process the authors designate “allowed data extraction”. Using the quad-pattern and clauses stored by each policy, the LDA Platform completed this process by building a temporary dataset containing only allowed data for a given Intent. Additionally, the authors describe the algorithm used for insert and delete operations.

Benaribi et al. [76] present a work whose goal is slightly different from the goal of the works we have seen so far. While in the other works the presented solutions focused on not allowing users to perform operations or receive data that don’t have permission to, in this work, the authors aim to protect the data by encrypting the sensitive information that a given user is not allowed to see. Because of this, this approach is only suitable for read operations, providing no security for data update actions. The authors describe their approach to the specification and formalization of privacy policies by presenting a language based on SPARQL’s algebraic expressions. The authors believe that a language structured like this is easily understood by users unfamiliar with SPARQL and can be smoothly translated to SPARQL in order to enforce the specified policies. To perform such a task, the enforcement strategy is later described. For this, the authors implemented two separate modules, the Privacy Preserving Sensitive Data (P2SD) and the Disable P2SD (D-P2SD). The P2SD is responsible, at startup time, for reading the specified policies and encrypting the sensitive data using a stream encryption algorithm. For this, it uses the algebraic expressions in the policy language to build SPARQL queries to delete the data that needs to be hidden from the dataset and to replace this data with encrypted information. These queries hide both the property and the value of the triples in order not to provide information about the values being masked. Therefore, if the triple presented first in Listing 3.5 were to be considered sensitive data, it would be masked to something similar to the second triple presented in the same listing.

```
<foaf:John_Doe> <foaf:ssn> "87548958"

<foaf:John_Doe> <chup:pred1> "
b719eb03906ad0344c369ab60248ebba47bdbde0330999700234217c5d2b659"
```

Listing 3.5: Sample RDF triple representing a person’s social security number and the encrypted version of the same triple.

At the same time, the P2SD module builds a table with the corresponding unmasked properties and decryption keys for the protected values. The Disable P2SD is the module responsible for authenticating users and determining whether they may or may not access the encrypted information. Despite describing thoroughly how the data is protected, the authors don’t mention how the language specifies the association of the privacy policies with the requesting users and how the enforcement mechanism obtains that information. Nevertheless, when a user is found to be allowed to access sensitive information, the D-P2SD rewrites the request and uses the table of correspondence so that the masked properties and values are replaced with the original information. Ultimately, the result of this work provides a tool similar to Stardog’s sensitive properties, which

allows the definition of sensitive properties and masks the values that are linked with them so that disallowed users cannot see them.

3.4 Discussion

In this chapter, we present a thorough analysis of the current situation of access control over linked data platforms. Initially, we looked at the different parts that compose an access control system. As is defined originally in Kirrane et al. [25], a definition also adopted by us, the Access Control Model, the Policy Specification, and the Policy Enforcement form the Access Control Framework. With this, we presented the duties of each element of the ACF along with a number of alternatives of strategies and implementations that each of the elements can adopt when particularly focused on the context of Linked Data. For the ACM, we described a set of different models that included MAC, DAC, RBAC, VBAC, ABAC, and CBAC. For the Policy Specification, we looked at a few languages and strategies used to represent policies, and finally, for the Policy Enforcement, we detailed the approaches that can be followed to ensure that access control restrictions are enforced. Next, we completed an overview of the access control features and mechanisms equipped in a collection of RDF storage service providers. Here, we found that currently most triplestores provide the means to specify and enforce AC over the resources they hold. Nonetheless, as opposed to the results of some research works we later reviewed, the mechanisms provided comprise the implementation of very high-level AC features, usually limited to ACLs, or the basic AC models of MAC, DAC and RBAC. As outlined by the Table 3.1, most triplestores provide only protection of the entire server, or at the level of its datasets and graphs. From the set of analysed services, Stardog stands out since it provides basic protection at the level of the triple through its “sensitive properties” feature.

Finally, we performed a literature review of the research works completed in the field, where we found a heap of distinct approaches, which we present in Table 3.3. In most cases, these approaches provide, in some way, improvements relative to the protection currently offered by triplestores. Either by implementing a more intricate ACM, allowing the definition of policies using a richer form of policy representation, or by allowing data protection at a finer level, the work accomplished in the literature describes solutions that are more complete than the AC mechanisms of RDF stores. For example, despite only offering protection at the level of the graph as most triplestores do, the work of Costabello et al. [74] and previous work of Villata et al. [72] still comprise valuable contributions since they present the implementation of a CBAC model and the specification of a context enabled RDF vocabulary to be used as a policy language. Nevertheless, the most significant improvement the solutions implemented in the literature provide over the existing mechanisms on RDF stores is in terms of granularity of protection, since most works enable the specification and enforcement of more fine-grained access control policies.

In summary, through our survey, we were able to understand the structure of an access control system for linked data, and the different access control mechanisms present in both the existing RDF storing services and the implemented solutions in research work. These outcomes can be

used to aid the design and development of an access control solution suitable for a linked data context in the archival domain.

Research Work	Access Control Model	Policy Representation	Policy Enforcement
Hollenbach et al. [80] (2009)	MAC	ACLs	ACLs associated with RDF files
Reddivari et al. [81] (2007)	–	Prolog-like notation	–
Abel et al. [10] (2007)	CBAC	High-level policy language (RDF patterns/triples)	Query Rewriting
Dietzold and Auer [49] (2006)	VBAC (also DAC and CBAC)	RDF and SWRL rules	View Filtering
Gabillon and Letouzey [11] (2010)	VBAC	Implemented First-order logic language and SPARQL CONSTRUCT queries	View filtering
Franzoni et al. [82] (2007)	CBAC	XACML	Query Rewriting
Lopes et al. [83] (2012)	RBAC	Annotations in RDF graphs	Query rewriting
Kirrane et al. [12] (2013) and Kirrane [84] (2015)	DAC	–	Query Rewriting
Muhleisen et al. [79] (2010)	–	SWRL rules	View filtering
Sacco et al. [73] (2011)	CBAC	RDF vocabulary (PPO)	ASK queries, View filtering
Costabello et al. [74] (2012)	CBAC	RDF vocabularies	SPARQL ASK queries and query rewriting
Unbehauen et al. [85] (2016)	CBAC	RDF vocabularies	SPARQL SELECT queries and query rewriting
Stojanov et al. [56] (2018)	CBAC	Implemented SPARQL-based language	View Filtering (temporary graphs)
Benaribi et al. [76] (2021)	–	–	Previous Encryption of sensitive data and query rewriting

Table 3.3: Summary of features in works that implement access control for RDF and linked data systems.

Research Work	Protected Actions	Granularity	Conflict Resolution
Hollenbach et al. [80] (2009)	Read, Write, Delete	RDF files	–
Reddivari et al. [81] (2007)	Read, Write, Delete	RDF triple level	Default
Abel et al. [10] (2007)	Read	RDF triple level	Default
Dietzold and Auer [49] (2006)	Read, Write, Delete	RDF triple level	N/A
Gabillon and Letouzey [11] (2010)	Read, Write, Delete	RDF triple level	Default
Franzoni et al. [82] (2007)	Read, Write, Delete	RDF triple level, through class individuals	N/A
Lopes et al. [83] (2012)	Read	RDF triple level	–
Kirrane et al. [12] (2013) and Kirrane [84] (2015)	Read	RDF triple and graph level	Algorithm that considers policy explicity and specificity
Muhleisen et al. [79] (2010)	Read	RDF triple level	N/A
Sacco et al. [73] (2011)	Read	RDF triple and graph level	N/A
Costabello et al. [74] (2012)	Read, Write, Delete	RDF Graphs	N/A
Unbehauen et al. [85] (2016)	Read	RDF graph	N/A
Stojanov et al. [56] (2018)	Read, Write, Delete	RDF Quads	Priority values in policies
Benaribi et al. [76] (2021)	Read	RDF triple and Individual	–

Table 3.3: Summary of features in works that implement access control for RDF and linked data systems (*cont.*).

Chapter 4

Access Control Requirements for Linked Data and the Archival Context

As we have already discussed, the implementation of Linked Data principles and their usage in the construction of software systems bring many advantages. Linked Data ultimately builds a network of information that can be connected to external sources, which results in having additional details about a given subject and increases the quality, quantity, and relevancy of the accessible knowledge one can have about it. Nevertheless, with the rise of interest in the implementation of systems that follow these principles, new use cases have surged that require the system to hold data that is meant to be public and, at the same time, data that has access restrictions. That is to say that it is needed an effective way of performing access control over the data held by such systems. Aside from this, more recently, use cases describe systems that have become more dynamic. While in the early days of the Semantic Web, the focus was on providing systems that statically hold data, in the current days, this has been expanded to providing the means to not only see information from other sources but also to connect that information to the data a system holds, adding the possibility of updating, deleting and creating additional knowledge. These actions are associated with the same access control needs, given that such operations, in most cases, should not be available to everyone who can access the system. In summary, the implementation of dynamic linked data systems to cater to current use cases requires proper security mechanisms that control the user's access to information and the available operations they have over such data. Ultimately, such mechanisms capacitate systems in these environments to securely store private data and supply features for modifying current and producing additional knowledge.

This chapter provides an examination of the existing requirements for access control over linked data in order to enable us to design and implement a solution that fits the context of Archives. Initially, we analyze the generic requirements of access control over linked data platforms and system implementations and follow this analysis with an inspection of the specific requirements that the Archival domain mandates.

4.1 Access Control Requirements for Linked Data

The appearance of new use cases for Linked Data and the different access control needs which come attached to them has urged the Semantic Web community to make developments in defining what the access control requirements of a linked data system should be. While there have been works in the literature that have discussed this matter, something that, aside from the survey presented, Kirrane et al. [25] also features, we focused on the notes released by the W3C regarding this subject. The existence of many new use cases motivated the W3C to put together a Working Group, named Linked Data Platform (LDP) Working Group [44], to formalize a proposal with recommendations for implementing AC solutions for linked data systems. The results of the work accomplished by this Working Group were published in W3C [92] and are summarized in a note produced by the Working Group [93], where are specified the access control requirements for linked data based on a set of three generic use cases. These use cases represent different scenarios relative to access control on the access and manipulation of resources over HTTP, the management of Access Policies and rules over HTTP, and use cases of User Interface scenarios. Through them, the Working Group elicited the following list of requirements, which we transcribed and are as follows [93]:

1. An Agent must be able to authenticate herself to a server with an identifier or as the owner of a token **(GR1)**
2. Ability of an authorized agent to CREATE, EDIT, UPDATE relevant Access Control Graphs **(GR2)**, enabling:
 - (a) Ability to specify a collection of agents, identifying agents with URIs, URI patterns, or by description. **(GR2a)**
 - (b) Ability to specify a collection of resources – identified by URIs or URI patterns, or by description – with a specified access policy. **(GR2b)**
 - (c) Ability to connect a collection of agents with a collection of resources with given access privileges. **(GR2c)**
3. Ability to specify access privileges at a fine-grained level. **(GR3)**
4. Ability for a server to describe access control policies for a resource. **(GR4)**
5. Ability for a server to potentially describe reasons for access being disallowed in a machine-readable format. **(GR5)**
6. Ability for a user-agent to be able to find the Access Control Graph for a given resource. **(GR6)**

4.2 Access Control Requirements in the Context of Archives

For every system that is implemented to cater to a specific domain, there are requirements associated with that context, commonly referred to as business requirements, that influence the features that should be present in the system's implementation. Such requirements dictate the direction that the development of a project takes and are used to measure the project's success. While the results of this dissertation do not have a specific, singular use case to accommodate, it aims to generically fulfill the needs of the archival domain in a linked data context, and because of this, it is still necessary to understand those needs. Since this dissertation was developed under the scope of the EPISA project [94], we benefited from the project's resources to complete this task. We obtained the knowledge that enabled eliciting these requirements from informal testimonies of the researchers involved in the EPISA project and some project partners with roles directly related to managing digital systems from the Archival domain.

In this section, we describe the Archival domain, how the data of linked data systems in this context is arranged, and determine their access control needs.

4.2.1 The Archival Domain

Archives are organizations responsible for storing, managing, and curating extensive collections of historical records. These institutions have an essential role in our society since, through their records, they provide us with a connection to the past. As we mentioned earlier in this work, Archives represent the memory of nations, institutions, organizations, and much more and are reliable primary sources of information and provide firsthand facts, data, and evidence from letters, reports, notes, memos, and photographs [95].

The technological advances, along with the rise of the Web, have made digital archives a possibility, and they have become a regular part of public and private organizations. A lot of countries have developed their own versions of digital archives, and there are multiple cases of private institutions that have done the same. As the number of digital archives expanded, International Council on Archives (ICA) introduced international standards for the description of archival information for institutions to follow, such as ISAD(G) and DACS [96]. More recently, there have been developments in different conceptual models that aim to tackle the limitations of the current standards. Particularly in the context of the Semantic Web, there are cases that focus on the subject of archival description. Among the most well-known examples are CIDOC-CRM [97] and Records in Context (RiC) [98], which intend to provide a graph-oriented model capable of representing this kind of data. Further work has also been developed by the EPISA project that, through the ArchOnto ontology [99], builds on the results obtained by CIDOC-CRM and was an important source of knowledge for the development of this dissertation.

Throughout the rest of this section, we analyze the Access Control Requirements in the context of Archival digital systems.

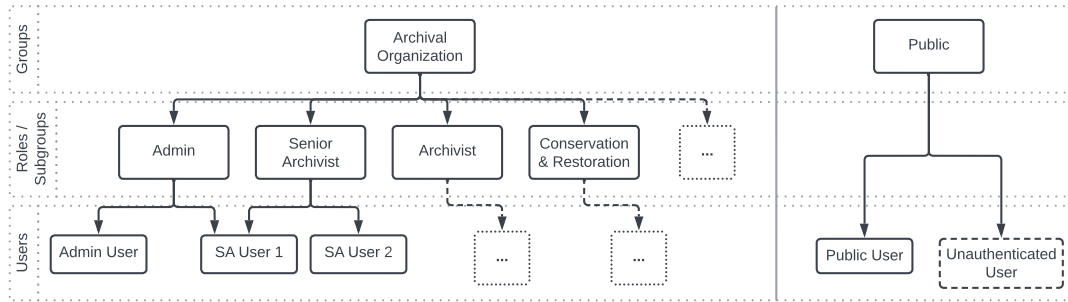


Figure 4.1: Organization of groups and subgroups in the Archival domain.

4.2.1.1 Subject and Action specification

The users of a system in archival institutions usually can be separated into two large groups: users that are part of the archival institution or users from outside the organization. For simplicity, we will refer to the former as institutional users and the latter as public users. These two groups have one significant difference from each other. While the public users can only have read access to data, the institutional users also have permission to perform update operations to insert new and edit or delete old information. Nevertheless, regardless of the group to which the user belongs, these actions are often limited to some extent. In the case of public users, there are usually limitations on the data they can access. These limitations can be in the form of small portions of documents or at the level of an extensive collection of documents. For institutional users, the limitations are even more fine-grained. Inside this group, there are smaller subgroups organized according to the role or function a given user has within the organization, which then have their respective specific restrictions. Aside from existing data access limitations, like what happens for public users, there are restrictions regarding the update operations for some subgroups of institutional users.

Thus, the representation of the structure of groups and subgroups from the Archival domain may be described similarly to what the diagram in Figure 4.1 depicts. In this figure is represented what the NIST standard defines as a Flat RBAC model, where the subjects can be assigned to many roles and a role can have many subjects [62]. In cases where subjects are inserted in multiple groups, the permissions of the groups are combined to determine the subjects' whole set of permissions. Despite the existence of these groups and subgroups, in the case of institutional users, in the Archival domain, there are also specific access requests granted only in particular scenarios that transcend the group in which the user is inserted. As we have seen, Archives are a trustworthy source of a wide variety of private and public information, often meant to be used and consulted by everyone. So, to fulfill this purpose, it is common to see temporary grants of access to private documents to public users, which comprises a case for a fine-grained access policy. Additionally, specific institutional users may have different access permission to their peers regarding both read and update operations when considering some contextual information. From our study based on the informal testimonies of partners of the EPISA project, we concluded that

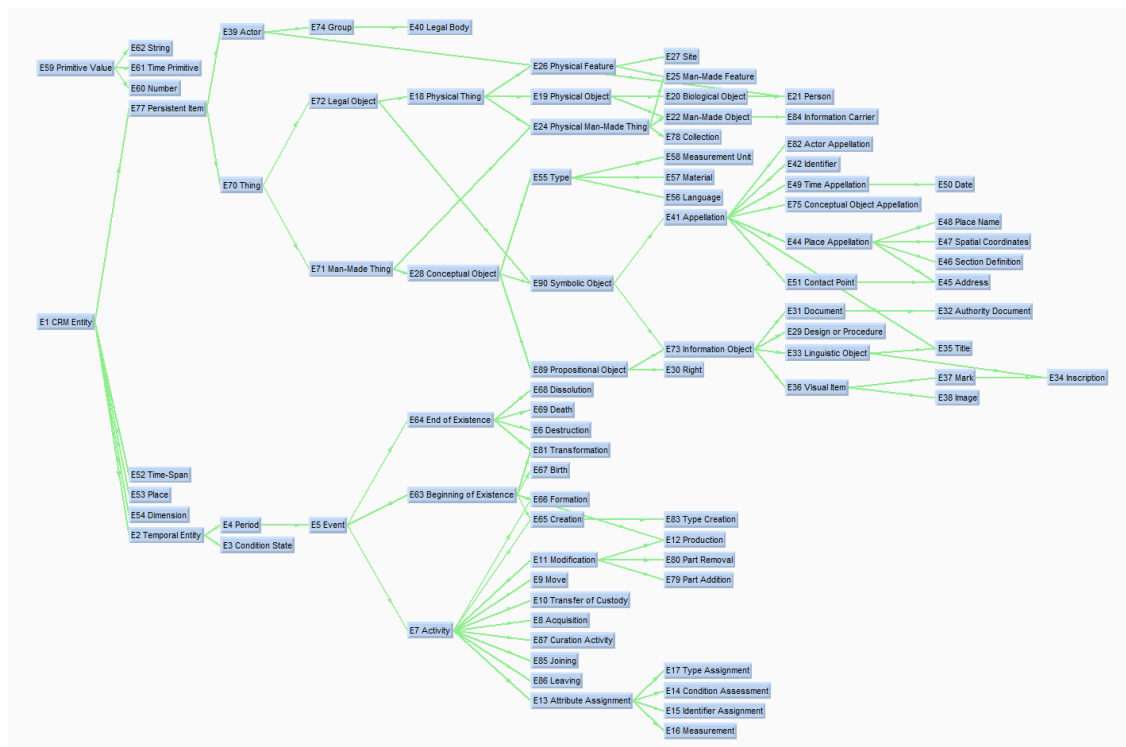


Figure 4.2: Entities represented in the CIDOC-CRM ontology [4].

these specific permissions are only affected by the user that performs the access requests and the time in which those same requests are completed.

4.2.1.2 Object specification

The digital representation of archival content gathers large units of information. As discussed above, procedures have been developed for the same purpose in the context of the Semantic Web, such as CIDOC-CRM, RiC, and the ArchOnto from the EPISA project. Since that archival information can be diverse and comprises data that is precise, profound, and finely described, ontologies and vocabularies representative of this domain are complex and have very rich semantics. As we can see by the number of distinct classes in Figure 4.2, which are relative to the CIDOC-CRM ontology and exemplifies the level of details of vocabularies of this field.

Archival data, when represented in a knowledge graph through the use of such ontologies, produces a graph with a large number of distinct individuals, links, and concepts. Unlike other older standards of description, characterized by a hierarchical structure that gathers information in a document-styled method, the representation of this data in a knowledge graph means that the information is atomized and expanded into much smaller units. While in hierarchical models, the information is aggregated in documents and then may be organized in smaller sections, in an equivalent graph in RDF format, this structure is lost and all information is represented in the form of triples. Thus, a document, a section of a document, or a group of documents are represented by a set of triples that comprise a subsection of the graph.

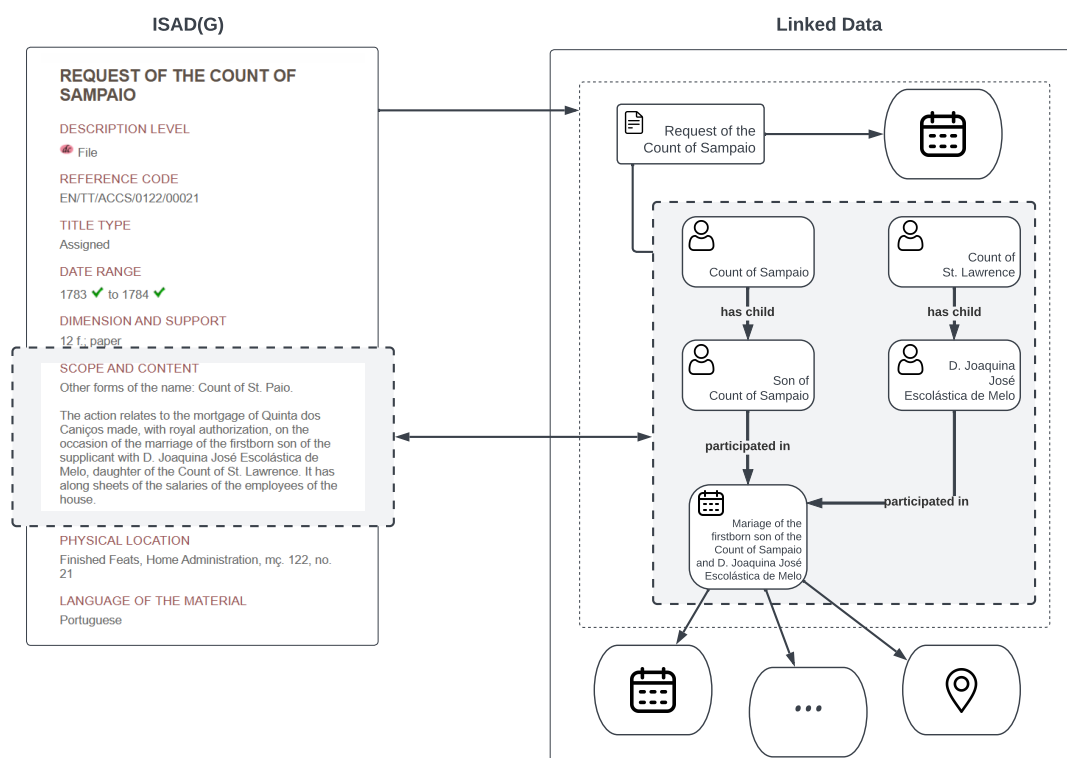


Figure 4.3: Informal example of the transformation of information from a hierarchical-based model to a graph-based model [5].

In regards to access control, in the hierarchical model, this means that subjects may not be able to access a part of that structure, that is, a document, a group of documents, or a section of given document. In a graph model approach, this amounts to a subject not being able to access a part of the graph, more formally, a subset of nodes and edges that describe a specific matter. In Figure 4.3, this is depicted with an informal example of the representation of information in both an existing hierarchical model, ISAD(G), and in a graph structure. As it can be noted, restricting the access to a section of information of a document, on the left comprises only the omission of that section, whereas on the right it comprises the omission of a part of a graph structure. Therefore, looking at the access types according to the different types of users, we can see that an RDF graph containing information Archival information may not be entirely accessible. So, in terms of the policies' object specification, the Archival domain demands a strategy capable of selecting sets of triples, subsets of a larger RDF graph.

To summarize, regarding the aspect of access control over linked data, the archival domain mandates a set of characteristics. In relation to the specification of the subjects of Access Control Policies, it is required a description that can be based on subjects that are part of a determined group or have a particular role, specific subjects commonly referred to as “users”, or based on the evaluation of the time conditions in which the access request is made. These requisites add to the requirement presented by the LDP Working Group in GR2a. Additionally, there are different

actions available for a subject to perform, something that is already specified by **GR3**. Furthermore, since the data to be protected in this context is grouped in sets of triples, it is required a fine-grained level of selection of objects, which the generic requirements also partially request through the **GR2b**

Finally, as a result of this analysis, we were able to elicit the requirements listed below.

1. Ability to specify access restrictions according to group or role of a subject (**SR1**)
2. Ability to specify access restrictions to specific subjects (**SR2**)
3. Ability to specify access restrictions that consider the time of access requests (**SR3**)
4. Ability to specify restrictions for the different actions that regard viewing and curating the contents of the archival system (reading, inserting, deleting) (**SR4**)
5. Ability to specify access restrictions at a fine-grained level, at the level of the resource and triple; permissions to partial graphs, parts of an RDF graph (**SR5**)

Chapter 5

Access Control for the Archival Domain

In Chapter 3, we evaluated research works and the mechanisms presented by the RDF storing services. If we analyze the requirements elicited in the previous chapter more specifically, the ones mandated by the Archival context, we can see that none of the services inspected is capable of fulfilling them all. While most of them can represent and apply an RBAC model, none considers contextual data, nor is it able to enforce access control at the level required by the Archival domain, given that all support protection up to the level of RDF graphs. The exception to this is Stardog, with its promising feature of sensitive properties; however, the fact that they have to be configured at startup makes it nonviable. Additionally, the results produced by the research works reviews either do not provide ready-to-use solutions, or, the ones that do, are not capable of satisfying the demands of the archival domain. Therefore, we are required to implement our solution for the problem.

In this chapter, we present the design of an AC solution that fits the Archival domain, the decisions made, and the implementation details of the solution developed. To do so, we look at the requirements for access control for linked data according to the W3C and the needs of a system within the context of Archives listed in the previous chapter and evaluate the strategies used in the literature for tackling similar problems. The chapter is divided into two sections: the first is where we design and conceptualize the framework of our solution, presenting the elected Access Control Model and the chosen strategies for Policy specification and enforcement; the second is where the details of the implementation are provided.

5.1 A Solution for the Archival Domain

5.1.1 Access Control Model

As described in the previous chapter, a linked data system from the Archival Domain has specific needs in terms of security. Such requirements are decisive for the determination of what is the right ACM to implement when building an access control system to protect it. Through the

knowledge obtained from the EPISA project regarding the Archival domain, we were able to elicit the requirements that an access control solution for this context must fulfill. In summary, we determined that, in a system for the Archival domain, users can be part of one of two major groups, the group composed of users affiliated with the Archival organization, provider of the system whose access needs to be managed, and the group composed of users from outside of the organization, commonly referred to as the general public. The two groups have different access conditions, the general public is usually allowed to perform read-only operations. Moreover, the group that is composed of users that have a role inside the organization is sometimes divisible into subgroups as some of its elements have different permissions from each other. Aside from this, individual users, independently from the group they belong to, may have specific access to resources and be allowed to execute a determined set of actions over them. To finalize, access conditions may be altered by the conditions of the access request's environment, such as the time in which the request is performed.

From the different ACM types covered in Chapter 3, and the requirements outlined above, in Chapter 4, we can understand that this scenario does not require a specific Access Control Model but a combination of the features of multiple Models, in particular the Role-Based Access Control Model (RBAC) and the Context-Based Access Control Model (CBAC). Due to the fact that, in a linked data system from the Archival domain, users can be either from outside the organization or be affiliated with the Archiving organization and have a distinct role or group of roles, it is clear that an AC system needs to feature mechanisms from the RBAC model. At the same time, given that some actions may be restricted because of the conditions (time) in which the request is made, the features present in the CBAC model are also required.

5.1.2 Policy Specification

The Policy Specification is relative to how the access policies are represented within the system. This specification is the modeling of the Access Control Model chosen and the source of information that the Policy Enforcement module must use to exercise the access policy over the protected data. The only requirement for the strategy of this specification is that it is capable of representing the ACM detailed in the previous subsection. Nevertheless, when deciding what specification strategy to follow, some other aspects need to be factored in. Aspects such as the results obtained in similar works, whether a determined strategy hinders any later stage of the procedure, or if eventually increments the effort associated with implementation are all points to consider.

To determine the correct approach to follow, we analyzed the requirements mandated by the solution we are aiming to develop, described in the previous chapter, the needs of the ACM presented above, and the strategies followed by the research works reviewed, present in Chapter 3. Looking at the strategies implemented by these works, we can see that some of them are not feasible for the Archival context or the time frame of this work. Using annotations, like in Lopes et al. [83], is not viable since it inhibits the representation of context information. Developing a specialized access control language, like the one Stojanov et al. [56] constructed, requires an effort that may hamper the development of other components of this work and, since that there it would

provide no additional features or advantages when compared to existing options, it is not a desirable approach. On the contrary, we also gathered a set of solutions that we considered to be valid options for our solution. XACML is one of these options, given that it is an OASIS standard and has been actively maintained by this community, and provides expressiveness for context-aware scenarios. Despite not having by default the means to represent RBAC, there have been works that focus on extending it with such features in Ferrini and Bertino [55] so, due to these reasons, using XACML as a policy language was thoughtfully considered. However, there are not many software modules with easy integration with other projects to handle the reading, writing, and evaluation of policies specified in such language, and the ones that exist do not seem to be capable of fulfilling the requirements of the Archival domain for an RDF scenario. For example, W2SO's¹ identity server, does not provide means to develop procedures capable of enforcing AC over RDF data with the desired granularity. Because of this, the option of using XACML appeared to be unpractical. Aside from XACML, using OWL, and consequently RDF, for representing Access Policies for linked data environments is a common approach. As we have seen, many research works have followed this path to obtain means for specifying access control constraints.

After evaluating all options, we concluded that the most adequate approach to include in our AC solution was to follow the strategy of using RDF along with a vocabulary to specify the policies. Ultimately, we opted to follow a strategy identical to the one Costabello et al. [6] used when building the Shi3ld platform. We considered the vocabularies developed and used in this work very suitable for our scenario since they comprise a coherent context-aware vocabulary. Additionally, its implementation is based on Semantic Web technologies and constitutes a methodology that can be easily extended both now to cater to the additional needs the Archival context possesses and in the future. Moreover, Costabello et al. present a very attainable strategy for evaluating context information using SPARQL ASK queries, something that will be discussed further, and that, along with the deficiencies of the remaining options also played a part in the decision of selecting this strategy.

In summary, the elected strategy was similar to the one implemented in the work developed by Costabello et al. with the production of the Shi3ld platform. Furthermore, we built on the work developed on the previous works of the same authors and used the vocabularies developed in them, S4AC and PRISSMA. The S4AC vocabulary is an RDF-S vocabulary that can be used to describe Access Control Policies over RDF resources. Figure 5.1 depicts the main concepts defined by this vocabulary. In it, we can see the composition of an Access Policy as the core concept by being linked to a set of Access Conditions, an Access Privilege, and the resources it protects. In Costabello et al. and on the associated webpage [7], we can see further details about each of the concepts and their respective properties. The PRISSMA vocabulary is an RDF-S vocabulary that can be used to describe context conditions by default but not exclusively for mobile environments, given that it can be applied to other environments. Figure 5.2 summarizes the concepts PRISSMA is capable of representing, along with how they relate to each other and to external vocabulary. Like S4AC, PRISSMA is described in finer detail in Costabello et al. and in its official website [8].

¹W2SO, <https://wso2.com/>

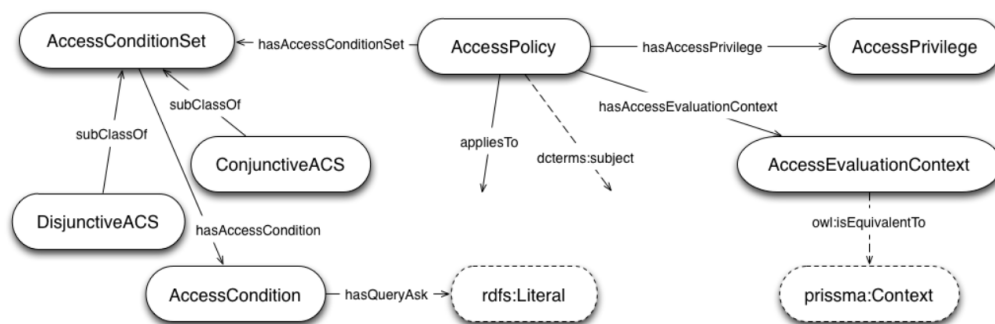


Figure 5.1: Summary of S4AC vocabulary [6, 7].

Despite almost fulfilling all requirements mandated by the ACM described and the needs of the Archival context, S4AC and PRISMA do not consider some aspects necessary for the domain of archives. So, it was required to extend them in a way that made them fit our system and fulfill the needed requirements.

5.1.2.1 Extensions to the S4AC and PRISMA vocabularies

S4AC and PRISMA are the vocabularies presented and used in works Costabello et al. [74] and Costabello et al. [6], and were developed for describing access control for linked data in context-aware scenarios, a scenario that, as we have already seen, satisfies one of the requirements that the context of archives demands. However, one thing that is crucial to the domain of archives that these vocabularies do not contemplate is the representation of roles. In Archival organizations and their software systems, roles represent the function of its users within the organization or, if they are not part of it, their class as an external user. Thus, we added the modeling of this concept to the User class, present in the PRISMA vocabulary. Aside from this, since that S4AC is not prepared to accept both positive and negative typed Access Policies, we also modeled a small extension to give the vocabulary this feature that empowers it with additional expressiveness. These complements were modeled and appended as illustrated in Figures 5.3 and 5.4.

Furthermore, some other additions joined to the S4AC and PRISMA occurred at the level of Data properties. In order to complete the vocabulary, we added some data properties either because they were useful for the implementation or to answer to requirements of the solution being implemented. The added properties are detailed in Table 5.1.

While we included some new concepts and made some modifications to the vocabulary, some of the concepts of S4AC and PRISMA were not used. From S4AC, the Access Evaluation Context class and the property that links it to an Access Policy were overlooked due to a minor difference in the enforcement strategy to be discussed later, whereas from PRISMA, Device, and Location concepts were left out since the Archival domain does not need to evaluate such aspects to determine whether an access request should or should not be granted. Therefore, considering all additions and omissions, the vocabulary used for the specification of Access Control Policies can be summarized by Figure 5.5.

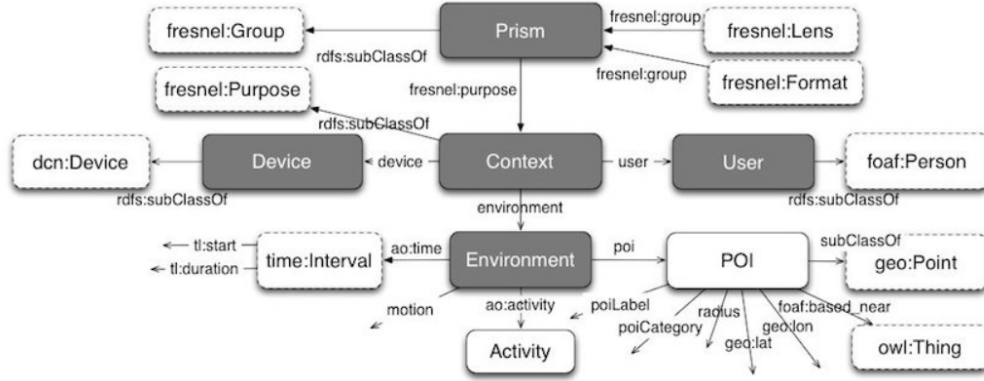


Figure 5.2: Summary of PRISSMA vocabulary [6, 8].

5.1.3 Policy Enforcement

The last piece of the Access Control Framework is the policy enforcement strategy which is responsible for the implementation of a procedure that guarantees the fulfillment of the rules modeled by the ACM and specified in the framework’s policy language. As detailed above, after inspecting the Table 3.3, we concluded that only the work by Stojanov et al. [56] was capable of fulfilling the specific requirements for the Archival domain that we outlined in Chapter 4. That is a context-aware environment, with the possibility of adding the comprehension of roles and subject groups and capable of expressing and controlling the access at the level of the triple and triple patterns. Nevertheless, we considered the strategies used by all of the analyzed works since some could be adaptable to our scenario. We partitioned the enforcement process into two parts. The first comprises the evaluation of the context of the request. And a second that is responsible for restricting, allowing, or denying the access or the execution of the operations requested.

5.1.3.1 Evaluation of Context

The evaluation of context information is guaranteed by the specification ASK queries defined in each one of the Access Conditions present in an Access Policy’s Access Condition Set. By taking advantage of the format of specification of the policies, that is, RDF format, and the SPARQL endpoint the storage point provides, we can use ASK queries to run against the representation of a request’s context. This strategy is followed by Costabello et al. [74] and Sacco and Passant [71]. Unbehauen et al. [85] have a similar approach using SELECT queries. Although the authors in this work deviate from the usage of ASK queries to the detriment of more complex SELECT queries, we found no evidence that the execution of the same number of SELECT queries is consistently faster than the ASK queries. While the authors combine multiple queries into one, with the intent of saving time in the communication phase, for now, we settled to keep the separate execution of these queries in order to collect information needed for the second stage of the enforcement process in a more straightforward way.

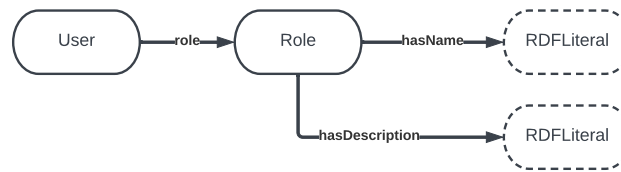


Figure 5.3: Addition of Role concept to PRISSMA vocabulary.

An Access Policy has an Access Condition Set which in turn has a group of Access Conditions. This Set can either be “Disjunctive” or “Conjunctive”. In the case of the former, only one of its containing Access Conditions has to be validated for the policy to be activated, whereas, in the case of the latter, all of them need to be validated for the same result to happen. This rationale is equivalent to the logical operators “AND” and “OR”.

The validation of Access Conditions through the use of ASK queries happens as follows. After the creation of context like the one depicted in Figure 5.6, the ASK queries that pertain to the Access Conditions associated with a given Access Policy are run against it. If the result is “true”, then that Access Condition is validated in that context; otherwise, it means that the requisites of that Condition are not met. Then, with the results of the validation process, the final validation of the Access Policy is calculated using one of the operators mentioned above, according to the type of the policy’s Access Condition Set. For example, if we consider an Access Condition Set with two conditions represented by the SPARQL ASK queries from Listing 5.1 and run them against the context presented in Figure 5.6, we will obtain two different responses. Since the first one attempts to see if the requesting user is attributed with the role with the name "Archivist" and if the request was performed before the 31st of November of 2022, its result will be “false”. For the execution of the second query, the response will be positive, given that it only checks if the user is attributed with the "Admin" role. Thus, this context will be approved depending on the type of the Access Condition Set: if it is a disjunctive ACS the context is valid as a result of the operation “false OR true”; if it is a conjunctive ACS the context is invalid as a result of the operation “false AND true”.

```

1  ASK WHERE {
2    ?context a Context;
3    user ?user .
4    ?user role ?role .
5    ?role hasName "Archivist" .
6    ?context environment ?environment .
7    ?environment time ?time
8    FILTER(?time < "2022-11-31T23:57:00"^^xsd:dateTime)
9  }
10
11 ASK WHERE {
12   ?context a Context;

```

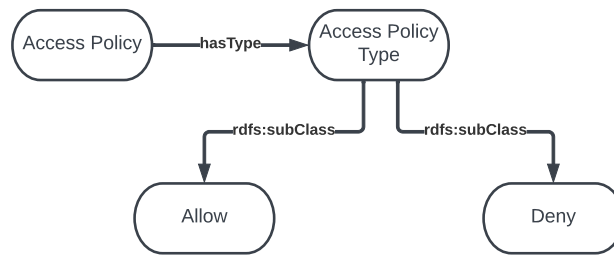


Figure 5.4: Addition of Access Policy Type concept to S4AC vocabulary.

```

13   user ?user .
14   ?user role ?role .
15   ?role hasName "Admin"
16 }

```

Listing 5.1: Sample ask context SPARQL queries.

5.1.3.2 Enforcement Strategy

While until this point we have implemented a strategy of specification and context evaluation similar to Costabello et al. [74] and Sacco and Passant [71], since these works aim to provide access control mechanisms at the level of RDF graphs, we could not do the same for the enforcement of policies. Driven by the need to block access to a set of triples present in an RDF graph, it was needed a different approach to ensure that the specification of policy objects had the granularity demanded by the Archival domain's requirements.

As we detailed in Chapter 3, there are two main strategies to enforce access control in RDF at a fine level which are the processes of view filtering and query rewriting. After evaluating both techniques and how each was implemented in the research works reviewed, we elected to follow an approach based on the former. This decision was primarily due to a string of reasons related to the inadequacy of the query rewriting procedure. Since that query rewriting involves applying modifications to the requests received, it brings a set of difficulties and disadvantages. First off, it is a technique that is subordinate to the format of the request. This means that different formats of requests require different implementations. For example, as we have already seen, the techniques implemented in [10] can only be applied to SeRQL, other types of requests would require a different implementation. Secondly, this type of strategy is not trivial to implement in a generic form making its implementation difficult to reuse. Most implementations are meant to cater to a specific vocabulary. Aside from this, they detail cases of policies more focused only on the semantics of the ontology, policies like "block all individuals of type X class". The scenario we want to serve has data-driven needs that comprise more complex and expressive policies to enable the protection of a specific set of triples. Additionally, some ontologies like ArchOnto can be very large, which, together with a scenario that requires access policies at the level of an aggregation of data,

Class	Data property	Description
AccessPolicy	hasName	Associates the Access Policy with a xsd:string that represents the policy's name
	hasDescription	Associates the Access Policy with a xsd:string that represents the policy's description
AccessCondition	fromTime	Associates the Access Condition with a xsd:dateTime, that represents the start time present in its validating ASK query
	toTime	Associates the Access Condition with a xsd:dateTime, that represents the end time present in its validating ASK query
	user	Associates the Access Condition with a user present in its validating ASK query
	role	Associates the Access Condition with a role present in its validating ASK query
	hasQueryAsk	Associates the Access Condition with a xsd:string that represents the condition's validating ASK query
User	hasUsername	Associates the user with a xsd:string value, used to store his username
	hasPassword	Associates the user with a xsd:string value, used to store his password
Environment	time	Associates the environment with a xsd:dateTime value

Table 5.1: Data properties added to the S4AC and PRISSMA vocabularies.

causes the number of clauses added to be very large, resulting in increased query response times. Furthermore, this strategy is not easy to implement for update operations when considering a protection granularity at the level of the triple. Despite existing implementations that contemplate this issue in Kirrane [84], we are not certain about their effectiveness and correctness in certain cases. Finally, while the implementation of view filtering may trigger concerns regarding performance times, as we could anticipate and as it is outlined by Stojanov et al. [56], according to the evaluations completed by Abel et al. [10] and Kirrane [84], query rewriting also causes troubles in that field.

So, for these reasons, and due to the successful outcome achieved by Stojanov et al. [56] in a similar scenario, one where it was required protection with the same level of granularity not only for operations of read but also for update operations, we decided to follow a similar strategy to the one presented in that work. This strategy incorporates the creation of a temporary context-specific view, which is later needed for the protection of all types of operations. When compared with the strategy of this work, our approach diverges mainly in three key aspects. Initially, for the specifi-

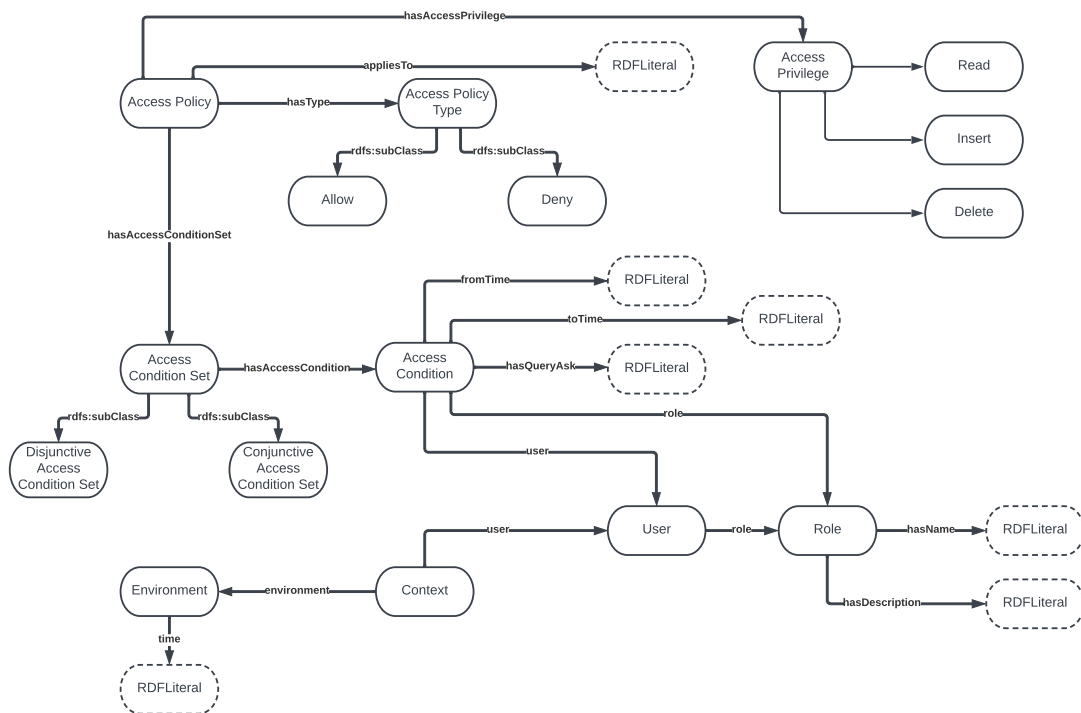


Figure 5.5: Summary of the AD4LDA vocabulary.

cation of the policy’s object, the data to be protected, we elected to use SPARQL CONSTRUCT queries, a technique already proven by Dietzold and Auer [49], that differs from the specification through graph patterns that the authors in Stojanov et al. [56] follow. Another aspect from which we deviate lies in the protection strategies implemented. In Stojanov et al. [56], for all types of operations, the authors decided to provide the options of allowing and denying the partial execution of operations, which means that it is added another factor into the decision of allowing or denying a request. Therefore, there are two alternatives of the algorithm to follow when a request is received: if the partial execution is allowed, then read requests are performed in the created view, and update requests modify the data that the subject in that context is allowed to modify if the partial execution is not allowed, then the requests received are executed both in a protected and unprotected environment, and are only completed if the outcome is the same in both environments. For our scenario, we felt that providing these two alternatives was important for update operations but negligible for read operations, so we overlooked this aspect. However, what we consider to be partial execution of operations is different from the approach of the authors. While they consider it to be the execution of operations that have a different effect on the protected and unprotected set of data, we consider the partial execution an operation that cannot be completely fulfilled due to being disallowed by the Access Policies. Finally, the matter of conflict resolution was another aspect that we followed a different path, since we resolved to use a defaulting technique, prevalent in the works analyzed, contrarily to the usage of priority values by Stojanov et al..

The creation of the temporary view is, more formally, the computation of the set of triples that

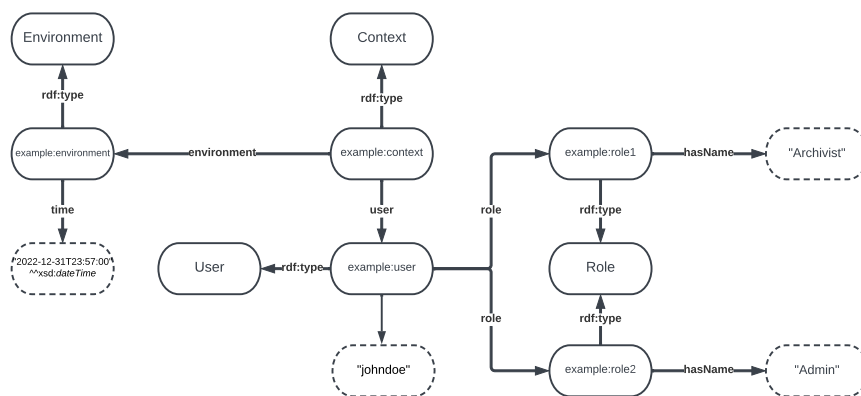


Figure 5.6: Sample of RDF graph containing a request context.

the subject that performs the request can access. Therefore, this is done by evaluating and enforcing all policies that are associated with the “Read” Access Privilege through the implementation of the algorithm presented in Listing A.1. This algorithm iterates through the policies relative to the “Read” Access Privilege and, after positively evaluating that the policy is applicable to the current context through the process earlier described, does one of two things depending on the sign of the policy. If the policy is of type “allow”, the triples selected by the policy’s CONSTRUCT query are added to the allowed set, or else if the policy is of type “deny”, the selected triples are added to the set marked as denied. Then, with these two sets, and since we are implementing a conflict resolution strategy defaulting towards the denial of access and that “deny” policies have a higher priority, the set of denied triples is subtracted from the set of allowed triples to obtain the final allowed data view. After this process, the protection of the different operations happens as follows.

Protection of Read operations

“Read” operations are operations that are used to retrieve information from a triplestore. For example, as we have seen in the case of SPARQL, these operations comprise the query forms of SELECT, DESCRIBE, CONSTRUCT, and ASK. Since that, as mentioned above, we elected to allow by default the access of partial data to guarantee the protection of these operations the process is simple. Allowing access to partial data means that it is only needed that the requested operation is executed against the previously created context-specific view of allowed data. Given that the creation of this temporary view is performed with the constraints applied to the READ Access Privilege, it contains only the subset of data that the requesting subject can access. For this reason, the execution of read operations in this environment is intrinsically secure.

Protection of Update operations

Update operations comprise the operations that modify the data stored in a system’s database. When compared with previous, the protection of these operations is more complex. While the

protection of read operation comprised only its execution in the environment of allowed data, the protection of update operations, even though it also relies on the execution in this environment, requires additional procedures to be enforced. These operations are run against the allowed data model, and, through this, is created a modified data model. Then, the enforcement process is started with the determination of what modifications resulted from the previous operation execution. The logic for this varies according to the operation requested. For insert operations, the model of allowed data is subtracted from the modified model since this calculation results in the set of triples that were added by the executed operation. In the case of delete operations, the operands of this calculation switch position so that it results in a model containing the set of triples that were removed. Afterward, the process that iterates through and enforces the existing policies is started. For this process are created two additional models marked as “allowed” and “denied”. Then, the policies with the same Access Privilege as the request received are iterated, and their respective CONSTRUCT queries are run against a set of triples. In the case of insert operations, this set of triples is the one after the modifications are applied whereas in the case of delete operations it is the one before the modifications are applied. Finally, if the result of this query intersects with the set of inserted or removed triples, that intersection is then added to one of the created models depending on the type of the Access Policy: if it is a positive policy, the triples are added to the “allowed” model; if it is a negative policy the triples are added to the “denied” model.

At last, the set of allowed triples is added or removed permanently to the temporary view and the original graph of the data. This action occurs immediately if the execution of partial operations is allowed; otherwise, before the modification is completed, it is verified if the set of denied triples is empty and if all of the modified triples are present in the set of allowed triples. One important thing to note is that the requested update operation is not actually run on the original graph since its execution in a different context may result in the unwanted modification of a larger set of triples. The formalization of this process is presented in listings A.2 and A.3.

5.2 The AC4LDA Platform

This section and the rest of the present chapter detail the implementation of a platform that integrates the framework presented above. This platform, comprehensibly named AC4LDA (Access Control for Linked Data Archives), comprises two components, a Java package for the Spring² framework that implements the strategies determined in the previous section and provides interfaces for enabling and managing access control and a web application that works with one of such interfaces and aids the administration and control of the access Policies of a linked data system. The overall architecture of the platform can be seen in Figure 5.7. This figure depicts a client-server architecture between the Admin Interface and the Spring Boot [100] application maintained by the REST API presented by the AC4LDA Package. The Package then communicates with Fuseki [58] through a Java API using the HTTP protocol for communication and

²Spring, <https://spring.io/>

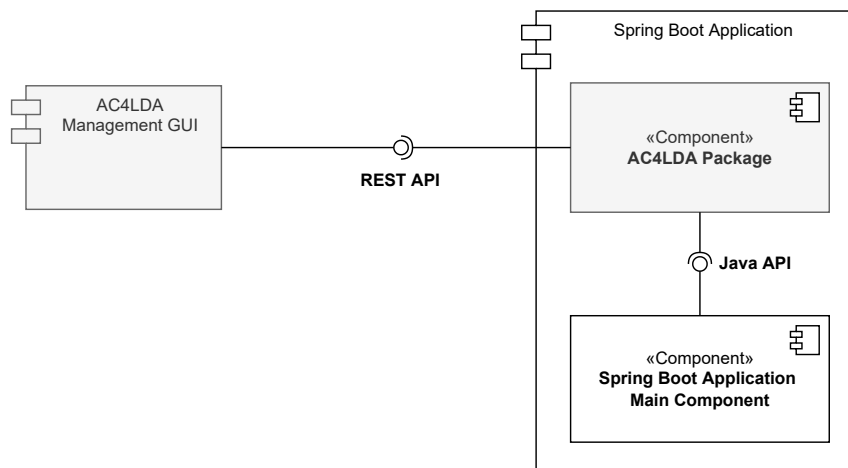


Figure 5.7: Architecture of the AC4LDA Platform.

presents a Java API to the host Spring Boot application. These components and the communication between them are detailed further in this section which is partitioned into two subsections: the first features the development of the AC4LDA package, and the second presents and describes the implementation of the AC4LDA Admin Interface. In each of them are presented details pertaining to the design, implementation, and technologies used in the respective software artifacts, along with their features and process for its integration.

5.2.1 The AC4LDA Package

The AC4LDA Package is the component where the Access Control Framework designed and adapted to the Archival domain is implemented. This Access Control Framework is described in the previous section and consists of a model and set of strategies that ensure that the requirements of access control for linked data, particularly in the context of Archives, are fulfilled. Its implementation relies on two factors: having a point of storage of data and a setting to implement the mechanisms proposed. For the storage of data, given that the policy language is based on an RDF vocabulary and that the enforcement strategy requires the usage of SPARQL, the system requires a SPARQL-enabled triplestore. Regarding the setting for its implementation, for it to be possible to implement the enforcement methods detailed by the Access Control Framework, it is needed an environment where operations like querying, updating, and performing operations in RDF triple store services are possible, along with the calculation of set operations with RDF graphs such as union, intersection, and difference. Additionally, considering the use cases listed by the LDP Working Group, we can perceive that they describe platforms that support the HTTP protocol [93], a characteristic that we desire to maintain for our solution. For this reason, the implementation environment needs to be empowered with additional capabilities so that the resulting platform is able to serve functionalities over this protocol.

After looking at these aspects and deliberation, we decided to proceed using Fuseki, from the Apache Jena framework [101], as the triplestore to store the data and the Java web framework

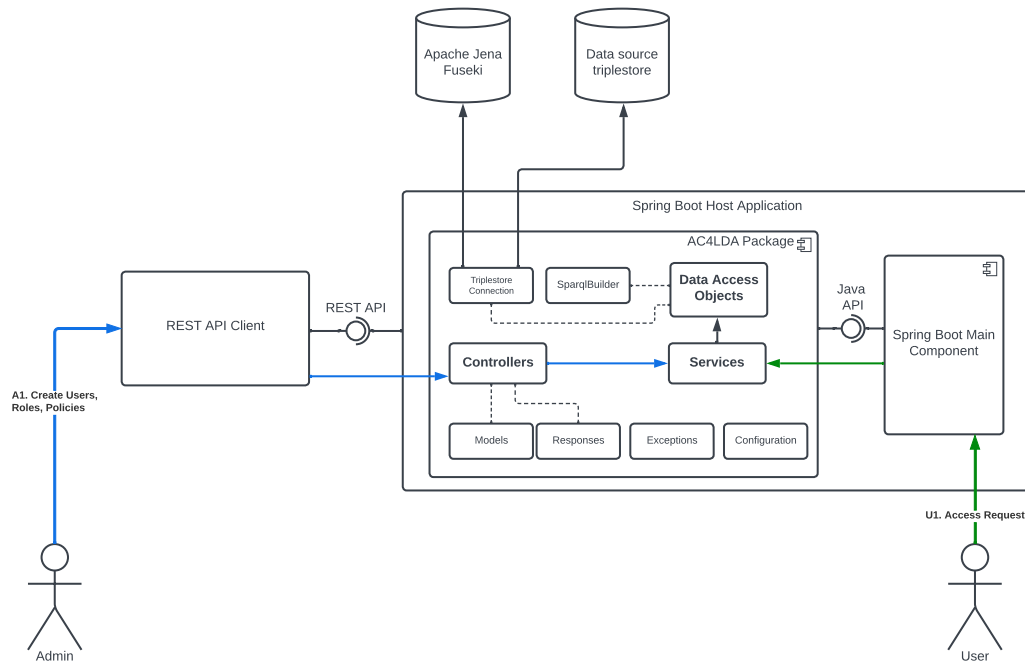


Figure 5.8: Architecture of the AC4LDA Package.

Spring Boot as the framework for the implementation of the AC4LDA module. The reasoning behind this decision lies in Fuseki being a widespread solution, as we can see by Figure 3.1, presented in Chapter 3, and coming from an open-source project that contains useful features like support for reasoning and a graphical user interface that eases the configuration, debugging and testing of our strategies. Regarding the usage of Spring Boot, we elected it as the framework to use since that, like Fuseki, is a well-adopted framework for the construction of Web services, and due to the support and quantity of libraries it has available. Besides this, we already had some familiarity with its host language, Java, which itself possesses high compatibility and support of APIs like the ones provided by the Apache Jena project. The EPISA project's technology stack already included systems implemented using both the Apache Jena Fuseki triplestore service and the Java web framework Spring Boot. In summary, the AC4LDA package is a library for the Spring Boot framework bundled as a Maven package that can be easily integrated into other applications to manage access control in RDF sources. For that, the package provides REST and Java APIs to administer the creation of policies for such sources and to enforce those rules to a set of available SPARQL operations.

5.2.1.1 System Architecture

The Spring Boot framework does not enforce or recommend any particular pattern for implementations and provides the developer with the freedom to elect how the system is structured. Nevertheless, among several sources on the Web, we found that an ordinary and appropriate approach is to divide the into three layers: Controllers, Services and Data Access Objects. So we

opted to follow a structure based on this method. These three layers are meant to ensure some separation of duty and are responsible for tasks at different levels. The Data Access Objects are responsible for handling the access to the triplestore for functions such as reading, creating, and deleting data. The Services are accountable for handling the logic of the multiple parts of the and for providing the implemented features to users through the aforementioned Java API and the controllers, which serve as the interface with external HTTP interactions through a REST API. The interaction of these components can be perceived in Figure 5.8, where are shown the two possible types of interaction with the AC4LDA Package through both APIs. Aside from this, that figure presents other sub-modules of the AC4LDA Package that aid the execution of the central components. The Controllers resort to two modules, one named “Responses” for building the responses of the REST API and another named “Models” for modeling the structure of the JSON objects present in the requests and responses of the same API. The Data Access Objects also make use of two additional modules to aid the interaction with the RDF storage services. The first named “SparqlBuilder”, is the module responsible for the construction of the needed SPARQL operations, whereas the second “Triplestore Connection” deals with the communication with the triplestores. These modules make use of two libraries, the RDF4J SPARQLBuilder [102] and the Apache Jena Java APIs, respectively, which are detailed below. Their purpose can be observed in the different stages present in Figure 5.9, which depicts the stages and sequence of operations of the implementation of the enforcement process we designed and described in Subsection 5.1.3, present in the implementation of the AC4LDA package.

In stages (1) and (2), that are relative to the evaluation of context, is where the construction of the context associated with the request received and the subsequent calculation of applicable policies occurs. Here, the “SparqlBuilder” module is used to build the needed queries and update operations for retrieving information, creating the context graph, and evaluating the access conditions of each policy. These operations are then executed through the “Triplestore Connection” module. Stages (3) and (4) concern the concrete enforcement strategy. In stage (3) the construction of the temporary view is completed through the execution of the SPARQL CONSTRUCT queries associated with the access policies collected in the previous stage, using the “Triplestore Connection” module. This module is then used once more to either complete the execution of a requested read operation or, in case it was requested an update, in the procedures of stage (4) that relate to the application of the enforcement mechanisms for these operations.

RDF4J SPARQLBuilder

The execution of SPARQL queries is dependent on the RDF4J SPARQLBuilder library, a library that is used for building queries. In our implementation, the execution of SPARQL queries is required in different parts of the processes. Particularly in activities related to the administration process, SPARQL queries are needed to read, insert and delete the entities present on Access Policies, Users, and Roles, to serve the features that the implemented REST API provides. So, we used this library in order to easily, safely, and correctly build them. In Listing 5.2 is presented the

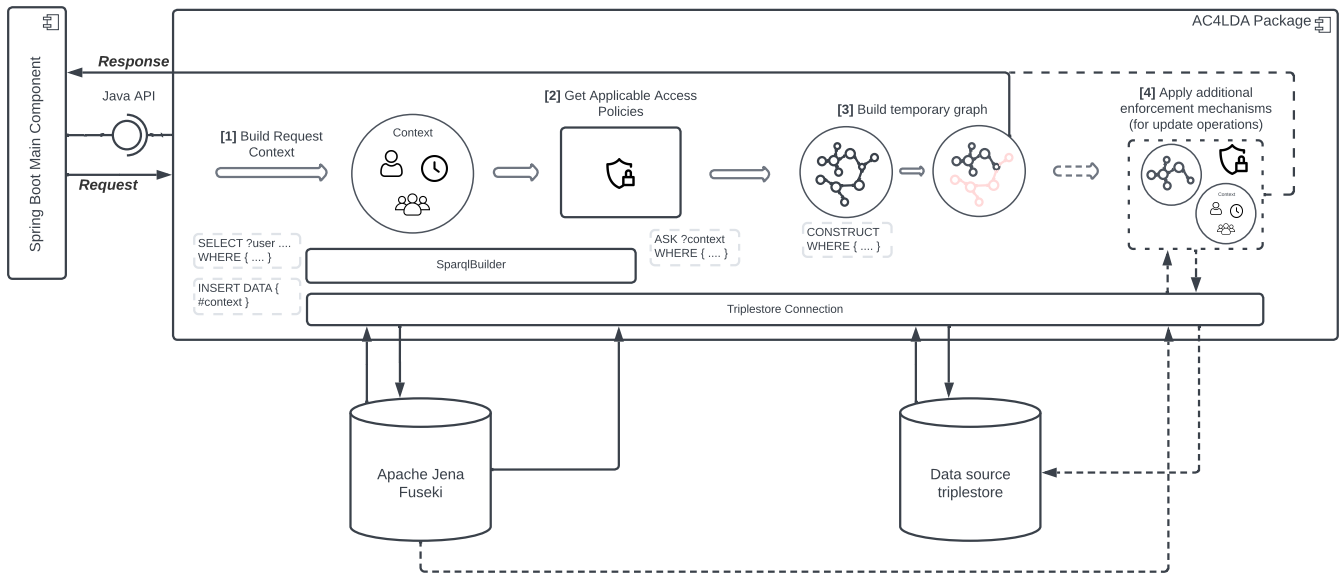


Figure 5.9: Sequence of operations in the AC4LDA Package.

construction of the SELECT query presented in Listing 5.3 that retrieves the basic information of a determined access policy.

```

1 // Definition of the IRI and the variables to be used in the query
2 Iri policy = ac4lda.iri(uuid);
3 Variable accessConditionSet = SparqlBuilder.var("accessConditionSet");
4 Variable accessConditionSetType = SparqlBuilder.var("accessConditionSetType");
5 Variable subtype = SparqlBuilder.var("subtype");
6 Variable target = SparqlBuilder.var("target");
7
8 // Definition of the type of the query, the used prefixes, and the selected
  variables
9 Queries.SELECT().prefix(rdf).prefix(rdfs).prefix(owl).prefix(s4ac).prefix(ac4lda).
10   select(accessConditionSet, accessConditionSetType, target).
11
12 // Specification of the WHERE clause
13 where(
14   // Specification of statements about the variable ?policy
15   policy.isA(s4ac.iri(s4acOntology.AccessPolicy)).
16     andHas(s4ac.iri(s4acOntology.hasAccessConditionSet), accessConditionSet
17       ).
18     andHas(s4ac.iri(s4acOntology.appliesTo), target),
19
20   // Specification of statements about the variable ?accessConditionSet
21   accessConditionSet.isA(accessConditionSetType).
22     // Addition of filter to remove unwanted results
23     filter(

```

```

24         Expressions.notEquals(accessConditionSetType, owl.iri("
           NamedIndividual")),
25         Expressions.not (Expressions.isBlank(accessConditionSetType))
26     )
27 ).
28 // Addition of filter to retrieve only the most specific type
29 filterNotExists(
30     accessConditionSet.has(rdf.iri("type"), subtype),
31     subtype.has(rdfs.iri("subClassOf"), accessConditionSetType).
32     filter(Expressions.notEquals(accessConditionSetType, subtype)))
33 );

```

Listing 5.2: Construction of a SPARQL SELECT query.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX s4ac: <http://ns.inria.fr/s4ac/v2#>
5 PREFIX ac4lda: <http://episa-labs.inesctec.pt/ac4lda#>
6
7 SELECT ?accessConditionSet ?accessConditionSetType ?target WHERE {
8     ?ap a s4ac:AccessPolicy ;
9     s4ac:hasAccessConditionSet ?accessConditionSet;
10    s4ac:appliesTo ?target .
11    ?accessConditionSet a ?accessConditionSetType
12    FILTER(?accessConditionSetType != owl:NamedIndividual && !isBlank(?
        accessConditionSetType)) .
13    FILTER NOT EXISTS {
14        ?accessConditionSet a ?subtype .
15        ?subtype rdfs:subClassOf ?accessConditionSetType .
16        FILTER(?accessConditionSetType != ?subtype)
17    }
18 }

```

Listing 5.3: SPARQL SELECT query to retrieve basic information about an access policy.

Apache Jena (RDF Java API)

Apache Jena is a Java-based framework for building and manipulating Semantic Web (RDF) data. It provides a set of libraries and tools for creating and querying RDF data, including a SPARQL query engine and inferencing using RDF-S and OWL. Additionally, the framework supplies other tools. As we have seen, Jena provides Fuseki, a triplestore server for storing RDF data which we used in the configuration of our platform. Nevertheless, the libraries provided by the framework are known to work with other triplestore services. This makes up a key factor since using these libraries means that our system does not stay bound to a singular RDF storing service. Services

such as Virtuoso and GraphDB are examples of triplestores with compatibility with these Java APIs.

In the AC4LDA Package, the Jena libraries are used to ensure the communication between the Spring Boot component and the triplestore services. Through this connection are executed several types of SPARQL operations required for the functioning of the AC4LDA Package. Additionally, the Jena libraries are used for the validation of SPARQL queries. As we described earlier, SPARQL queries are required in multiple steps of the access control enforcement process. For it to behave as expected, the executed queries must be valid so, we use the provided libraries to guarantee this aspect. This correctness verification occurs, for example, in the construction of the protected views where it is required the validation of the user-provided CONSTRUCT queries that define the data protected by a determined access policy.

5.2.1.2 Data Validation (SHACL)

When storing data crucial for the correct functioning of a software system it is vital to guarantee that such data is valid and leads to the expected behavior. Moreover, when that data is manipulated, that state of validity must be maintained. As presented above, the vocabulary used by the AC4LDA uses RDF to describe the Access Control Policies, Users, Roles, and the relations between them. One disadvantage of RDF-S vocabularies and triplestores is that the structure defined in the former is not natively enforced or verified by the last so this has to be carried out by a particular process. This can be accomplished using features provided by OWL. However, despite existing semantics in this language for defining multiplicity and data restrictions and the Apache Jena framework offering support on its use, OWL is more directed towards the definition of restrictions for inference and reasoning [103, 104].

Many triplestore services offer support for an alternative in the form of an engine that uses the Shapes Constraint Language (SHACL) [105], a W3C Recommendation to determine and ensure the validity of the data stored in an RDF graph. When compared with OWL in this department, SHACL provides more flexibility for error message definition and different levels of error, features that OWL lacks. Additionally, SHACL has a simpler syntax for defining constraints. Essentially, SHACL defines an RDF vocabulary capable of representing conditions used to validate RDF graphs. With this task in mind and using SHACL, we modeled the shapes for validating the data that is present and inserted in the policies graph. To perform this validation, it is required to describe the restrictions to be applied to data using SHACL's vocabulary. Even though there is a more human-readable syntax for SHACL, SHACL Compact syntax³, supported the Apache Jena framework, we opted to use an RDF syntax to specify the restrictions, more specifically Turtle [106], due to already having some familiarity with this representation.

Regarding the AC4LDA vocabulary, the restrictions we intended to enforce were related to class types, the number of resources linked through a specific property, and the datatype of literal

³SHACL Compact syntax, <https://w3c.github.io/shacl/shacl-compact-syntax/>

linked to individuals of a determined class. Table 5.2 shows the shapes and constraints enforced and the full specification is present in Appendix D.

5.2.1.3 AC4LDA Java API

The AC4LDA Package equips its host application with a Java API containing a heap of methods that implement the enforcement strategies discussed earlier. The package provides a class named AC4LDAService that enables the access control mechanisms in operations of reading inserting and deleting information from RDF graph sources, according to the conditions specified by previously defined Access Policies. Since these actions occur over an RDF graph and the execution of operations in such sources is usually performed through the usage of SPARQL, we decided that the provided API should also be in compliance with this protocol. So, with this in mind, we developed a series of methods that enable the execution of the different existing SPARQL operations. Regarding reading procedures, the AC4LDA Package includes the execution of queries of type SELECT, ASK, CONSTRUCT, and DESCRIBE. Whereas for update purposes, the package supports the execution of operations of type “Insert Data” and “Insert”, “Delete Data”, “Delete”. Additionally, in the case of the update operations, the package provides two additional methods that indicate the policies that restricted the requested operation. In Listing 5.4 we present the headers of the methods for SPARQL execution provided by the AC4LDA Java API. In this listing, we can observe that all methods have identical signatures. Aside from the operation to be executed, each of them requires the username of the already authenticated requesting user which is used to build the context of the request. Since the remaining user information required can be retrieved from the triplestore and the time-related contextual data can be produced internally, the username is sufficient for this task. The methods vary in the return type by virtue of the distinct results produced by the different SPARQL operations supported.

```

1 public List<HashMap<String, String>> readRequest(String username, String
    queryString) throws ServerException, UserException { ... }
2
3 public boolean askRequest(String username, String queryString) throws
    ServerException, UserException { ... }
4
5 public boolean insertRequest(String username, String queryString) throws
    ServerException, UserException { ... }
6
7 public Pair<Boolean, List<String>> insertRequestExp(String username, String
    queryString) throws ServerException, UserException { ... }
8
9 public boolean deleteRequest(String username, String queryString) throws
    ServerException, UserException { ... }
10
11 public Pair<Boolean, List<String>> deleteRequestExp(String username, String
    queryString) throws ServerException, UserException { ... }

```

Listing 5.4: Definition of the methods for SPARQL operations provided the AC4LDA Java API.

The layer for the authentication of users was one aspect kept out of the scope of the AC4LDA Package. So, since the responsibility of its implementation was left to the host Spring Boot application, it was required to provide the means to access user data for that purpose. For that reason, aside from providing the already mentioned methods for SPARQL execution, the API supplies the procedures exhibited in Listing 5.5, which retrieve and validate authentication information. This validation is a process that evaluates if the password provided matches the encrypted password for the user with the provided username.

```
1 public boolean validateUser(String username, String password) throws
    ServerException, UserException { ... }
```

Listing 5.5: Definition of the methods for authentication provided the AC4LDA Java API.

5.2.1.4 AC4LDA REST API

To facilitate the interaction with the AC4LDA system, avoid the need to write Access Policies directly in RDF, and enable the system to create new access control conditions and modify or delete existing ones, we implemented a REST API that supplies endpoints for these purposes. This REST API implements the operations known as CRUD as exposes a series of endpoints that allow viewing, editing, and deleting the Access Policies, Users, and Roles that the system stores and, additionally, creating new individuals of these classes. In Table 5.3 is presented a summary of the implemented endpoints, information that is also present and described with more detail in the OpenAPI specification of the API displayed in Appendix C.

The AC4LDA API's endpoints follow a strict but simple response format as every request is provided with a comprehensible JSON object. The structure of this JSON object is based on the specification presented by omniTI-labs, JSend [107], which defines a structure for data objects included in HTTP responses in order to extend the messages that the HTTP status codes have the capacity to provide. The response object structure is based on 3 fields, the two string value fields status and message, which respectively, are set with a string that indicates the result of the request performed and a string that sends additional information about the obtained outcome, and the field data, which can either be an object or an array of objects and is set with any information that needs to be sent back to the requester. Each request sent can reach any of the following three states where, for each case, the resulting response has distinct requirements. In case of success, the status field is set with the value "success", and the data field is either set with a null value or with the object or array of objects required by the request. In case of error, a state that is met when, while processing a request, the server reaches an error for a reason unimputable to the user, the status field is set with the value "error", and the message field is set. Lastly, in case of failure,

a state that is reached when the request is flawed, for example, when API is not correctly used or there is a problem with the sent data, the status field is set to "fail", and the message field is set. Optionally, the data field is set with an object with specific details that precisely identify the fault in the received request, a slightly different approach from the original JSend specification, which indicates the data field must always be set and that the response should not contain a message field when a failure is detected. Note that these each of these three outcomes has a set of corresponding HTTP response codes, when the status is set to "success" the HTTP response code will be set to values of type 2xx, when set to "fail" a value in the class of 4xx, and when it is set to "error", a value in class of 5xx.

5.2.1.5 Integration

To integrate the AC4LDA package into other projects that want to enable the implemented access control features, one must follow a set of steps that go from importing the source code of the package to configuring some aspects of its behavior.

The AC4LDA package was compiled and built into a Maven⁴ package in order to become an easy-to-use software component. The import process is no different from the one followed when importing other Maven modules so, it is required to add the present in Listing 5.6 to the project's dependency file.

```
1 <dependency>
2   <groupId>com</groupId>
3   <artifactId>ac4lda</artifactId>
4 </dependency>
```

Listing 5.6: Maven dependency of the AC4LDA package.

Among the Spring Boot environment, the usage of Java annotations is a common practice as they are used to inject features in classes, methods, and variables and to instruct Spring Boot to perform certain operations. For example, the annotation `@RestController` along with subsequent `@GetMapping`, `@PostMapping` are used to the definition of API endpoints and respective controller functions. Because of this and the fact that it comprises an elegant and quickly-removable instruction, we decided to follow this approach and add it as one of the steps of the method that integrates the AC4LDA system into an application. In Listing 5.7 is shown how the definition of this annotation was accomplished, whereas Listing 5.8 displays how it should be used.

```
1 @Retention(RetentionPolicy.RUNTIME)
2 @Target(ElementType.TYPE)
3 @Import(Ac4ldaConfiguration.class)
4 public @interface EnableAc4lda {
```

⁴Maven, <https://maven.apache.org/>

```
5 }
```

Listing 5.7: Definition of AC4LDA Annotation.

```
1 @SpringBootApplication
2 @EnableAc4lda
3 public class ApplicationMainClass {
4     ...
5 }
```

Listing 5.8: Usage of AC4LDA Annotation.

Finally, in order to configure the imported software it is required to set two variables crucial for the functioning of the AC4LDA package. These are the variables that set the hostnames of the dataset used to store the system's information and the dataset meant to be secured, files containing data for the initialization of both datasets, the location of the vocabulary and constraint files, and setting for allowing or not partial updates. Such configuration should be effected in the host application's properties file. If the file is a text-based file, this configuration should be done as shown in Listing 5.9.

```
1 ac4lda.host=http://localhost:3030/policies
2 ac4lda.protectedHost=http://localhost:3030/default
3
4 ac4lda.initHost=/home/tests/performance-test-cases/25k/all.n3
5 ac4lda.initProtectedHost=/home/tests/quality-test-cases/context/sp2b.n3
6
7 ac4lda.resources=/home/resources/
8
9 ac4lda.partialUpdates=false
```

Listing 5.9: Sample configuration file for the AC4LDA Package.

Completing these actions concludes the integration of the AC4LDA package which results in two consequences: it makes available the AC4LDA Java API and adds the AC4LDA REST API to the host application.

5.2.2 The AC4LDA Admin Interface

Aside from integrating into the AC4LDA system an API that eases the management of the entities of the access control environment, Users, Roles, and Access Policies, we also developed a client for that API under the form of a straightforward web application that acts as an administration GUI. The focus when developing this client was to provide a GUI that takes advantage of all of AC4LDA API endpoints and provides an alternative to programmatically configuring the desired access restrictions. The Admin Interface was implemented using Laravel [108], an open-source

PHP web framework for building full-stack web applications. While there were other options for developing applications of this kind, similarly to the reasons that motivated choosing other technologies for developing other parts of this work, the Laravel framework was already familiar to us and is a framework that produces server-side applications, with less impact on the end users' computer resources when compared to JavaScript frameworks commonly used to build front-end applications. Aside from this, since Laravel was already part of the technology stack of the EPISA project, we elected to align this decision with that stack. For development and execution purposes, the application was enclosed in a virtual environment through the use of Docker containers, which we describe in Appendix B.

5.2.2.1 Application Architecture

Laravel applications, under the recommendation of the framework's documentation, are usually implemented using the Model-View-Controller (MVC) architectural pattern since the framework's implementation is based on this pattern. The MVC pattern, commonly used for the implementation of user interfaces, divides the implementation of a software system into three elements: Models, Views, and Controllers. The Models are the conceptual representation of the information stored by the system, Views are the visual representation of that information, and, lastly, Controllers are the more central elements and are responsible for interacting with the Models and reflecting the data retrieved from them in the View components [109]. Even though it is not ideal for cases where the Model element does not have an explicit role, for the implementation of the Admin Interface, we elected to follow this pattern as well. However, since the AC4LDA Admin Interface is not meant to effectively hold data and recur only to the AC4LDA API as a data source, the application does not formally contain any Models and implements only Controllers and Views. Therefore, the implementation of the application involved the development at these two levels.

Controllers The Controllers are responsible for the implementation of the logic of the application. This task comprises mainly two processes: accessing the data that the app serves and channeling it to the View elements, handling the requests to modify the same data performed by users through their interaction with the Views. As we have already mentioned, the Admin GUI does not implement any Model elements since it does not possess data, so Controller elements mainly comprise the usage of an HTTP client package to call the REST API endpoints provided by the AC4LDA Package. This was accomplished using Guzzle [110], a package for performing HTTP requests for the PHP language that the Laravel framework integrates. Within the context of this component, it was also required the implementation of the application's routes, which are responsible for the definition of the application's endpoints and for their mapping with the available pages and the different actions provided by the Controllers. Table 5.4 displays these routes and briefly describes their purpose.

Views The Views are the element responsible for depicting the information provided by the Controllers. In this case, In the AC4LDA Admin GUI, this component was implemented with Blade [111], a template engine supplied by the Laravel framework to build the structure and the

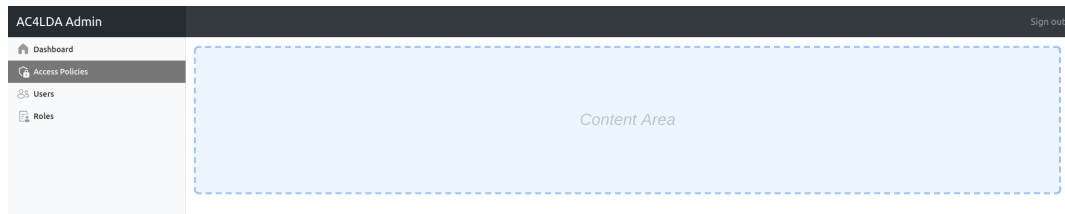


Figure 5.10: Layout of the AC4LDA Admin GUI.

contents of HTML pages and Bootstrap [112], a collection of CSS and JS libraries that aid the development of such pages by providing a set of pre-built components and easily configurable styles. Additionally, we resorted to third-party components of this library, Tempus Dominus [113] and Multiselect [114], in order to accommodate the needs of a time and date select box and a multivalue select box. Finally, we also made use of Yasgui [115], a JS library that provides easy integration with a SPARQL-enabled query text editing area along with features of executing operations on remote SPARQL servers, to provide a section to ease the construction and to preview the effect of the SPARQL CONSTRUCT queries required by Access Policies.

5.2.2.2 Implemented Interfaces

Ultimately, the Admin GUI comprises a group of interfaces meant to expose the different administration activities that involve the specification of the AC restrictions, to affect the behavior of the AC4LDA in the enforcement process. Such activities include listing, creating, editing, and deleting the entities that compose the AC4LDA platform: Roles, Users, and Access Policies. The application was built over a straightforward layout common to all existing pages. This layout, presented in Figure 5.10, possesses an area where the different interfaces diverge, represented by the highlighting in the depiction. The content of that area varies according to the purpose of the current page. The figures presented below comprise the interfaces with the content that is fitted inside this area. This layout is essential for navigation since it connects the different available pages together. From the “navbar” that composes the layout, we can access the interfaces for viewing the existing individuals of the different entities and, from there, navigate to the creating and editing pages of each one of them. The navigation in the web app is represented in Figure 5.11, which depicts the sitemap of the Admin GUI.

The web app is composed of the implementation of the different interfaces relative to the Roles, Users, and Access Policies concepts. In Figure 5.12 we can see the interface that presents and allows the deletion of existing Roles, whereas in Figure 5.13 the interface that enables the edition of instances of that entity. Similarly, in figures 5.14 and 5.15, we can see the implemented interfaces for the same purpose for Users. The interfaces for these two entities are relatively simple since each is composed of a small amount of information. The interfaces for listing and deleting existing individuals are composed of a table that presents the information regarding each one of them. In contrast, the creation and edition interfaces are composed of straightforward forms containing fields to provide the necessary information for each entity. Finally, the interfaces

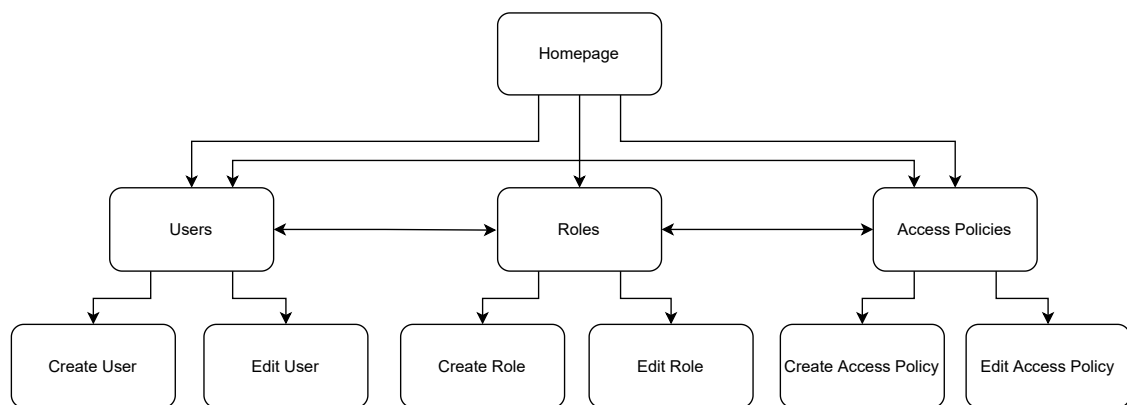


Figure 5.11: Sitemap of the AC4LDA Admin GUI.

for managing Access Policies are presented in figures 5.16 and 5.17. As it can be seen, these interfaces are more complex than the ones relative to the Role and User entities mentioned before. This is due to the fact that an access policy aggregates more information than each of the previous entities. Aside from the meta-information about the policy, the interfaces integrate more intricate sections for specifying the policy's subject and object.

Roles Add Role

URI	Name	Description	Users	
ac4lda:8247c828-4e0b-4ba8-906f-5dbf56adac29	Public	No description provided...	No associated users...	 
ac4lda:a83545f6-fe9b-4ee8-a6d7-d6772a613a77	Archivist	No description provided...	No associated users...	 
ac4lda:861b7779-b28b-46ef-8bb1-7dac7380ec72	Admin	No description provided...	No associated users...	 

Figure 5.12: Interface of the AC4LDA Admin GUI for viewing and deleting existing Roles.

Edit Role

Name

Public

Description

The public role aggregates a set of permissions and access levels that are available to any user or external system that interacts with the system.

Cancel Save

Figure 5.13: Interface of the AC4LDA Admin GUI for editing existing Roles.

Users Add User

URI	Username	Roles	
ac4lda:b3caca0f-93dd-4622-95cf-c5ba4eb187d1	admin_user	Archivist Admin	 
ac4lda:3d619259-451a-4c47-ae7f-d1d64639d365	archivist_user	Archivist	 
ac4lda:b11b08ca-8c14-4603-9d42-42033925849b	public_user	Public	 

Figure 5.14: Interface of the AC4LDA Admin GUI for viewing and deleting existing Users.

Create New User

Username

public_user

Roles

Public  Search

☐ Select all

☒ Public

☐ Archivist

☐ Admin

Password

Confirm Password

Cancel Submit

Figure 5.15: Interface of the AC4LDA Admin GUI for creating Users.

Concept Class	Constraints
Access Policy	• has at most one individual of type <code>AccessPolicyType</code>, linked through the property <code>hasType</code> • has between one and four individuals of type <code>AccessPrivilege</code>, linked through the property <code>hasAccessPrivilege</code> • has exactly one individual of type <code>AccessConditionSet</code>, linked through the property <code>hasAccessConditionSet</code> • has at least one resource or literal linked through the property <code>appliedTo</code>
Access Condition Set	• has at least one individual of type <code>AccessCondition</code>, linked through the property <code>hasAccessCondition</code>
Access Condition	• has exactly one string literal, linked through the property <code>hasQueryAsk</code> • the property <code>user</code> links only to individuals of class <code>User</code> • the property <code>role</code> links only to individuals of class <code>Role</code> • has at most one literal of datatype <code>xsd:dateTime</code> linked through the property <code>fromTime</code> • has at most one literal of datatype <code>xsd:dateTime</code> linked through the property <code>toTime</code> • if it <code>fromTime</code> is always less than <code>toTime</code> if both values are set • has at least one resource linked either through the property <code>user</code> or the property <code>role</code>
User	• has exactly one literal of type <code>xsd:string</code>, linked through the property <code>hasUsername</code> • has exactly one literal of type <code>xsd:string</code>, linked through the property <code>hasPassword</code> • the property <code>role</code> links only to individuals of class <code>Role</code>
Role	• has exactly one one literal of type <code>xsd:string</code>, linked through the property <code>hasName</code> • has at most one one literal of type <code>xsd:string</code>, linked through the property <code>hasDescription</code>
Environment	• has exactly one literal of type <code>xsd:dateTime</code>, linked through the property <code>time</code>
Context	• has exactly one individual of type <code>User</code>, linked through the property <code>user</code> • has exactly one individual of type <code>Environment</code>, linked through the property <code>environment</code>

Table 5.2: Restrictions applied to each of the AC4LDA vocabulary concept classes.

Path	HTTP Method	Description
/ac4lda/users	GET	Gets all users
/ac4lda/user?uuid=id	GET	Gets the user with the specified id
	DELETE	Deletes the user with the specified id
/ac4lda/user	POST	Creates a user based on the information in the object provided
	PUT	Updates the user with the specified id with the information in the object provided
/ac4lda/roles	GET	Gets all roles
/ac4lda/role?uuid=id	GET	Gets the role with the specified id
	DELETE	Deletes the role with the specified id
/ac4lda/role	POST	Creates a role based on the information in the object provided
	PUT	Updates the role with the specified id with the information in the object provided
/ac4lda/policies	GET	Gets all access policies
/ac4lda/policy?uuid=id	GET	Gets the access policy with the specified id
	DELETE	Deletes the access policy with the specified id
/ac4lda/policy	POST	Creates an access policy based on the information in the object provided
	PUT	Updates the access policy with the specified id with the information in the object provided

Table 5.3: AC4LDA REST API Endpoints.

Path	HTTP Method	Description
/	GET	Displays the page for viewing and deleting existing access policies
/new	GET	Displays the page for creating new access policies
	POST	Creates a user based on the information in the object provided
/edit/{uuid}	GET	Displays the page for editing existing access policies
	PUT	Updates the user with the specified id with the information in the object provided
/users	GET	Displays the page for viewing and deleting existing access policies
/users/new	GET	Displays the page for creating new access policies
	POST	Creates a user based on the information in the object provided
/users/edit/{uuid}	GET	Displays the page for editing existing access policies
	PUT	Updates the user with the specified id with the information in the object provided
/roles	GET	Displays the page for viewing and deleting existing access policies
/roles/new	GET	Displays the page for creating new access policies
	POST	Creates a user based on the information in the object provided
/roles/edit/{uuid}	GET	Displays the page for editing existing access policies
	PUT	Updates the user with the specified id with the information in the object provided

Table 5.4: AC4LDA Admin GUI Routes.

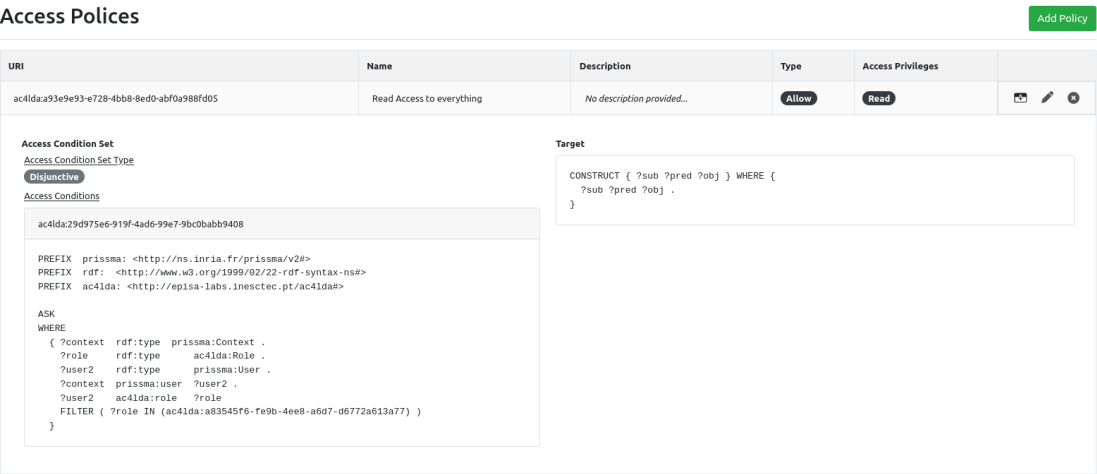


Figure 5.16: Interface of the AC4LDA Admin GUI for viewing and deleting existing Access Policies.

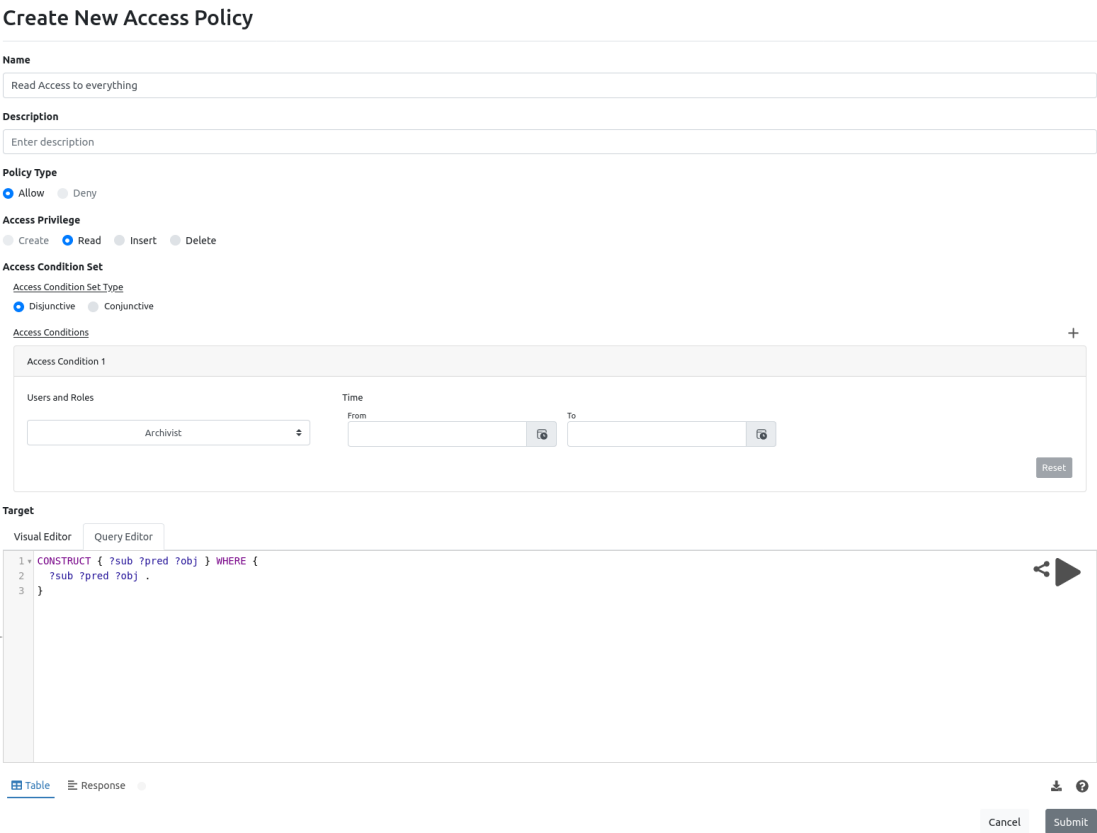


Figure 5.17: Interface of the AC4LDA Admin GUI for creating Access Policies.

Chapter 6

Evaluation

One of the main goals of this work was to develop a solution suitable for the Archival domain for performing access control over a linked data system. In this chapter, we perform the validation and evaluation of the work accomplished. We describe the methodology used for performing these tasks to present and discuss afterward the results obtained. The process described in this chapter solely takes into account the AC4LDA Package. The Admin GUI, since there were no requirements, nor a study developed to direct its implementation, was not formally appraised. Finally, it is important to note that this process is a result of an introspection where we did not perform any survey, and no outside opinions were considered. Still, we sought to be as objective as possible.

6.1 Methodology

For the access control mechanisms to work and to be possible to put them to practice, it is required for them to be tested and verified to gauge their overall effectiveness and assess their adequacy for the needs of a specific use case. As it is observed by Kirrane et al. [25], as of 2017 there was no standard for conducting such types of evaluation, a situation that until today has remained the same. Because of this, some authors have opted to formulate their validation method. In Chapter 3 we mention some of these techniques since they are present in the research works we covered. Unbehauen et al. [85] make use of the Berlin SPARQL Benchmark to access and compare how the different implemented alternatives perform when executed against a growing amount of data. Stojanov et al. [56] follow an identical strategy but, instead, chose to use a dataset generated for the use case utilized for the validation of the implemented platform. Another example is the evaluation performed by Abel et al. [10], which, aside from measuring performance by increasing the size of the dataset being protected, also increases the complexity of the access policies to measure the effect of query rewriting.

After appraising the procedures followed in other works, we planned our own validation and evaluation strategy. Here we opted to partially follow the methods of other works. We perform an evaluation on two levels, a qualitative and a quantitative level. At the qualitative level, we evaluate

the results produced based on the requirements elicited in Chapter 4. Additionally, we tested the correctness of the implemented system by conducting a series of integration tests that appraised if the results obtained are the ones expected. Then, we evaluated it at a quantitative level, where we conduct a performance evaluation of the implemented access control strategies and assessed the behavior of the system when exposed to different scenarios. Inspecting with more attention these two degrees:

Quality Level The validation at the quality level is about gauging if and how the solution we provide responds to the needs of the scenario we aim to answer. That is, if the previously set objectives are met. For the assessment of these factors, we will perform a validation considering the following aspects:

- The generic access control requirements for linked data defined by the W3G Working Group on Linked Data.
- The specific requirements of the Archival domain collected for this work.
- Building and executing a test suit to examine if the provided resources are effective.

Performance Level While the quality level validation attempts to determine if and how the solution provided fits the problem the work proposes to solve, the performance level, or the quantitative level, is about checking how well the system responds to requests. At this level, we aim to understand whether if or in what scenarios the usage of our solution is feasible. To do so, we will perform tests and measure mostly time and size values while varying a number of key aspects such as the size of the protected dataset, the number of existing roles, users, and access policies. In this evaluation we will also separate the process of enforcement of the multiple types of access policies over the different types of available requests to determine the process' bottleneck. Hopefully, the values obtained by this evaluation will tell us in what situations the usage of the AC4LDA Package is or is not advantageous. This evaluation will be based on the strategy followed by other research works by using a dataset of a SPARQL benchmark to evaluate the performance of responses. While the ideal testing scenario would be one that closely resembles the one seen in an Archival context, producing datasets for contexts like these is not trivial, and neither would be the definition of the CONSTRUCT queries required by our solution for the specification of access policies. This is due to the intricacy of Archival domain vocabularies, which we have already noticed in Chapter 4. So, one aspect that we had to overlook was the complexity of the CONSTRUCT queries, which may not be the same as in a real scenario of the Archival domain. Still, we feel that a less detailed vocabulary provides a good enough testing scenario. In works that focus solely on the benchmark of SPARQL endpoints, we can see that the complexity of the queries has less impact on the response times than the potential number of results returned by a given query, so just varying the number of expected results of a given query already provides an acceptable testing sample.

6.2 Results

6.2.1 Quality Level

As mentioned above, at the quality level, we performed an analysis with two key factors: the requirements listed in Chapter 4, and the completion of a group of tests.

6.2.1.1 Requirements

From a requirements-related standpoint, our solution is in partial conformity with the requirements, as Tables 6.1 and 6.2 show. In the case of the generic access control requirements, the AC4LDA Package complies with the requirements GR2-5 but disregards the requirements GR1 and GR6. In the case of the requirements for access control in the Archival domain, the requirements were entirely met.

Generic Access Control Requirements	
Compliant with	GR2; GR2a; GR2b; GR2c; GR3; GR4; GR5 (partially)
Uncompliant with	GR1; GR6

Table 6.1: Compliance with the Generic access control requirements.

Archive-specific Access Control Requirements	
Compliant with	SR1; SR2; SR3; SR4; SR5
Uncompliant with	—

Table 6.2: Compliance with the archive-specific access control requirements.

The AC4LDA Package complies with the requirements GR2 and its sub-items since it comprises a solution that allows the specification of either a specific or a group of users through the usage of roles, the fine-grained specification of data using SPARQL CONSTRUCT queries and their association in the form of an Access Policy. Simultaneously, because of these characteristics, the SR1, SR2 and SR5 are also fulfilled. Additionally, since the vocabulary used for representing access policies is enabled for the definition of contextual information such as the time of the request the SR3 is also attained. Aside from this, the AC4LDA enables the attachment of Access Privileges to policies at a fine-grained level, that is, for the different actions of reading, inserting, and deleting, meaning that the GR3 and SR4 are accomplished. Finally, regarding the GR5, this requirement, as is noted in Table 6.1, is only partially fulfilled since, for read operations, the AC4LDA Package allows by default the partial execution of operations, something that we have already described in the previous chapter. This means that a request for reading will always be successful independently of if a part of the access was or was not denied. While we could have developed a procedure for determining if that happened, we felt that disclosing to the requesting

agent that a part of his request was denied and providing the information of what is necessary for the decision to be contrary introduced a security threat in the system.

Concerning the requirements that were neglected, **GR1**, **GR4**, and **GR6** were overlooked for different motives. For the **GR1**, even though we planned to include an authentication layer in the implementation of the AC4LDA Package, due to time constraints, we were not able to complete it. During development, we were able to reach an authentication solution; however, we were incapable of producing a functional authentication procedure that allowed keeping the characteristics of reusability and extendability that the AC4LDA Package, as a library, should feature. For the **GR4** and **GR6**, we deliberately left them out of the AC4LDA Package since they were not a demand in the context of Archives. Additionally and similarly to what happened with the **GR5**, in the case of the **GR6** we thought it comprised a risk in terms of security since the knowledge of in what RDF graphs a given resource is present would be provided to any user-agent.

6.2.1.2 System Testing

Regarding the testing activities performed, these were subdivided into two groups that coincide with the two stages of the enforcement process: the evaluation of the context and the actual enforcement of the defined policies over the requests received. The goal of these tasks was mainly to assess the effectiveness of the defined strategies and to gauge the correctness of their implementations. For this, we initialized a new Spring Boot application to integrate the AC4LDA Package and provide a testing environment and created a pair of basic scenarios to provide a setting for completing the tests.

Evaluation of Context

For the evaluation of context, we constructed a scenario with the objective of it being representative of the different context possibilities there may exist. With respect to the user and role entities, we built a situation so that there was a user with one, multiple, and no association to roles and, equally, a role with one, multiple, and no users attached. Figure 6.1 depicts this setting by showing the connection between the users User1, User2, and User3 and the roles Role1, Role2, and Role3. Then, relative to the access policies, we also created a group of policies that guaranteed that

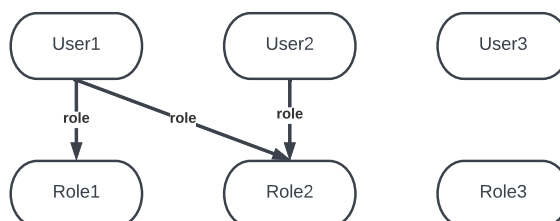


Figure 6.1: Association of Users and Roles for testing the evaluation of context.

all of its intricacies were tested. We specified policies with one and multiple access conditions,

ID	Access Set type	Condition	Nr. of Access Conditions	Access Conditions description
AP1	Disjunctive		1	Subject must be associated with Role1
AP2	Disjunctive		1	Subject must be associated with Role2
AP3	Disjunctive		1	Subject must be associated with Role3
AP4	Conjunctive		2	Subject must be associated with Role1 and Role2
AP5	Disjunctive		2	Subject must be associated with Role3 or be User3
AP6	Disjunctive		1	Subject must be User3 and time of the request must be between 30/12/2022 and 30/01/2023
AP7	Disjunctive		1	Subject must be User3 and time of the request must be between 30/01/2023 and 30/03/2023
AP8	Disjunctive		1	Subject must be User3 and time of the request must be between 30/03/2023 and 30/04/2023

Table 6.3: Access Policies for testing Evaluation of Context.

with different operators for the combination of conditions that can be either a conjunctive or a disjunctive operator, and looked to explore the different possible types of restriction. As the motive of this group of tests was only to verify the evaluation of the context, some details pertaining to the Access Policy entities, such as their type, access privileges, and the SPARQL query responsible for defining the data they affect, were negligible. Thus, we created the set of policies specified in Table 6.3. Then, we executed the evaluation of context for each of the three specified users through the method “userPolicies”, provided by the AC4LDA Package API. By analyzing the information that Figure 6.1 and Table 6.3 contain, we can see what the expected results of that evaluation are. For the evaluation of “User1” it is expected that the Access Policies AP1, AP2, and AP4 are positively evaluated. For “User2”, only AP2 should be returned, whereas for “User3”, considering the date of request is, for example, 1/02/2023, AP5 and AP7 are the policies that should be present in the response.

After the execution of the tests, we verified that the results corresponded with the expectations, as all contexts were properly evaluated and matched correctly with the existing Access Policies.

Enforcement in SPARQL operations

For the verification of the enforcement of access control in the supported operations by the AC4LDA Package, the strategy was similar to the one used to validate the context evaluation phase. We created a heap of Access Policies representing the different types of protection that the AC4LDA Package allows specifying. This time, the focus was not so much on the context or the conditions of the policies but more on the type, privilege, and data selection details that were overlooked in the previous testing phase. Therefore, we created a set of users and a group of Access Policies

ID	Access Conditions				Type	Access Privilege	Target description
AP1	Subject	must	be	Allow	Read	All information from documents of type bench:Journal	
AP2	Subject	must	be	Allow	Insert	All information to journal from 1942	
AP3	Subject	must	be	Allow	Delete	All information from journal from 1942	
AP4	Subject	must	be	Allow	Read	All information	
AP5	Subject	must	be	Deny	Read	All information from journal from 1940 and its articles	
AP6	Subject	must	be	Allow	Insert	All information about articles	
AP7	Subject	must	be	Deny	Insert	Link articles to journals	
AP8	Subject	must	be	Allow	Delete	All information about articles	
AP9	Subject	must	be	Deny	Delete	All links from articles to journals and the types of articles	

Table 6.4: Access Policies for testing Enforcement phase.

associated with them. Then, we conceived a diverse set of SPARQL operations to run in different contexts and test the various features of the AC4LDA Package. The elicited policies and operations are summarized in Tables 6.4 and 6.5. Aside from executing the operations exactly as Table 6.5 details, in some cases, we also varied the type of operation. For example, operations OP10 and OP11 were executed as CONSTRUCT, ASK and DESCRIBE operations as well.

The dataset of the protected data used in this stage is a subset of the ones used for the testing of the system's performance, which is detailed in the next section. This dataset represents entries of the DBLP database using the vocabulary represented in Figure 6.2, as the example in the same figure depicts. It provides samples for the description of Articles, Journals, and Incollections and amounts to approximately 1,000 triples, a number that rises if a reasoner is activated for inference. The operations presented in Table 6.5 took into consideration the content of this dataset as it is noted in their description.

Regarding the effective evaluation, it comprises running the prepared operations while being restricted by the Access Policies drawn out. This process was completed in different inference conditions, since we tested both with and without a reasoner and with different rationales for the partial execution of operations.

through AP1, and the update operations OP5 to OP8 worked accordingly with the restrictions applied by AP2 and AP3. Then, the operations executed in the context of “User3” also produced the expected results. The reading operations OP9 to OP11 returned the data as expected, and the update operations OP12 to OP15 updated the data as we intended. In respect to the latter group, when executed while the system is configured to allow partial updates, the insert operations OP12 and OP13 and the delete operations OP14 and OP15 produced the same results, as intended. Finally, the remaining operations were used to positively validate that it is not possible to override the setting where the operations are executed through the usage of “FROM” and “USING” clauses in the submitted SPARQL queries.

6.2.2 Performance Level

In regards to the evaluation at the performance level, the process was similar to the one followed for the testing at the quality level. Nonetheless, since its goal was to quantitatively assess the behavior of the platform, some aspects had to be modified. The first one was the data to be protected. While in the previous testing activity, the protected data comprised a small set of data, in this phase, it is required both a larger and a group of datasets of diverse sizes. For this matter, like the authors of other works elected to do, we opted to use the contents produced by a SPARQL Benchmark study to obtain data samples to which we want to protect access. Here, we analyzed several different options [116] but ultimately only strongly considered the ones that were the most available. The already mentioned BSBM, the SP2Bench, and the DBpedia SPARQL benchmark are works that provide accessible data generation tools to assemble datasets of different sizes [9, 88, 117]. Ultimately, we decided to make use of the tool provided by SP2Bench since it produces datasets based on a straightforward vocabulary describing entries of the DBLP¹ database.

Since, even with small volumes of data, running all components of the AC4LDA platform in our personal machines was cumbersome, one other aspect that was required to diverge from the process detailed earlier was the testing environment. For the execution of the performance tests, we used a Linux virtual machine with greater computational resources, where we were able to run and execute the platform and the testing activities. For this exercise, we constructed a straightforward scenario at the context and enforcement level. Then, we used it to test the efficiency of the implemented strategies at different dataset sizes and modified it to analyze the impact of factors like the number of Access policies and the allowed data size on the overall performance.

The conducted experiment included performing access and update operations in datasets of different sizes, with 25k, 75k, 225k, 675k, and 2 million triples and in different scenarios, by varying the number of applicable Access policies, with one, four and sixteen policies. The experiment was concluded with the execution of the various operations both in a protected and unprotected setting to evaluate the influence of the application of the techniques for enforcing access control in the performance of a linked data system. At this level, the tests performed were composed of a

¹DBLP, <https://dblp.org/>

Dataset size	Number of policies			
	n = 0	n = 1	n = 4	n = 16
25k triples	245.2	259.2	427	1077.8
75k triples	493.8	1693	1711	2953.4
225k triples	1270.2	5156.4	4686.8	7074.8
675k triples	3862.2	16281	16052.4	19599.4
2m triples	20769	64412.4	52546.4	87035.8

Table 6.6: Average execution times for tests (ms).

straightforward query that reads all of the data available in the given context. In each of the testing scenarios within the different dataset sizes, the set of data retrieved was the same. In the unprotected setting, it was retrieved all of the information stored since there are no restrictions to its access; in the setting with protection included, this was also the case since the constructed access policies were designed to allow access to all data. This was done so that the number of evaluated triples when a request is received was the same in each case, and we were able to observe the time added by the authorization process. In the cases where there were several access policies constructed, we scattered the selection of data over that number of policies by dividing the total number of triples of the dataset being used and splitting their selection over multiple construct queries using SPARQL’s LIMIT and OFFSET features. The three queries present in Listing 6.1 exemplify how this was achieved. The first query refers to the entirety of a graph’s data, while the following two queries, considering a graph with 1,000 triples, refer to half of the data each.

```

1 CONSTRUCT WHERE { ?s ?p ?o }
2
3 CONSTRUCT WHERE { ?s ?p ?o } LIMIT 500
4 CONSTRUCT WHERE { ?s ?p ?o } LIMIT 500 OFFSET 500

```

Listing 6.1: Selection of a 1,000 triple graph using one and two SPARQL CONSTRUCT queries.

Regarding the testing procedure, it was composed of performing various rounds of the same request in the different scenarios constructed, measuring the time taken to complete each one of them, and then calculating their average. In Table E.1, in Appendix E, we present the results obtained in each experiment. Table 6.6 summarizes those results by presenting the average of the values measured in each of the testing scenarios. By inspecting the values present in this table, we can draw some conclusions about the adequacy of the implemented methodologies for performing access control over linked data.

The application of these techniques, as expected, increases the response times to the performed requests. The addition and further increase of number of applied policies in each scenario adds to the total processing time. Yet, one key aspect is that the average response times, despite growing

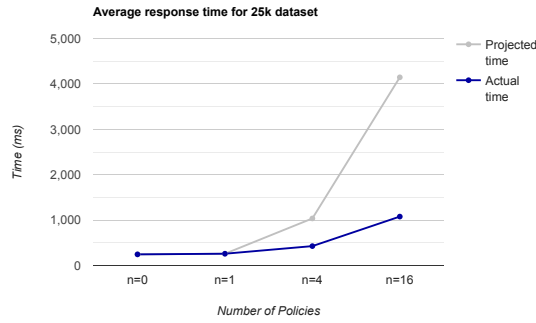
with the increasing number of policies, and still at a linear rate, they do not meet the multiplication factor that we used for increasing their count. In the graphs of Figure 6.3 is depicted the growth of the the actual average of the times measured and a projection using a successively quadrupling the values of the second column of Table 6.6. As it can be noted, the lines indicating the actual measurements are consistently below the projection lines. The same cannot be said in regard to the increase of the dataset size. As the graphs in Figure 6.4 show, successively tripling the dataset's size results in a growth of the average response times at similar and larger rates. Although this behaviour can also be detected when no access policies are considered, as we can observe in Figure 6.4a, their introduction negatively impacts the linked data system's performance, as the graph in Figure 6.3 show. Thus, introducing our solution for applying AC over linked data sources reduces the dataset size that a linked data system is capable of hosting while providing the requesters with a response in a timely manner. Considering, for example, the dataset of two million triples, while the approximately 20 seconds taken by the unprotected setting may amount to an acceptable time for retrieving all of the data, the same cannot be said for the 80 seconds required to retrieve the same amount of data in a scenario with 16 access policies.

Lastly, as the values present in Table 6.6 and in the graphs of figures 6.3 and 6.4 show us, the implementation and usage of the methodologies we developed to ensure authorized access to linked data is dependent on the both the size of the dataset to be protected and on the number of access policies required. Thus, its usage is reliant on the balancing of weights of these two aspects.

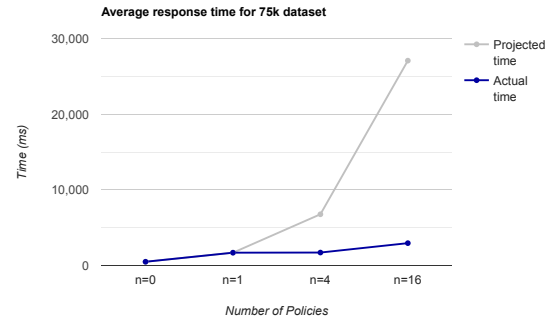
6.3 Discussion

As the previous section demonstrates, in this work, we have built a valid solution for enforcing access control suitable for archival linked data. As pointed out by the first of the two-part testing and validation process, the AC4LDA system is in compliance with the requirements of the archival domain. At the same time, it also complies with most of the high-level requirements for access control over linked data, defined by the W3C. Our solution, as noted by the tests conducted, detailed in Subsection 6.2.1, successfully provides features to impose restrictions to read and update actions. In the end, we consider it to be one of the most suitable solutions to serve the archival domain. When compared with the other existing options, it is more or at least equally adequate to cater for the archival domain.

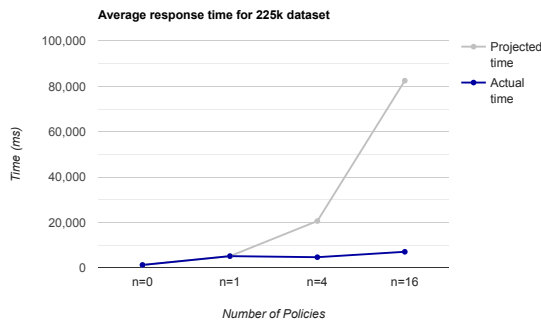
The answer provided by existing RDF storing services lacks the required protection granularity. As we have seen, only Stardog is capable protecting at the level of the triple. In general, the outcomes of the analysed research works are also inadequate and insufficient for the use case of archives. Most of them have limitations either in terms of the model implemented, the granularity of protection, or operations protected that the context of archives demands. One exception among this group of works is the work accomplished by [Stojanov et al.](#), which produced the LDA, a platform that could be used to secure the contents and the operations executed over RDF triples sources while following a context-based approach. Additionally, the resulting platform implements authentication methods something that we left out of the scope of the AC4LDA package,



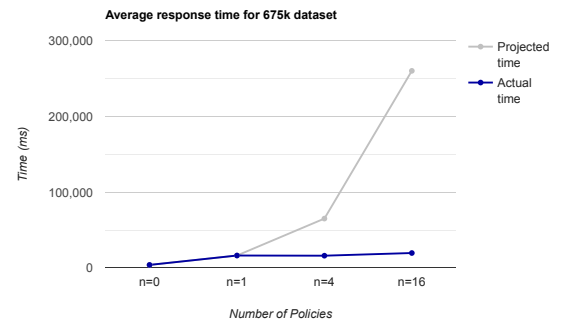
(a) 25 thousand triples.



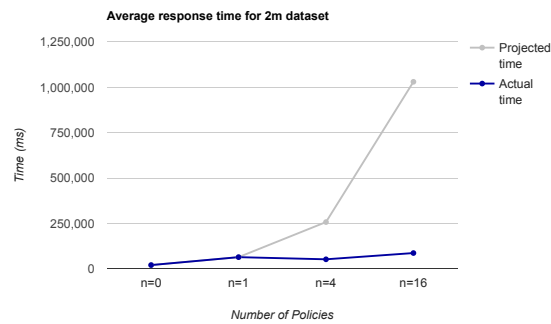
(b) 75 thousand triples.



(c) 225 thousand triples.

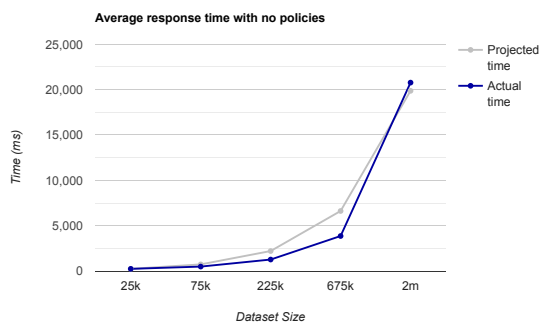


(d) 675 thousand triples.

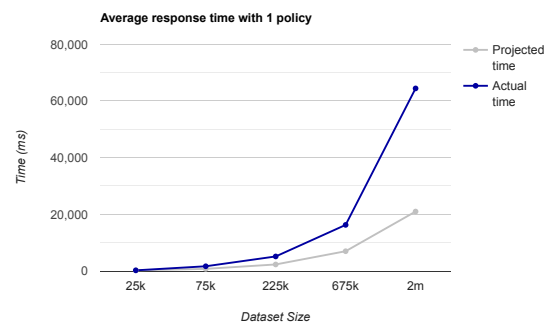


(e) 2 million triples.

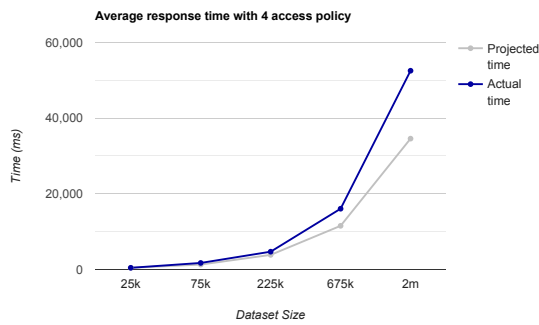
Figure 6.3: Projection and average response time by the number of policies in different dataset sizes.



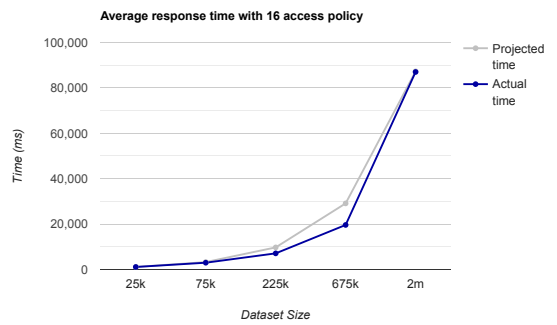
(a) No access policies.



(b) 1 access policy.



(c) 4 access policies.



(d) 16 access policies.

Figure 6.4: Projection and average response time at different dataset sizes for a different number of access policies.

and provide protection of multiple RDF graphs of a dataset, a detail that we overlooked since it was not considered a necessity for the archival domain although it may comprise a useful feature for other use cases. Nevertheless, we consider our solution to be aligned with the LDA and matched with its features and advantages. At the same time, since both platforms are based on an identical approach, the construction of temporary views, and implement similar mechanisms for the enforcement of access control, they also share the same scalability concerns. As pointed out by the authors of the LDA platform, and by the performance results presented in the previous section, strategies of this kind do not fare well with the increase of volume of protected data. Since both solutions were not subject to similar testing conditions, it is not possible to perform a direct comparison between the results of both other works; however, both displayed unpractical results around the threshold of 700k triples.

Another negative aspect that was noticeable during testing was the intricacy of the process for defining access policies for insert operations. As was pointed out earlier the definition of the target data for policies of this type is relative to data that is not yet present on the database, contrarily to policies of read and delete types. Thus, previewing its effect and determining the correctness of the policy was not as smooth as we hoped. This was even more troublesome if any kind of inference was used and partial insertions were disallowed. In such cases, for creating policies it was required complete knowledge about the vocabulary and the subsequent inference process, since there needs to exist policies explicitly allowing the insertion of all triples created through reasoning.

Chapter 7

Conclusions

7.1 Conclusions and Main Contributions

The implementation of access control mechanisms for the Semantic Web and linked data systems is a crucial aspect in many use cases of this concept. It is what empowers a system to hold private and public data securely and in some way limit the actions its users are allowed to complete. As we have seen, one of these use cases is the digital system from the Archival domain that demands a series of access control guarantees. For this use case and for similar scenarios, we were able to conclude through the analysis of existing technologies and solutions presented in Chapter 3 and the requirements elicited in Chapter 4 that there is not a viable solution, sufficiently mature to constitute a good, comprehensible answer for this particular problem. Considering the assumptions we had at the beginning of this work due to the high and rising number of use cases of Linked Data in need for fine-grained access control, there is an unexpected underdevelopment of the mechanisms of RDF store providers and an almost to nonexistent set of implemented and available solutions capable of fulfilling the needs of some scenarios. Attending that the first concerns with problems related to access control over linked data sources date back to the early 2000s, it is surprising there is still no evident approach and solution or a fully developed standard. In the research field, things have developed in that direction since there is a group of research works that aim to tackle scenarios with more fine-grained needs. However, only a few of them provide demonstrations of the implemented strategies. Faced with this scenario, where there is no implemented solution or concrete extendable work, we were forced to design and develop our own solution so that it would completely fit our needs.

In this work, we have taken steps to develop an access control solution for the systems of linked data Archives and propose and implement an answer to the need of data protection inside RDF graphs as well as insertion and deletion restrictions. Initially, we looked at the main components of an access control System, the Access Control Model, the Policy Representation, and the Policy Enforcement, and detailed their characteristics and the existing alternatives for each of them. Afterward, apart from inspecting the available solutions for performing access control in the most used services capable of storing RDF data, we identified and collected related research

works that tackle problems associated with the implementation of access control on linked data systems. This analysis had to be thorough since that, due to the lack of extendable options, it was essential that we comprehended the strategies implemented in works by other authors so that we could design and implement our own solution.

In Chapter 4, we discuss the access control requirements that have been collected, at a generic level, for linked data platforms by the works of the Working Group formed by the W3C but also reflected on the specific needs of the Archival domain. Such demands probed an important part of this work since they served as a guideline for the decisions made when developing a solution.

This solution is presented in Chapter 5, where we discuss its design and adaptation for the Archival domain and present the details of its implementation. For this, we based our implementation decisions on the outcome accomplished and described in some of the works analyzed earlier. Ultimately we produced a platform comprised of two software artifacts, one to fit inside a Spring Boot application and empower it with access control capabilities over an RDF graph, and another that amounts to a client in the shape of a Web application to make use of some of the features the former provides. Regarding the platform's characteristics, it is a service that includes certain procedures so that it provides fine-grained protection over the contents of graphs while also enabling restrictions for the execution of update operations in that context. The AC4LDA platform guarantees these features through the implementation of an enforcement mechanism that relies on temporary graphs to ensure it secures each one of the aforementioned actions. To do so, it utilizes RDF and RDF-S-based access policies that are previously specified and describe the conditions that a given subject can access a determined object from the content of a linked data archive.

Nevertheless, one thing to note is that the implemented components of the AC4LDA are not tied to any particular use case. Because of that, despite being developed with the intent to fit the needs of the Archival domain, it as it can also be used in other contexts that are constrained by similar requisites.

Lastly, in Chapter 6 we perform an evaluation of the work developed by verifying its adequacy for the Archival domain and gauging in which scenarios its application is advantageous. Through this evaluation, we provide an introspection at a qualitative and quantitative level. In the first moment, we validate the solution by putting it against the requirements set in Chapter 4, to afterward measure its performance in various distinct conditions. While this evaluation served to validate the outcome of this work, it also exposed some limitations such as the authentication layer that is currently lacking from the platform, the confusing rationale behind the definition of Access Policies relative to actions of inserting new data, and the sub-optimal results when the volume of protected data is increased. Still, we produced a generic platform that, overall, we believe successfully supports the access control needs of the Archival domain.

7.2 Further Work

Despite considering that the main objective of this work was successful given that, as the results of the evaluation detailed in the previous chapter, we provided a complete solution and the outcome

comprises an acceptable and good enough answer to the needs of a linked data archive system, there are still aspects to be improved.

Even though we attained the main goals of this work, our validation and evaluation evidenced some limitations that make up candidates points for future work. This group includes the implementation of a procedure that handles the authentication of the requesting subject, something that we had planned but could not produce while maintaining the AC4LDA Package's characteristics of being flexible and extensible. Furthermore, future work could include implementing changes in the system architecture or applying different technical and non-technical methodologies to improve the system's overall performance in order to abolish the restraints pointed out by the tests conducted in Chapter 6. The first comprises integrating a strategy that scales more efficiently for calculating graph operations since they are at the core of the most time-consuming processes. For instance, replacing the triplestore currently used for a tool or graph-storing service built for high scalability should improve performance results. Regarding the changes in the methodologies, these include maintaining user session information and a caching system so that the calculation of graphs is not so often required. Thus, obtaining an improved performance of the service.

Finally, another aspect that can be improved is the conflict resolution strategy. Allowing the specification of both positive and negative policies makes the conflict between policies possible. In this area, the algorithm for resolving the decision of conflicting policies has room for improvement. Since we follow a defaulting strategy where access to everything that is not explicitly allowed is denied, and we give priority to "deny" policies, we create a solution that has less expressiveness than what it could have. For example, if we consider two contradicting policies for the same context where the data protected by the positive one is a subset of the data protected by the negative, we know that through the methods present in our system, it will always result in the subject being unable to access both sets of data. Sometimes, this may not be the desired behavior, as one may expect that the positive policy overrides the negative one. For that reason, a methodology that associates precedence values with the access policies of the platform would comprise a valuable feature.

References

- [1] Wikimedia, “Semantic Web Stack,” https://commons.wikimedia.org/wiki/File:Semantic_Web_Stack.png, 2012, (accessed Jan. 30, 2023).
- [2] G. Schreiber and Y. Raimond, “RDF 1.1 Primer,” <https://www.w3.org/TR/rdf11-primer/>, W3C, Tech. Rep., 2014, (accessed Jan. 30, 2023).
- [3] D. Brickley, “foaf-basic.png,” <https://www.w3.org/2001/sw/Europe/talks/leon/foaf-basic.png>, (accessed Jan. 30, 2023).
- [4] CIDOC-CRM, “CIDOC CRM Class Hierarchy | CIDOC-CRM,” <https://www.cidoc-crm.org/FunctionalUnits/cidoc-crm-class-hierarchy>, (accessed Jan. 30, 2023).
- [5] DGARQ - Direção-Geral de Arquivos, “requerimento do Conde de Sampaio - Arquivo Nacional da Torre do Tombo - DigitArq,” <https://digitarq.arquivos.pt/details?id=4410531>, (accessed Feb. 27, 2023).
- [6] L. Costabello, S. Villata, N. Delaforge, and F. Gandon, “Linked data access goes mobile: Context-aware authorization for graph stores,” in *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*, ser. CEUR Workshop Proceedings, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., vol. 937. CEUR-WS.org, 2012. [Online]. Available: <http://ceur-ws.org/Vol-937/ldow2012-paper-05.pdf>
- [7] S. Villata, N. Delaforge, and F. Gandon, “S4AC Vocabulary specification,” https://ns.inria.fr/s4ac/v2/s4ac_v2.html, Inria, Tech. Rep., Oct. 2011, (accessed Jan. 30, 2023).
- [8] L. Costabello, “PRISSMA Vocabulary specification,” https://ns.inria.fr/prissma/v2/prissma_v2.html, Wimmics, Tech. Rep., Dec. 2014, (accessed Jan. 30, 2023).
- [9] M. Schmidt, T. Hornung, M. Meier, C. Pinkel, and G. Lausen, “Sp²bench: A SPARQL performance benchmark,” in *Semantic Web Information Management - A Model-Based Perspective*, R. D. Virgilio, F. Giunchiglia, and L. Tanca, Eds. Springer, 2009, pp. 371–393. [Online]. Available: https://doi.org/10.1007/978-3-642-04329-1_16
- [10] F. Abel, J. L. D. Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla, “Enabling advanced and context-dependent access control in RDF stores,” in *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, ser. Lecture Notes in Computer Science, K. Aberer, K. Choi, N. F. Noy, D. Allemang, K. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., vol. 4825. Springer, 2007, pp. 1–14. [Online]. Available: https://doi.org/10.1007/978-3-540-76298-0_1

- [11] A. Gabillon and L. Letouzey, “A view based access control model for SPARQL,” in *Fourth International Conference on Network and System Security, NSS 2010, Melbourne, Victoria, Australia, September 1-3, 2010*, Y. Xiang, P. Samarati, J. Hu, W. Zhou, and A. Sadeghi, Eds. IEEE Computer Society, 2010, pp. 105–112. [Online]. Available: <https://doi.org/10.1109/NSS.2010.35>
- [12] S. Kirrane, A. Mileo, and S. Decker, “Applying DAC principles to the RDF graph data model,” in *Security and Privacy Protection in Information Processing Systems - 28th IFIP TC 11 International Conference, SEC 2013, Auckland, New Zealand, July 8-10, 2013. Proceedings*, ser. IFIP Advances in Information and Communication Technology, L. J. Janczewski, H. B. Wolfe, and S. Sheno, Eds., vol. 405. Springer, 2013, pp. 69–82. [Online]. Available: https://doi.org/10.1007/978-3-642-39218-4_6
- [13] T. Berners-Lee, “Realising the Full Potential of the Web,” W3C, <https://www.w3.org/1998/02/Potential.html>, February 1998, (accessed Feb. 27, 2023).
- [14] N. Choudhury, “World wide web and its journey from web 1.0 to web 4.0,” *International Journal of Computer Science and Information Technologies*, vol. 5, no. 6, pp. 8096–8100, 2014.
- [15] T. Berners-Lee, “The World Wide Web: Past, Present and Future,” W3C, <https://www.w3.org/People/Berners-Lee/1996/ppf.html>, August 1996, (accessed Feb. 27, 2023).
- [16] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.
- [17] T. Berners-Lee, “Linked Data - Design Issues,” W3C, <https://www.w3.org/DesignIssues/LinkedData.html>, July 2006, (accessed Feb. 27, 2023).
- [18] I. C. on Archives, “Archives are Accessible - Search the Map ! | International Council on Archives,” <https://www.ica.org/en/what-archive/archives-are-accessible-search-the-map>, (accessed Jan. 30, 2023).
- [19] J. P. McCrae, “The linked open data cloud,” <https://lod-cloud.net/>, (accessed Jan. 30, 2023).
- [20] L. Ehrlinger and W. Wöß, “Towards a definition of knowledge graphs,” in *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, ser. CEUR Workshop Proceedings, M. Martin, M. Cuquet, and E. Folmer, Eds., vol. 1695. CEUR-WS.org, 2016. [Online]. Available: <http://ceur-ws.org/Vol-1695/paper4.pdf>
- [21] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, “Dbpedia: A nucleus for a web of open data,” in *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, ser. Lecture Notes in Computer Science, K. Aberer, K. Choi, N. F. Noy, D. Allemang, K. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., vol. 4825. Springer, 2007, pp. 722–735. [Online]. Available: https://doi.org/10.1007/978-3-540-76298-0_52

- [22] A. Singhal, “Introducing the Knowledge Graph: things, not strings,” Google Blog, <https://www.blog.google/products/search/introducing-knowledge-graph-things-not/>, May 16 2012, (accessed Nov. 27, 2022).
- [23] A. Krishnan, “Making search easier: How Amazon’s Product Graph is helping customers find products more easily,” Amazon Blog, <https://blog.aboutamazon.com/innovation/making-search-easier>, August 17 2018, (accessed Nov. 27, 2022).
- [24] S. Shrivastava, “Bring rich knowledge of people, places, things and local businesses to your apps,” Bing Blogs, <https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps>, July 12 2017, (accessed Nov. 27, 2022).
- [25] S. Kirrane, A. Mileo, and S. Decker, “Access control and the resource description framework: A survey,” *Semantic Web*, vol. 8, no. 2, pp. 311–352, 2017. [Online]. Available: <https://doi.org/10.3233/SW-160236>
- [26] W3C, “Semantic Web - W3C,” <https://www.w3.org/standards/semanticweb/>, (accessed Jan. 30, 2023).
- [27] —, “Data - W3C,” <https://www.w3.org/standards/semanticweb/data>, (accessed Jan. 30, 2023).
- [28] J. M. Keil, “IRI, URI, URL, URN and their differences,” <https://fusion.cs.uni-jena.de/fusion/2016/11/18/iri-uri-url-urn-and-their-differences>, Nov. 2016, (accessed Jan. 30, 2023).
- [29] R. Cyganiak, D. Wood, and M. Lanthaler, “RDF 1.1 Concepts and Abstract Syntax,” <https://www.w3.org/TR/rdf11-concepts/>, W3C, Tech. Rep., 2014, (accessed Jan. 30, 2023).
- [30] D. Brickley and R. Guha, “RDF Schema 1.1,” <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>, W3C, Tech. Rep., 2014, (accessed Jan. 30, 2023).
- [31] D. Brickley and L. Miller, “FOAF Vocabulary Specification,” FOAF Project, Namespace Document 2 Sept 2004, 2004, <http://xmlns.com/foaf/0.1/>. [Online]. Available: <http://xmlns.com/foaf/0.1/>
- [32] E. Prud’hommeaux and A. Seaborne, “SPARQL Query Language for RDF,” <https://www.w3.org/TR/rdf-sparql-query/>, W3C, Tech. Rep., 2008, (accessed Jan. 30, 2023).
- [33] P. Haase, J. Broekstra, A. Eberhart, and R. Volz, “A comparison of RDF query languages,” in *The Semantic Web - ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, ser. Lecture Notes in Computer Science, S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds., vol. 3298. Springer, 2004, pp. 502–517. [Online]. Available: https://doi.org/10.1007/978-3-540-30475-3_35
- [34] The W3C SPARQL Working Group, “SPARQL 1.1 Overview,” <https://www.w3.org/TR/sparql11-overview/>, W3C, Tech. Rep., 2013, (accessed Jan. 30, 2023).
- [35] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder, and L. A. Stein, “OWL Web Ontology Language Reference,” World Wide Web Consortium (W3C), Recommendation, February 2004, see <http://www.w3.org/TR/owl-ref/>.

- [36] Cambridge Semantics, “Rdfs vs. owl,” <https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/>, 2023, (accessed Jan. 30, 2023).
- [37] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, “OWL 2 Web Ontology Language Primer (Second Edition),” <https://www.w3.org/TR/2012/REC-owl2-primer-20121211/>, W3C, Tech. Rep., 2012, (accessed Jan. 30, 2023).
- [38] W3C OWL Working Group, “OWL 2 Web Ontology Language Document Overview (Second Edition),” <https://www.w3.org/TR/owl2-overview>, W3C, Tech. Rep., 2012, (accessed Jan. 30, 2023).
- [39] P. Samarati and S. D. C. di Vimercati, “Access control: Policies, models, and mechanisms,” in *Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000]*, ser. Lecture Notes in Computer Science, R. Focardi and R. Gorrieri, Eds., vol. 2171. Springer, 2000, pp. 137–196. [Online]. Available: https://doi.org/10.1007/3-540-45608-2_3
- [40] OWASP, “Access control | owasp,” https://owasp.org/www-community/Access_Control, (accessed Jan. 30, 2023).
- [41] OASIS Open Foundation, “OASIS eXtensible Access Control Markup Language (XACML) TC | OASIS,” <https://www.oasis-open.org/committees/xacml>, (accessed Jan. 30, 2023).
- [42] R. Iannella and S. Villata, “ODRL Information Model 2.2,” <https://www.w3.org/TR/odrl-model/>, W3C, Tech. Rep., Feb. 2012, (accessed Jan. 30, 2023).
- [43] W3C, “WebAccessControl - W3C Wiki,” <https://www.w3.org/wiki/WebAccessControl>, (accessed Jan. 30, 2023).
- [44] —, “Linked Data Platform (LDP) Working Group,” <https://www.w3.org/groups/wg/ldp>, (accessed Jan. 30, 2023).
- [45] S. L. Osborn, “Mandatory access control and role-based access control revisited,” in *Proceedings of the Second Workshop on Role-Based Access Control, RBAC 1997, Fairfax, VA, USA, November 6-7, 1997*, C. E. Youman, E. J. Coyne, and T. Jaeger, Eds. ACM, 1997, pp. 31–40. [Online]. Available: <https://doi.org/10.1145/266741.266751>
- [46] E. Bertino and R. S. Sandhu, “Database security-concepts, approaches, and challenges,” *IEEE Trans. Dependable Secur. Comput.*, vol. 2, no. 1, pp. 2–19, 2005. [Online]. Available: <https://doi.org/10.1109/TDSC.2005.9>
- [47] R. S. Sandhu and P. Samarati, “Access control: principles and practice,” *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, 1994. [Online]. Available: <https://doi.org/10.1109/35.312842>
- [48] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-based access control: a multi-dimensional view,” in *10th Annual Computer Security Applications Conference, ACSAC 1994, 5-9 December, 1994 Orlando, FL, USA*. IEEE, 1994, pp. 54–62. [Online]. Available: <https://doi.org/10.1109/CSAC.1994.367293>

- [49] S. Dietzold and S. Auer, “Access Control on RDF Triple Stores from a Semantic Wiki Perspective,” in *Proc. of 2nd Workshop on Scripting for the Semantic Web at ESWC, Budva, Montenegro, June 12, 2006*, ser. CEUR Workshop Proceedings ISSN 1613-0073, C. Bizer, S. Auer, and L. Miller, Eds., vol. 183, June 2006. [Online]. Available: <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-181/paper3.pdf>
- [50] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, “Attribute-based access control,” *Computer*, vol. 48, no. 2, pp. 85–88, 2015. [Online]. Available: <https://doi.org/10.1109/MC.2015.33>
- [51] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *Handheld and Ubiquitous Computing, First International Symposium, HUC’99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*, ser. Lecture Notes in Computer Science, H. Gellersen, Ed., vol. 1707. Springer, 1999, pp. 304–307. [Online]. Available: https://doi.org/10.1007/3-540-48157-5_29
- [52] I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, and L. A. Stein, “DAML+OIL,” <http://www.daml.org/2001/03/daml+oil-index.html>, March 2001.
- [53] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean, “Swrl: A semantic web rule language combining owl and ruleml,” W3C Member submission 21 may 2004, Tech. Rep., 2004. [Online]. Available: <http://www.w3.org/Submission/SWRL/>
- [54] OASIS Open Foundation, “A Brief Introduction to XACML,” https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html, (accessed Jan. 30, 2023).
- [55] R. Ferrini and E. Bertino, “Supporting RBAC with XACML+OWL,” in *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 3-5, 2009, Proceedings*, B. Carminati and J. Joshi, Eds. ACM, 2009, pp. 145–154. [Online]. Available: <https://doi.org/10.1145/1542207.1542231>
- [56] R. Stojanov, S. Gramatikov, I. Mishkovski, and D. Trajanov, “Linked data authorization platform,” *IEEE Access*, vol. 6, pp. 1189–1213, 2018. [Online]. Available: <https://doi.org/10.1109/ACCESS.2017.2778029>
- [57] solid IT, “Db-engines ranking - popularity ranking of rdf stores,” <https://db-engines.com/en/ranking/rdf+store>, (accessed Jan. 30, 2023).
- [58] Apache Software Foundation, “Apache Jena Fuseki,” 2022, [Software]. Available at: <https://jena.apache.org/documentation/fuseki2/>.
- [59] “Apache Jena | Data Access Control for Fuseki,” <https://jena.apache.org/documentation/fuseki2/fuseki-data-access-control.html>, (accessed Jan. 30, 2023).
- [60] Eclipse Foundation, “RDF4J Server and Workbench | The Eclipse Foundation,” <https://rdf4j.org/documentation/tools/server-workbench/#access-rights-and-security>, (accessed Jan. 30, 2023).
- [61] Ontotext, “Access Control - GraphDB 10.1.0 documentation,” <https://graphdb.ontotext.com/documentation/10.1/access-control.html>, (accessed Jan. 30, 2023).

- [62] R. S. Sandhu, D. F. Ferraiolo, and D. R. Kuhn, “The NIST model for role-based access control: towards a unified standard,” in *Fifth ACM Workshop on Role-Based Access Control, RBAC 2000, Berlin, Germany, July 26-27, 2000*, K. Rebersburg, C. E. Youman, and V. Atluri, Eds. ACM, 2000, pp. 47–63. [Online]. Available: <https://doi.org/10.1145/344287.344301>
- [63] Stardog Union, “Security Model - Stardog Documentation Latest,” <https://docs.stardog.com/operating-stardog/security/security-model>, (accessed Jan. 30, 2023).
- [64] —, “Named Graph Security - Stardog Documentation Latest,” <https://docs.stardog.com/operating-stardog/security/named-graph-security>, (accessed Jan. 30, 2023).
- [65] —, “Fine Grained Security - Stardog Documentation Latest,” <https://docs.stardog.com/operating-stardog/security/fine-grained-security>, (accessed Jan. 30, 2023).
- [66] OpenLink Software, “1.4.34.What sort of security features are built into Virtuoso?” https://docs.openlinksw.com/virtuoso/virtuosofaq33_01/, 2023, (accessed Jan. 30, 2023).
- [67] Amazon, “Managing Access Using IAM Policies - Amazon Neptune,” <https://docs.aws.amazon.com/neptune/latest/userguide/security-iam-access-manage.html>, (accessed Jan. 30, 2023).
- [68] —, “Custom IAM data-access policy statements for Amazon Neptune - Amazon Neptune,” <https://docs.aws.amazon.com/neptune/latest/userguide/iam-data-access-policies.html>, (accessed Jan. 30, 2023).
- [69] N. Kodali, C. Farkas, and D. Wijesekera, “An authorization model for multimedia digital libraries,” *Int. J. Digit. Libr.*, vol. 4, no. 3, pp. 139–155, 2004. [Online]. Available: <https://doi.org/10.1007/s00799-004-0080-1>
- [70] L. Cirio, I. F. Cruz, and R. Tamassia, “A role and attribute based access control system using semantic web technologies,” in *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, OTM Academy Doctoral Consortium, MONET, OnToContent, ORM, PerSys, PPN, RDDS, SSWS, and SWWS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4806. Springer, 2007, pp. 1256–1266. [Online]. Available: https://doi.org/10.1007/978-3-540-76890-6_53
- [71] O. Sacco and A. Passant, “A privacy preference ontology (PPO) for linked data,” in *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, ser. CEUR Workshop Proceedings, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., vol. 813. CEUR-WS.org, 2011. [Online]. Available: <http://ceur-ws.org/Vol-813/ldow2011-paper01.pdf>
- [72] S. Villata, N. Delaforge, F. Gandon, and A. Gyrard, “An access control model for linked data,” in *On the Move to Meaningful Internet Systems: OTM 2011 Workshops - Confederated International Workshops and Posters: EI2N+NSF ICE, ICSP+INBAST, ISDE, ORM, OTMA, SWWS+MONET+SeDeS, and VADER 2011, Hersonissos, Crete, Greece, October 17-21, 2011. Proceedings*, ser. Lecture Notes in Computer Science, R. Meersman, T. S. Dillon, and P. Herrero, Eds., vol. 7046. Springer, 2011, pp. 454–463. [Online]. Available: https://doi.org/10.1007/978-3-642-25126-9_57

- [73] O. Sacco, A. Passant, and S. Decker, “An access control framework for the web of data,” in *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, Changsha, China, 16-18 November, 2011*. IEEE Computer Society, 2011, pp. 456–463. [Online]. Available: <https://doi.org/10.1109/TrustCom.2011.59>
- [74] L. Costabello, S. Villata, and F. Gandon, “Context-aware access control for RDF graph stores,” in *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31, 2012*, ser. Frontiers in Artificial Intelligence and Applications, L. D. Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds., vol. 242. IOS Press, 2012, pp. 282–287. [Online]. Available: <https://doi.org/10.3233/978-1-61499-098-7-282>
- [75] J. D. Fernández, S. Kirrane, A. Polleres, and S. Steyskal, “Self-enforcing access control for encrypted RDF,” in *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, Eds., vol. 10249, 2017, pp. 607–622. [Online]. Available: https://doi.org/10.1007/978-3-319-58068-5_37
- [76] F. I. Benaribi, M. Malki, and K. M. Faraoun, “Protecting sensitive data in web of data,” in *Advances in Model and Data Engineering in the Digitalization Era - MEDI 2021 International Workshops: DETECT, SIAS, CSMML, BIOC, HEDA, Tallinn, Estonia, June 21-23, 2021, Proceedings*, ser. Communications in Computer and Information Science, L. Bellatreche, G. A. Chernishev, A. Corral, S. Ouchani, and J. Vain, Eds., vol. 1481. Springer, 2021, pp. 152–165. [Online]. Available: https://doi.org/10.1007/978-3-030-87657-9_12
- [77] J. Li and W. K. Cheung, “Query rewriting for access control on semantic web,” in *Secure Data Management, 5th VLDB Workshop, SDM 2008, Auckland, New Zealand, August 24, 2008, Proceedings*, ser. Lecture Notes in Computer Science, W. Jonker and M. Petkovic, Eds., vol. 5159. Springer, 2008, pp. 151–168. [Online]. Available: https://doi.org/10.1007/978-3-540-85259-9_10
- [78] G. Flouris, I. Fundulaki, M. Michou, and G. Antoniou, “Controlling access to RDF graphs,” in *Future Internet - FIS 2010 - Third Future Internet Symposium, Berlin, Germany, September 20-22, 2010. Proceedings*, ser. Lecture Notes in Computer Science, A. Berre, A. Gómez-Pérez, K. Tutschku, and D. Fensel, Eds., vol. 6369. Springer, 2010, pp. 107–117. [Online]. Available: https://doi.org/10.1007/978-3-642-15877-3_12
- [79] H. Muhleisen, M. Kost, and J.-C. Freytag, “Swrl-based access policies for linked data,” *Procs of SPOT*, vol. 80, 2010.
- [80] J. Hollenbach, J. Presbrey, and T. Berners-Lee, “Using rdf metadata to enable access control on the social semantic web,” in *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, vol. 514, 2009, p. 167.
- [81] P. Reddivari, T. Finin, A. Joshi *et al.*, “Policy-based access control for an rdf store,” in *Proceedings of the IJCAI-07 workshop on semantic web for collaborative knowledge acquisition*, 2007.

- [82] S. Franzoni, P. Mazzoleni, and S. Valtolina, “Towards a fine-grained access control model and mechanisms for semantic databases,” in *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*. IEEE Computer Society, 2007, pp. 993–1000. [Online]. Available: <https://doi.org/10.1109/ICWS.2007.176>
- [83] N. Lopes, S. Kirrane, A. Zimmermann, A. Polleres, and A. Mileo, “A logic programming approach for access control over RDF,” in *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, ser. LIPIcs, A. Dovier and V. S. Costa, Eds., vol. 17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 381–392. [Online]. Available: <https://doi.org/10.4230/LIPIcs.ICLP.2012.381>
- [84] S. Kirrane, “Linked data with access control,” Ph.D. dissertation, Galway, Ireland, 1 2015. [Online]. Available: <https://penni.wu.ac.at/papers/PhDThesisLinkedDatawithAccessControl.pdf>
- [85] J. Unbehauen, M. Frommhold, and M. Martin, “Enforcing scalable authorization on SPARQL queries,” in *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS’16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, ser. CEUR Workshop Proceedings, M. Martin, M. Cuquet, and E. Folmer, Eds., vol. 1695. CEUR-WS.org, 2016. [Online]. Available: <http://ceur-ws.org/Vol-1695/paper38.pdf>
- [86] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia, “A general framework for representing, reasoning and querying with annotated semantic web data,” *J. Web Semant.*, vol. 11, pp. 72–95, 2012. [Online]. Available: <https://doi.org/10.1016/j.websem.2011.08.006>
- [87] S. Kirrane, N. Lopes, A. Mileo, and S. Decker, “Protect your RDF data!” in *Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings*, ser. Lecture Notes in Computer Science, H. Takeda, Y. Qu, R. Mizoguchi, and Y. Kitamura, Eds., vol. 7774. Springer, 2012, pp. 81–96. [Online]. Available: https://doi.org/10.1007/978-3-642-37996-3_6
- [88] C. Bizer and A. Schultz, “The berlin SPARQL benchmark,” *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009. [Online]. Available: <https://doi.org/10.4018/jswis.2009040101>
- [89] C. Bizer and A. Schultz, “Berlin SPARQL Benchmark,” <http://wbsg.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/>, Nov. 2013, (accessed Jan. 30, 2023).
- [90] A. Jain and C. Farkas, “Secure resource description framework: an access control model,” in *11th ACM Symposium on Access Control Models and Technologies, SACMAT 2006, Lake Tahoe, California, USA, June 7-9, 2006, Proceedings*, D. F. Ferraiolo and I. Ray, Eds. ACM, 2006, pp. 121–129. [Online]. Available: <https://doi.org/10.1145/1133058.1133076>
- [91] O. Sacco and J. G. Breslin, “PPO & PPM 2.0: extending the privacy preference framework to provide finer-grained access control for the web of data,” in *I-SEMANTICS 2012 - 8th International Conference on Semantic Systems, I-SEMANTICS ’12, Graz, Austria, September 5-7, 2012*, V. Presutti and H. S. Pinto, Eds. ACM, 2012, pp. 80–87. [Online]. Available: <https://doi.org/10.1145/2362499.2362511>

- [92] W3C, “LDP Access Control,” <https://www.w3.org/TR/ldp-acr/>, (accessed Jan. 30, 2023).
- [93] —, “AccessControl - Linked Data Platform,” <https://www.w3.org/2012/ldp/wiki/AccessControl>, (accessed Jan. 30, 2023).
- [94] EPISA, “EPISA,” <https://episa.inesctec.pt/>, accessed: Jan. 30, 2023.
- [95] “What are archives? | International Council on Archives,” <https://www.ica.org/en/what-archive>, (accessed Jan. 30, 2023).
- [96] M. Boyd, “Standards for archives,” *Bulletin of the Association for Information Science and Technology*, vol. 43, no. 2, pp. 24–28, 2017. [Online]. Available: <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/bul2.2017.1720430210>
- [97] CIDOC-CRM, “Home | CIDOC CRM,” <https://www.cidoc-crm.org/>, (accessed Jan. 30, 2023).
- [98] International Council on Archives, “Records in Contexts - Conceptual Model | International Council on Archives,” <https://www.ica.org/en/records-in-contexts-conceptual-model>, (accessed Jan. 30, 2023).
- [99] I. Koch, C. Ribeiro, and C. T. Lopes, “ArchOnto, a CIDOC-CRM-Based Linked Data Model for the Portuguese Archives,” in *Digital Libraries for Open Knowledge - 24th International Conference on Theory and Practice of Digital Libraries, TPDL 2020, Lyon, France, August 25-27, 2020, Proceedings*, ser. Lecture Notes in Computer Science, M. M. Hall, T. Mercun, T. Risse, and F. Duchateau, Eds., vol. 12246. Springer, 2020, pp. 133–146. [Online]. Available: https://doi.org/10.1007/978-3-030-54956-5_10
- [100] VMWare Inc, “Spring Boot,” 2023, [Software]. Available at: <https://spring.io/projects/spring-boot>.
- [101] Apache Software Foundation, “Apache Jena,” 2022, [Software]. Available at: <https://jena.apache.org/>.
- [102] Eclipse Foundation, “RDF4J SPARQLBuilder,” <https://rdf4j.org/documentation/tutorials/sparqlbuilder/>, 2022.
- [103] H. Knublauch, “SHACL and OWL Compared,” <https://spinrdf.org/shacl-and-owl.html>, (accessed Jan. 30, 2023).
- [104] Apache Software Foundation, “Apache Jena - Reasoners and rule engines: Jena inference support,” <https://jena.apache.org/documentation/inference/#validation>, (accessed Jan. 30, 2023).
- [105] D. Kontokostas, “Shapes constraint language (SHACL),” W3C, Tech. Rep., Jul. 2017. [Online]. Available: <https://www.w3.org/TR/shacl/>
- [106] G. Carothers, E. Prud’hommeaux, D. Beckett, and T. Berners-Lee, “RDF 1.1 turtle,” W3C, W3C Recommendation, Feb. 2014. [Online]. Available: <http://www.w3.org/TR/2014/REC-turtle-20140225/>
- [107] OmniTI Labs, “JSend,” 2022, [Software]. Available at: <https://github.com/omniti-labs/jsend>.

- [108] T. Otwell, “Laravel,” 2022, [Software]. Available at: <https://laravel.com/>.
- [109] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
- [110] M. Dowling, “Guzzle,” 2022, [Software]. Available at: <https://docs.guzzlephp.org/en/stable/>.
- [111] T. Otwell, “Blade Templates,” 2022, [Software]. Available at: <https://laravel.com/docs/master/blade>.
- [112] Bootstrap, “Bootstrap,” 2022, Available at: [Software].<https://getbootstrap.com/>.
- [113] Eonasdan, “Tempus Dominus,” 2022, [Software]. Available at: <https://github.com/Eonasdan/tempus-dominus>.
- [114] D. Stutz, “Multiselect,” 2022, [Software]. Available at: <https://github.com/davidstutz/bootstrap-multiselect>.
- [115] Triply, “Yasgui,” 2022, [Software]. Available at: <https://triply.cc/docs/yasgui/>.
- [116] W3C, “RdfStoreBenchmarking - W3C Wiki,” <https://www.w3.org/wiki/RdfStoreBenchmarking>, (accessed Jan. 30, 2023).
- [117] M. Morsey, J. Lehmann, S. Auer, and A. N. Ngomo, “Dbpedia SPARQL benchmark - performance assessment with real queries on real data,” in *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, ser. Lecture Notes in Computer Science, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, Eds., vol. 7031. Springer, 2011, pp. 454–469. [Online]. Available: https://doi.org/10.1007/978-3-642-25073-6_29
- [118] D. Inc, “Docker,” <https://www.docker.com/resources/what-container/>, 2023, (accessed Mar. 1, 2023).

Appendix A

Algorithms for Access Control Enforcement

This appendix presents the pseudo-code of the algorithms used for the process of access control enforcement. Initially is presented the algorithm for the construction of a view of allowed data and afterwards the enforcement of access control for insert and delete operations.

A.1 Creation of temporary views

```
1 Context (C) and set of AccessPolicies (P)
2
3 allowed = empty
4 denied = empty
5 final = empty
6
7 // Iterate through the set of policies
8 for policy in P
9     // Use only policies of type 'read'
10    if policy[access_privilege] != READ
11        continue
12    endif
13
14    policy_decision = false
15    initialized = false
16    // Iterate through the access conditions of each policy
17    for access_condition in policy[access_condition_set]
18        // Execute the condition's ASK query against the current context
19        result = askContext(C, access_condition[queryAsk])
20
21        // Initialize policy_decision
22        if initialized = false
23            policy_decision = result
```

```

24         endif
25
26         // Calculate the policy decision based on the type of the access condition
           set
27         if policy[access_condition_set][type] = DISJUNCTIVE
28             policy_decision = policy_decision | result;
29         else if policy[access_condition_set][type] = CONJUNCTIVE
30             policy_decision = policy_decision & result;
31         endif
32     end
33
34     // If the decision is positive, execute the policy's SPARQL CONSTRUCT query and
       append it to a set of triples according to the type of the policy
35     if policy_decision = true
36         if policy[type] = ALLOW
37             allowed = allowed + query(policy[queryConstruct])
38         else if policy[type] = DENY
39             denied = denied + query(policy[queryConstruct])
40         endif
41
42     // Exclude the denied set of triples from the allowed ones
43     final = allowed - denied
44 end

```

Listing A.1: Algorithm for the creation of temporary views.

A.2 Enforcement of Access Control on Insert Operations

```

1 Operation (O), Context (C), set of AccessPolicies (P) and Allowed Data Model (M)
2
3 // Apply operation and subtract Allowed Data Model (M) to the modifiedReadModel
4 modifiedReadModel = applyOperation(O, M)
5 insertedTriples = modifiedReadModel - M
6
7 allowed_insert = empty
8 denied_insert = empty
9
10 // Iterate through the set of policies
11 for policy in P
12     // Use only policies of type 'read'
13     if policy[access_privilege] != INSERT
14         continue
15     endif
16
17     policy_decision = false

```

```

18   initialized = false
19   // Iterate through the access conditions of each policy
20   for access_condition in policy[access_condition_set]
21       // Execute the condition's ASK query against the current context
22       result = askContext(C, access_condition[queryAsk])
23
24       // Initialize policy_decision
25       if initialized = false
26           policy_decision = result
27       endif
28
29       // Calculate the policy decision based on the type of the access condition
        set
30       if policy[access_condition_set][type] = DISJUNCTIVE
31           policy_decision = policy_decision | result;
32       else if policy[access_condition_set][type] = CONJUNCTIVE
33           policy_decision = policy_decision & result;
34       endif
35   end
36
37
38   // Apply CONSTRUCT query to the modifiedReadModel
39   selected_triples = query(modifiedReadModel, policy[queryConstruct])
40
41   // Determine if the any of the selectedTriples are in the insertedTriples
42   intersected_triples = intersection(selectedTriples, insertedTriples)
43
44   // If the decision is positive, execute the policy's SPARQL CONSTRUCT query and
        append it to a set of triples according to the type of the policy
45   if policy_decision = true
46       if policy[type] = ALLOW
47           allowed_insert = allowed_insert + intersected_triples
48       else if policy[type] = DENY
49           denied_insert = denied_insert + intersected_triples
50       endif
51
52   if denied_insert == empty && allowed_insert.containsAll(inserted_triples)
53       M = M + allowed_insert
54   end
55 end

```

Listing A.2: Algorithm for the enforcement of access control on insert operations.

A.3 Enforcement of Access Control on Delete Operations

```

1 Operation (O), Context (C), set of AccessPolicies (P) and Allowed Data Model (M)
2
3 // Apply operation and subtract the modifiedReadModel to the Allowed Data Model (M)
4 modifiedReadModel = applyOperation(O, M)
5 deletedTriples = M - modifiedReadModel
6
7 allowed_delete = empty
8 denied_delete = empty
9
10 // Iterate through the set of policies
11 for policy in P
12     // Use only policies of type 'read'
13     if policy[access_privilege] != INSERT
14         continue
15     endif
16
17     policy_decision = false
18     initialized = false
19     // Iterate through the access conditions of each policy
20     for access_condition in policy[access_condition_set]
21         // Execute the condition's ASK query against the current context
22         result = askContext(C, access_condition[queryAsk])
23
24         // Initialize policy_decision
25         if initialized = false
26             policy_decision = result
27         endif
28
29         // Calculate the policy decision based on the type of the access condition
30         set
31         if policy[access_condition_set][type] = DISJUNCTIVE
32             policy_decision = policy_decision | result;
33         else if policy[access_condition_set][type] = CONJUNCTIVE
34             policy_decision = policy_decision & result;
35         endif
36     end
37
38 // Apply CONSTRUCT query to the modifiedReadModel
39 selected_triples = query(modifiedReadModel, policy[queryConstruct])
40
41 // Determine if the any of the selectedTriples are in the deletedTriples
42 intersected_triples = intersection(selectedTriples, deletedTriples)
43
44 // If the decision is positive, execute the policy's SPARQL CONSTRUCT query and
45 // append it to a set of triples according to the type of the policy
46 if policy_decision = true
47     if policy[type] = ALLOW
48         allowed_delete = allowed_delete + intersected_triples

```

```
48     else if policy[type] = DENY
49         denied_delete = denied_delete + intersected_triples
50     endif
51
52     if denied_insert == empty && allowed_insert.containsAll(deleted_triples)
53         M = M - allowed_delete
54     end
55 end
```

Listing A.3: Algorithm for the enforcement of access control on delete operations.

Appendix B

Docker Configuration

This appendix presents the Docker configurations of the instance of the Apache Jena Fuseki triplestore used and the AC4LDA Admin service.

B.1 Docker

Docker¹ is a tool that enables the virtualization of an operating system with the aim of building a unit that packages software and enables compatibility across platforms and operating systems, and, more importantly, a standard setting for the software to run [118]. These units or, as they are called, containers, isolate the execution of the software and ultimately guarantee that the software's dependencies are met and no other variables affect the software, independently of the machine where it is being executed. For these motives, Docker eases the process of development, sharing, execution, and deployment of software packages which formed a legitimate argument for using it. In the AC4LDA platform, we used Docker to build, execute and deploy the Apache Jena Fuseki triplestore and the environment for the Admin GUI, as presented in listings B.1 and B.2. Additionally we used Compose² to the configuration file presented in Listing B.3, manage the execution of both services.

B.2 Apache Jena Fuseki

```
FROM openjdk:11

RUN set -eux; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        bash curl ca-certificates findutils coreutils pwgen \
    ; \
```

¹Docker, <https://www.docker.com/>

²Docker Compose, <https://docs.docker.com/compose/>

```

rm -rf /var/lib/apt/lists/*

ENV FUSEKI_VERSION=4.5.0
ENV JENA_VERSION=4.5.0
ENV ASF_MIRROR=http://www.apache.org/dyn/mirrors/mirrors.cgi?action=download&
    filename=
ENV ASF_ARCHIVE=http://archive.apache.org/dist/

# Config and data
VOLUME /fuseki
ENV FUSEKI_BASE /fuseki

# Installation folder
ENV FUSEKI_HOME /jena-fuseki
ENV JENA_HOME /jena

WORKDIR /tmp

RUN curl --fail --silent --show-error --retry-connrefused --retry 3 --output fuseki.
    tar.gz $ASF_ARCHIVE/jena/binaries/apache-jena-fuseki-$FUSEKI_VERSION.tar.gz && \
    \
    tar xzf fuseki.tar.gz && \
    mv apache-jena-fuseki* $FUSEKI_HOME && \
    rm fuseki.tar.gz* && \
    cd $FUSEKI_HOME && rm -rf fuseki.war && chmod 755 fuseki-server

RUN curl --fail --silent --show-error --retry-connrefused --retry 3 --output jena.
    tar.gz $ASF_ARCHIVE/jena/binaries/apache-jena-$JENA_VERSION.tar.gz && \
    tar xzf jena.tar.gz && \
    mv apache-jena-* $JENA_HOME && \
    rm jena.tar.gz*

# copy a custom 'shiro.ini' allowing connections from anywhere with auth
# this will allow to use the webapp to see any already created datasets
COPY shiro.ini $FUSEKI_BASE/shiro.ini
COPY config.ttl $FUSEKI_BASE/config.ttl
COPY init.sh /
RUN chmod 755 /init.sh; chmod +x /init.sh

WORKDIR $FUSEKI_HOME
EXPOSE 3030

ENTRYPOINT ["/bin/bash"]

```

Listing B.1: Dockerfile for the Apache Jena Fuseki

B.3 AC4LDA Admin

```
FROM php:7.4-cli

# Set working directory
WORKDIR /var/www/app/

# Copy composer.lock and composer.json
COPY composer.json /var/www/app/
COPY package-lock.json package.json /var/www/app/

# Update packages and install Laravel dependencies
RUN apt-get update && apt-get install -y \
    g++ \
    libicu-dev \
    libpq-dev \
    libzip-dev \
    zip \
    zlib1g-dev \
    curl \
    gnupg \
    && docker-php-ext-install \
    intl \
    opcache \
    pdo \
    pdo_pgsql \
    pgsql

# Install NodeJS
RUN curl -sL https://deb.nodesource.com/setup_14.x | bash -
RUN apt-get -y install nodejs
RUN npm install -g concurrently

# Clear cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Install composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/
    bin --filename=composer

# Add user for laravel application
RUN groupadd -g 1000 www
RUN useradd -u 1000 -ms /bin/bash -g www www

# Copy existing application directory contents
COPY . /var/www/app/
```

```
RUN rm -rf node_modules
RUN rm -rf vendor

# Copy existing application directory permissions
RUN chown root:root /var/www/
RUN chmod 755 /var/www/

RUN chown -R www:www /var/www/app/
RUN chmod -R 774 /var/www/app/

# Change current user to www
USER www

# Install app dependencies
RUN composer install
RUN npm install

EXPOSE 8000

# To deploy, replace server for 'php-fpm server'
ENTRYPOINT concurrently "npm run watch" "php artisan serve --host 0.0.0.0"
```

Listing B.2: Dockerfile for the AC4LDA Admin GUI

B.4 docker-compose.yml

```
version: '3'
services:
  fuseki:
    container_name: ac4lda-fuseki
    build:
      context: ../fuseki
      dockerfile: Dockerfile
    command: [ "/init.sh", "-tdb2", "--config=/fuseki/config.ttl" ]
    ports:
      - 3031:3030
    restart: on-failure

  app:
    container_name: ac4lda_admin
    build:
      context: ../admin/app
    ports:
      - 8001:8000
    volumes:
      - ./app:/var/www/app/
```

```
- /var/www/app/node_modules/  
- /var/www/app/vendor/
```

Listing B.3: docker-compose.yml file for starting Apache Jena Fuseki and the Admin GUI

Appendix C

AC4LDA REST API Reference

This appendix presents the full OpenAPI specification of the AC4LDA REST API. Here are provided the details pertaining to the API's endpoints, the required format for requests and the format of its responses. A compiled version of this specification can be visualized in SwaggerHub¹.

```
openapi: 3.0.0

info:
  description: This specification describes the REST API provided by the AC4LDA
    Package
  version: "1.0.0"
  title: AC4LDA REST API
  contact:
    email: up201705985@edu.fe.up.pt
  license:
    name: MIT
    url: 'https://www.mit.edu/~amini/LICENSE.md'
tags:
  - name: /ac4lda

paths:
  /policies:
    get:
      tags:
        - /ac4lda
      summary: get all policies
      operationId: getPolicies
      description: Gets all policies in the system
      responses:
        '200':
          description: all policies in the system
          content:
            application/json:
```

¹AC4LDA REST API Reference, <https://app.swaggerhub.com/apis/tiagosilva/ac4lda/1.0.0>

```
      schema:
        type: array
        items:
          $ref: '#/components/schemas/Policy'
    '500':
      description: server error
/policy:
  get:
    tags:
      - /ac4lda
    summary: get policy information
    operationId: getPolicy
    description: |
      By passing in the appropriate options, you can get the information for
      an available policy in the system
    parameters:
      - in: query
        name: uuid
        description: pass an id string for looking up in the system
        required: true
        schema:
          type: string
    responses:
      '200':
        description: policy matching passed id
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Policy'
      '400':
        description: bad input parameter
      '500':
        description: server error
  post:
    tags:
      - /ac4lda
    summary: adds a policy
    operationId: addPolicy
    description: Adds a policy to the system
    responses:
      '201':
        description: policy created
      '400':
        description: 'invalid input, object invalid'
      '500':
        description: 'server error'
    requestBody:
      content:
        application/json:
```

```

    schema:
      type: object
      required:
        - type
        - name
        - appliedTo
      properties:
        name:
          type: string
          example: 'Allow read access to all users'
        appliedTo:
          type: string
          example: 'CONSTRUCT WHERE {?s ?p ?o}'
        type:
          type: string
          enum: ["allow", "deny"]
          example: 'allow'
        accessPrivileges:
          $ref: '#/components/schemas/Access_Previleges'
        accessConditionsSet:
          $ref: '#/components/schemas/Access_Condition_Set'
      description: Policy to add
  put:
    tags:
      - /ac4lda
    summary: updates a policy
    operationId: updatePolicy
    description: Updates a policy in the system
    parameters:
      - in: query
        name: policyId
        description: pass an id string for looking up in the system
        required: true
        schema:
          type: string
    responses:
      '201':
        description: policy updated
      '400':
        description: 'invalid input, object invalid'
      '500':
        description: 'server error'
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Policy'
      description: Policy to add
  delete:

```

```
tags:
  - /ac4lda
summary: deletes a policy
operationId: deletePolicy
description: Deletes a policy in the system
parameters:
  - in: query
    name: uuid
    description: pass an id string for looking up in the system
    required: true
    schema:
      type: string
responses:
  '201':
    description: policy deleted
  '400':
    description: 'invalid input, object invalid'
  '500':
    description: 'server error'

/users:
  get:
    tags:
      - /ac4lda
    summary: get all users
    operationId: getUsers
    description: Gets all users in the system
    responses:
      '200':
        description: all users in the system
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/User'
      '500':
        description: server error

/user:
  get:
    tags:
      - /ac4lda
    summary: get user information
    operationId: getUser
    description: |
      By passing in the appropriate parameter, you can get the information for
      an available user in the system
```

```
parameters:
  - in: query
    name: uuid
    description: pass an id string for looking up in the system
    required: true
    schema:
      type: string
responses:
  '200':
    description: user matching passed id
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/User'
  '400':
    description: bad input parameter
  '500':
    description: server error
post:
  tags:
    - /ac4lda
  summary: adds a user
  operationId: addUser
  description: Adds a user to the system
  responses:
    '201':
      description: user created
    '400':
      description: 'invalid input, object invalid'
    '500':
      description: 'server error'
  requestBody:
    content:
      application/json:
        schema:
          type: object
          properties:
            username:
              type: string
            roles:
              type: array
              items:
                $ref: '#/components/schemas/Role'
          description: User to add
delete:
  tags:
    - /ac4lda
  summary: deletes a user
  operationId: deleteUser
```

```
description: Deletes a user from the system
parameters:
  - in: query
    name: uuid
    description: pass an id string for looking up in the system
    required: true
    schema:
      type: string
responses:
  '201':
    description: policy deleted
  '400':
    description: 'invalid input, object invalid'
  '500':
    description: 'server error'

/roles:
  get:
    tags:
      - /ac4lda
    summary: get all roles
    operationId: getRoles
    description: Gets all roles in the system
    responses:
      '200':
        description: all roles in the system
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Role'
      '500':
        description: server error

/role:
  get:
    tags:
      - /ac4lda
    summary: get role information
    operationId: getRole
    description: |
      By passing in the appropriate parameter, you can get the information for
      an available role in the system
    parameters:
      - in: query
        name: uuid
        description: pass an id string for looking up in the system
```

```
      required: true
      schema:
        type: string
    responses:
      '200':
        description: role matching passed id
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Role'
      '400':
        description: bad input parameter
      '500':
        description: server error
  post:
    tags:
      - /ac4lda
    summary: adds a role
    operationId: addRole
    description: Adds a role to the system
    responses:
      '201':
        description: role created
      '400':
        description: 'invalid input, object invalid'
      '500':
        description: 'server error'
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              name:
                type: string
            description: role to add
  put:
    tags:
      - /ac4lda
    summary: updates a policy
    operationId: updateRole
    description: Updates a policy in the system
    parameters:
      - in: query
        name: roleId
        description: pass an id string for looking up in the system
        required: true
        schema:
          type: string
```

```

responses:
  '201':
    description: role updated
  '400':
    description: 'invalid input, object invalid'
  '500':
    description: 'server error'
requestBody:
  content:
    application/json:
      schema:
        type: object
        properties:
          name:
            type: string
          description: updated role
delete:
  tags:
    - /ac4lda
  summary: deletes a role
  operationId: deleteRole
  description: Deletes a role from the system
  parameters:
    - in: query
      name: uuid
      description: pass an id string for looking up in the system
      required: true
      schema:
        type: string
  responses:
    '201':
      description: role deleted
    '400':
      description: 'invalid input, object invalid'
    '500':
      description: 'server error'

```

components:

```

schemas:
  Policy:
    type: object
    required:
      - uuid
      - type
      - name
      - appliedTo
    properties:
      uuid:
        type: string

```

```

    example: '123e4567-e89b-12d3-a456-426614174000'
  name:
    type: string
    example: 'Allow read access to all users'
  appliedTo:
    type: string
    example: 'CONSTRUCT WHERE {?s ?p ?o}'
  type:
    type: string
    enum: ["allow", "deny"]
    example: 'allow'
  accessPrivileges:
    $ref: '#/components/schemas/Access_Previleges'
  accessConditionsSet:
    $ref: '#/components/schemas/Access_Condition_Set'

Access_Previleges:
  type: object
  required:
    - read
    - update
    - delete
    - create
  properties:
    read:
      type: boolean
      example: true
    update:
      type: boolean
      example: true
    delete:
      type: boolean
      example: false
    create:
      type: boolean
      example: false

Access_Condition_Set:
  type: object
  required:
    - type
    - accessConditions
  properties:
    type:
      type: string
      enum: ["disjunctive", "conjunctive"]
    accessConditions:
      type: array
      items:

```

```
$ref: '#/components/schemas/Access_Condition'

Access_Condition:
  oneOf:
    - type: object
      required:
        - askQuery
      properties:
        askQuery:
          type: string
          example:
    - type: object
      properties:
        users:
          type: array
          items:
            type: string
        roles:
          type: array
          items:
            type: string
        fromISODateTime:
          type: string
          example: '2000-10-31T01:30:00'
        toISODateTime:
          type: string
          example: '2020-01-31T12:30:00'

Role:
  type: object
  properties:
    uuid:
      type: string
      example: '123e4567-e89b-12d3-a456-426614174000'
    name:
      type: string

User:
  type: array
  items:
    type: object
    properties:
      uuid:
        type: string
        example: '123e4567-e89b-12d3-a456-426614174000'
      username:
        type: string
      roles:
        type: array
```

```
items:  
  $ref: '#/components/schemas/Role'
```

Listing C.1: OpenAPI specification for the AC4LDA REST API

Appendix D

SHACL Constraints

This appendix presents the full specification of SHACL constraints applied to the policies dataset, relative to each of the concepts represented in the vocabulary used by the AC4LDA package.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 @prefix sh: <http://www.w3.org/ns/shacl#>
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>
5 @prefix s4ac: <http://ns.inria.fr/s4ac/v2#>
6 @prefix prisma: <http://ns.inria.fr/prisma/v2#>
7
8 @prefix ac4lda: <http://episa-labs.inesctec.pt/ac4lda#>
```

Listing D.1: Prefixes used in SHACL Constraints

```
1 # Access Policy
2 ac4lda:AccessPolicyShape
3   a sh:NodeShape ;
4   sh:targetClass s4ac:AccessPolicy ;
5
6   sh:property [
7     sh:path s4ac:hasType ;
8     sh:maxCount 1 ;
9     sh:class s4ac:AccessPolicyType ;
10  ] ;
11
12   sh:property [
13     sh:path s4ac:hasAccessPrivilege ;
14     sh:minCount 1 ;
15     sh:maxCount 4 ;
16     sh:node ac4lda:AccessPrivilegeShape ;
17  ] ;
18
```

```

19   sh:property [
20     sh:path s4ac:hasAccessConditionSet ;
21     sh:minCount 1 ;
22     sh:maxCount 1 ;
23     sh:class s4ac:AccessConditionSet ;
24     sh:nodeKind sh:IRI ;
25   ] ;
26
27   sh:property [
28     sh:path s4ac:appliesTo ;
29     sh:minCount 1 ;
30   ] .
31
32 # AccessPrivilegeShape
33 ac4lda:AccessPrivilegeShape
34   a sh:NodeShape ;
35
36   sh:property [
37     sh:path rdfs:subClassOf ;
38     sh:hasValue s4ac:AccessPrivilege
39   ] .

```

Listing D.2: Access Policy SHACL Constraints

```

1 # AccessConditionSet
2 ac4lda:AccessConditionSetShape
3   a sh:NodeShape ;
4   sh:targetClass s4ac:AccessConditionSet ;
5
6   sh:property [
7     sh:path s4ac:hasAccessCondition ;
8     sh:minCount 1 ;
9     sh:class s4ac:AccessCondition ;
10    sh:nodeKind sh:IRI ;
11  ] .

```

Listing D.3: Access Condition Set SHACL Constraints

```

1 # AccessCondition
2 ac4lda:AccessConditionShape
3   a sh:NodeShape ;
4   sh:targetClass s4ac:AccessCondition ;
5
6   sh:property [
7     sh:path s4ac:hasQueryAsk ;
8     sh:minCount 1 ;

```

```

9      sh:maxCount 1 ;
10  ] ;
11
12  sh:property [
13      sh:path s4ac:hasQueryAsk ;
14      sh:datatype xsd:string ;
15      sh:severity sh:Warning ;
16      sh:message "Ask Query is not of xsd:string datatype"@en ;
17  ] ;
18
19  sh:property [
20      sh:path prissma:user ;
21      sh:class prissma:User ;
22  ] ;
23
24  sh:property [
25      sh:path ac4lda:role ;
26      sh:class ac4lda:Role ;
27  ] ;
28
29  sh:property [
30      sh:path ac4lda:fromTime ;
31      sh:datatype xsd:dateTime ;
32      sh:maxCount 1 ;
33      sh:lessThan ac4lda:toTime
34  ] ;
35
36  sh:property [
37      sh:path ac4lda:toTime ;
38      sh:datatype xsd:dateTime ;
39      sh:maxCount 1 ;
40  ] ;
41
42  sh:or (
43      [
44          sh:path prissma:user ;
45          sh:minCount 1 ;
46      ]
47      [
48          sh:path ac4lda:role ;
49          sh:minCount 1 ;
50      ]
51  ) .

```

Listing D.4: Access Condition SHACL Constraints

```

1  # User
2  ac4lda:UserShape

```

```

3  a sh:NodeShape ;
4  sh:targetClass prisma:User ;
5
6  sh:property [
7    sh:path ac4lda:hasUsername ;
8    sh:datatype xsd:string ;
9    sh:minCount 1 ;
10   sh:maxCount 1 ;
11 ];
12
13 sh:property [
14   sh:path ac4lda:hasPassword ;
15   sh:datatype xsd:string ;
16   sh:minCount 1 ;
17   sh:maxCount 1 ;
18 ];
19
20 sh:property [
21   sh:path ac4lda:role ;
22   sh:class ac4lda:Role ;
23 ] .

```

Listing D.5: User SHACL Constraints

```

1  # Role
2  ac4lda:RoleShape
3  a sh:NodeShape ;
4  sh:targetClass ac4lda:Role ;
5
6  sh:property [
7    sh:path ac4lda:hasName ;
8    sh:datatype xsd:string ;
9    sh:minCount 1 ;
10   sh:maxCount 1 ;
11 ];
12
13 sh:property [
14   sh:path ac4lda:hasDescription ;
15   sh:datatype xsd:string ;
16   sh:maxCount 1 ;
17 ] .

```

Listing D.6: Role SHACL Constraints

```

1  # Environment
2  ac4lda:EnvironmentShape

```

```
3  a sh:NodeShape ;
4  sh:targetClass prisma:Environment ;
5
6  sh:property [
7    sh:path ac4lda:time ;
8    sh:datatype xsd:dateTime ;
9    sh:minCount 1 ;
10   sh:maxCount 1 ;
11 ] .
```

Listing D.7: Environment SHACL Constraints

```
1  # Context
2  ac4lda:ContextShape
3  a sh:NodeShape ;
4  sh:targetClass prisma:Context ;
5
6  sh:property [
7    sh:path prisma:user ;
8    sh:class prisma:User;
9    sh:minCount 1 ;
10   sh:maxCount 1 ;
11 ];
12
13 sh:property [
14   sh:path prisma:environment ;
15   sh:class prisma:Environment;
16   # sh:minCount 1 ;
17   # sh:maxCount 1 ;
18 ];
```

Listing D.8: Context SHACL Constraints

Appendix E

Performance Test Results

This appendix presents the results obtained in the performance evaluation described in Chapter 6.

Table E.1: Execution times for tests (ms).

Dataset size	Round	Number of policies			
		n = 0	n = 1	n = 4	n = 16
25k triples	1	710	578	720	1556
	2	156	202	359	843
	3	116	174	355	889
	4	122	187	368	1109
	5	122	155	333	992
75k triples	1	1524	2868	2524	3702
	2	226	1549	1472	2647
	3	249	1483	1640	2760
	4	235	1291	1448	2882
	5	235	1274	1471	2776
225k triples	1	3435	7667	5694	9046
	2	742	4881	4423	6684
	3	730	4723	4227	6607
	4	750	4000	4461	6720

Continue on the next page

Table E.1: Execution times for tests (ms) (cont.).

Dataset size	Round	Number of policies			
		n = 0	n = 1	n = 4	n = 16
675k triples	5	694	4511	4629	6317
	1	9865	25035	17187	21712
	2	2448	15232	14282	18652
	3	2407	14265	13631	18105
	4	2319	13920	17280	20925
	5	2272	12953	17882	18603
2m triples	1	43506	99596	63499	104841
	2	13990	59872	47803	84663
	3	15302	49447	51312	78602
	4	15544	54446	50895	83923
	5	15503	58701	49223	83150