NUI Galway
OÉ Gaillimh

Linked Data with Access Control

Sabrina Kirrane

Supervisors: **Dr. Alessandra Mileo and Prof. Dr. Stefan Decker**

Internal Examiner: **Dr. John Breslin**

External Examiner: **Prof. Dr. Tim Finin**

Dissertation submitted in pursuance of the degree of Doctor of Philosophy

Insight Centre for Data Analytics,
National University of Ireland, Galway / Ollscoil na hÉireann, Gaillimh

January 29, 2015

The research presented herein was supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), the Irish Research Council for Science, Engineering and Technology Enterprise Partnership Scheme and Storm Technology Ltd.

Dedicated to my husband Kenneth and our twin boys Josh & Noah.

Acknowledgements

I would like to thank everyone who has encouraged and assisted me throughout the course of my Ph.D. In particular, I would like to thank Alessandra and Stefan for their support and direction. Thanks to Tim and John for taking time out to review the thesis. A big thank you the URQ guys, especially Aidan, Nuno and Jürgen for their ongoing advice. Thanks to my colleagues in DERI, Storm and GMIT for their support. A special word of thanks to Karl Flannery for affording me the opportunity to pursue a Ph.D. Thanks to my family for their understanding and encouragement, especially my sister Rachel who made the write-up much less lonely. Finally, thanks to Kenneth for everything.

Abstract

The explosion of digital content and the heterogeneity of enterprise content sources have pushed existing data integration solutions to their boundaries. An alternative solution is to use the Resource Description Framework (RDF) together with the existing web infrastructure, commonly known as the Linked Data Web (LDW), as a means to integrate both public and private data. With the advent of SPARQL 1.1, it is possible not only to execute queries over the LDW, but also to use the SPARQL query language to maintain distributed graph data. However, such a decentralised architecture brings with it a number of additional challenges with respect to both data security and integrity. In this thesis, we focus on the problem of access control for the LDW. We are particularly interested in lifting both the data and the access control policies from existing line of business applications and enforcing and maintaining access control over linked data, irrespective of how it is published.

We start by proposing a lifting strategy, which can be used to extract both data and access control policies from relational databases. The access permissions are represented using an extension of the RDF model known as annotated RDF (aRDF), which allows contextual information to be associated with RDF data. By using aRDF domain operations it is possible to combine different annotations for the same triple and to infer new annotations based on RDF Schema (RDFS) inference rules. We demonstrate how the proposed modelling, together with a set of inference rules, can be used to provide support for commonly used access control models, such as Role-Based Access Control, Attribute Based Access Control and View Based Access Control.

With regards to the enforcement and administration of access control over RDF, we focus specifically on Discretionary Access Control (DAC). Given DAC allows users to delegate their permissions to others, it is particularly suitable for managing access control over distributed data. Although a number of authors demonstrate how they can used Semantic Web technology to represent DAC policies, the authors do not examine DAC from a data model perspective. In order to fill this gap, we provide a summary of access control requirements for the RDF data model, based on the different characteristics of the RDF data model compared to relational and tree data models. In order to support access control policy specification at the triple, resource, graph and schema level, authorisations are specified using graph patterns. In addition, we demonstrate how our proposed flexible graph based authorisation framework, which we call G-FAF, can be used to cater for the specification, administration and enforcement of DAC policies over linked data.

Given the proposed access control framework enforces access control at the query layer, when access to the requested RDF data is partially restricted, it is necessary to rewrite the query so that it behaves in the same manner as a query executed over a filtered dataset. Therefore, we propose a query rewriting strategy for both the SPARQL 1.1 query and update languages, that can be used to partially restrict access to unauthorised data. In addition, we demonstrate how a set of criteria, which was originally used to verify relational access control policies, can be adapted to ensure the correctness of access control over RDF via query rewriting.

Declaration

I declare that this thesis is composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Sabrina Kirrane

January 29, 2015

Published Work

The work presented in this thesis has been published in the following conference proceedings:

Sabrina Kirrane, Alessandra Mileo, Stefan Decker
Secure Manipulation of Linked Data
The 12th International Semantic Web Conference, ISWC 2013.
Full Paper Research Track

Sabrina Kirrane, Alessandra Mileo, Stefan Decker
Applying DAC principles to the RDF graph data model
The 28th IFIP TC-11 International Information Security and Privacy Conference, SEC 2013.
Full Paper Research Track

Sabrina Kirrane, Nuno Lopes, Alessandra Mileo, Stefan Decker
Protect Your RDF Data!
The 2nd Joint International Semantic Technology Conference, JIST 2012.
Full Paper Research Track

Nuno Lopes, **Sabrina Kirrane**, Antoine Zimmermann, Axel Polleres, Alessandra Mileo
A Logic Programming approach for Access Control over RDF
The 28th International Conference on Logic Programming, ICLP 2012.
Technical Communication (Joint work with Nuno Lopes)

Sabrina Kirrane
DC Proposal: Knowledge Based Access Control Policy Specification and Enforcement
The 10th International Semantic Web Conference, ISWC 2011.
Doctorial Consortium Paper

Contents

Chapter 1

Introduction

Data on the web and within the Enterprise is continuously increasing and despite advances in Information Technology, it is still difficult for individuals and organisations to find relevant information in a timely manner. This problem is further magnified when related information is segregated in different information systems. Integrating the information contained in such systems is necessary to support day-to-day analytics. For instance, the integration of both internal and external data pertaining to customers and competitors could give a company market advantage. Likewise, external information about the individual compounds used in a candidate drug may allow pharmaceutical companies to identify potential issues early in the drug development process. Although, it is possible to develop point-to-point integration solutions between information systems, such an approach is neither flexible nor scalable. An alternative solution is to use a graph model, such as the Resource Description Framework (RDF), which underpins the Linked Data Web (LDW), to represent and link information, in a manner which can be interrupted by both humans and machines.

The fundamental building block of the RDF data model is an RDF triple, which is used to represent a statement in the form of a *subject-predicate-object* expression, describing the relationship between two pieces of information. For example, the following triple is used to state that there is an entity who's name is `Joe Bloggs`:

```
<http://example.org/JBloggs> <http://xmlns.com/foaf/0.1/name> "Joe Bloggs"
```

The Internationalised Resource Identifier (IRI) `http://example.org/JBloggs` is used to uniquely identify `Joe Bloggs`. The predicate `http://xmlns.com/foaf/0.1/name`, which is defined in the Friend of a Friend (FOAF) vocabulary, is used to state that the entity identified by the IRI `http://example.org/JBloggs` has a name `Joe Bloggs`. In addition to the `name` predicate, the FOAF vocabulary contains a range of predicates that can be used to describe people and their relationships with other people. FOAF is just one of a range of vocabularies, formally know as an *ontologies*, that can be used to describe information in a machine readable manner. Together RDF, IRIs and ontologies are used to represent information in a manner which can easily be integrated and extended.

Using RDF to represent information it is possible for an enterprise to perform large scale integration, search and analysis of both enterprise data and publicly accessible data. At the same time, using the existing LDW infrastructure it is possible for an enterprise to share data represented as RDF externally.

1.1 Background and Problem Statement

Although the technology underpinning the LDW has been in existence for a number of years, up until now data publishers have primarily focused on exposing and linking public data. With the advent of RDF update languages, such as SPARQL 1.1, the LDW has the potential to evolve from a medium for publishing and linking public data, to a dynamic read/write distributed data source. Such an infrastructure will be capable of supporting not only data integration of public and private data, but also the next generation of electronic business applications. However, in order to make the move from simply linking public data to using the Linked Data infrastructure as a global dataspace, we must first provide solutions for data security and integrity. The Semantic Web layer cake is a diagram which is used to depict the relationship between the technologies that are needed to realise the vision of the LDW. Trust an important component at the top of the Semantic Web layer cake. Here trust is an all encompassing term used also to relate to security and privacy policy enforcement.

In this thesis, we focus specifically on access control for the LDW. When it comes to access control, initially Semantic Web researchers focused on access control policy specification and enforcement using Semantic Technology. The term *policy* in this context, is used to refer to the access control model (which is the blueprint for restricting access to resources) and the policy language (which defines both the syntax and the semantics of the authorisations). Well known policy languages such as KAoS (Bradshaw et al., 1997), Rei (Kagal and Finin, 2003) and PROTUNE (Bonatti and Olmedilla, 2007) demonstrate how ontologies can be used to specify access control policies and how the corresponding enforcement frameworks can be used to restrict access to information systems, based on the defined policies. Several researchers focus on the specification and enforcement of access control policies over RDF data. Qin and Atluri (2003), Javanmardi et al. (2006b), Ryutov et al. (2009), and Amini and Jalili (2010) propose access control models for RDF graphs, that allow for policy propagation and inference based on semantic relations. Other researchers propose strategies for querying data residing in RDF stores securely (Abel et al., 2007; Chen and Stuckenschmidt, 2010; Flouris et al., 2010; Gabillon and Letouzey, 2010).

More recently the focus shifted to the specification and enforcement of access control over Linked Data. With, both Costabello et al. (2012a) and Sacco et al. (2011) proposing access control ontologies and enforcement frameworks that can be used to enforce access control over Linked Data. Although any of the solutions presented above could potentially be used to enforce access control over the Linked Data, to date no formal recommendation exists for access control for the LDW. As such, there is a need for (i) an investigation into the suitability of existing access control mechanisms to grant/deny access to RDF data; and (ii) an access control model and framework, which can be used to represent, enforce and administer, traditional and emerging access control models over distributed RDF data published as Linked Data.

1.2 Hypothesis

Interest in the LDW is growing, thanks to both the increased supply of machine readable information by data publishers and the demand for this information by data consumers. According to Heath and Bizer (2011), the LDW consists of information from a number of diverse domains (media, government, libraries, education, life sciences, retail and social), along with a number of cross domain datasets (for example, DBpedia[1], Freebase[2] and OpenCyc[3]). Like the early days of the web of documents, to date much of the focus has been on publishing and consuming public data, where access control is not a consideration. However, in order to support data analytics over public and private data and also to support the next generation of ebusiness applications, suitable forms of access control need to be put in place. With a view to determining the high level requirements for access control for the LDW, we examine how data is currently published (from the bottom up) and also how data is consumed (from the top down).

From a bottom up perspective, RDF data is primarily: (i) embedded in HTML pages or represented in static documents, that are served by regular web servers; (ii) persisted in RDF stores and exposed via web interfaces; or (iii) automatically generated from information, which is persisted in existing information systems.

RDF Documents. RDF data can be represented in RDF documents, using standard RDF syntax. Once the web server is configured to serve the relevant document type (for example, `.rdf, .n3` and `.ttl`), uploading an RDF document and serving it from a web server is relatively straightforward. Alternatively, RDF can be embedded in HTML documents. Using this approach it is not necessary to maintain two separate documents. However, for complex web documents care needs to be taken that the RDFa markup produces the desired output. Tools such as distillers and parsers can be used to parse HTML files and extract the RDF data.

Although it would be possible to use existing access control mechanisms that grant access to web documents or sets of documents, such an approach may result in an undesirable amount of duplication. An alternative would be to exploit existing RDF classification schemes, in order to cater for access control at multiple levels of granularity.

RDB2RDF. The relational data model, which came into being in 1970 (Codd, 1970), is still the predominant data storage mechanism for information systems almost half a century later. It is not surprising that considerable research has been conducted into the exposure of relational data as RDF, using Relational Database to RDF (RDB2RDF) mappings and a number of standards, that have been developed to provide guidance to developers (Arenas et al., 2012; Das et al., 2012).

When RDF data is published via RDB2RDF interfaces, it may also be necessary to extract the access rights, that were placed on the original relational data. One suggestion would be to use/extend the RDB2RDF technology, so that the permissions/prohibitions that were originally placed on the relational data, can be lifted and enforced over the corresponding RDF data.

SPARQL Endpoints. SPARQL is a query language for RDF, which is recommended by the World Wide Web Consortium (W3C). SPARQL 1.1 can be used to both query (Gearon et al., 2013) and update (Harris and Seaborne, 2013) RDF data. Although, to date, much of the focus has been on querying RDF data. The SPARQL protocol for RDF is a related W3C recommendation, which provides guidelines for the interaction between SPARQL query clients and engines, via existing web infrastructure.

Given that SPARQL can be used, not only as a means to query RDF data, but also as a means to maintain RDF datasets, it should be possible to enforce access control over the different SPARQL query and update operations.

[1]DBpedia,http://dbpedia.org/About
[2]Freebase, https://www.freebase.com/
[3]OpenCyc, http://www.cyc.com/platform/opencyc

From a top down perspective: (i) Linked Data can be explored using Linked Data browsers or queried using Linked Data search engines; and (ii) increasingly organisations are developing domain specific applications on top of Linked Data. A list of current Linked Data browsers and search engines, as well as several domain specific applications can be found on the linkeddata.org[4] website.

Browsers & Search Engines. Using Linked Data browsers, such as Tabular[5], Marbles[6] and Sig.ma[7], it is possible to explore the LDW as a whole and to navigate between different datasets, using IRI references. Like the web of documents, search engines, such as Sindice[8], Swoogle[9], Watson[10], enable users to execute queries over Linked Data. In addition SPARQL 1.1 supports federated search over multiple SPARQL endpoints.

Irrespective of whether the consumer is browsing or querying the LDW, they should only be able to access data which is either publicly accessible or data they have been authorised to access.

Domain Specific Applications. In recent years, numerous domain specific applications have been developed on top of Linked Data. Well known examples include the Talis Aspire university reading list application[11], the British Broadcasting Corporation (BBC) programmes website[12] and the Seevl music recommendation service[13].

Given the potential diversity of these domain specific applications, it is necessary to examine both traditional and emerging access control models, in order to determine the most effective means of protecting distributed RDF data.

The main hypothesis of this thesis can be summarised as follows:

Access control for the Linked Data Web can be achieved by (i) a representation format which can be used to express access control policies that are lifted from relational databases or associated directly with RDF data; (ii) a set of rules that simplify access control specification and maintenance; and (iii) an enforcement strategy which allows for the retrieval of partial query results.

Starting from the above hypothesis, we devise a number of discrete research questions:

(1) When relational data is exposed as RDF, how can we ensure the original access control policies are applied to the RDF data?

(2) Beyond triple level access control, what rules are necessary to (a) support existing access control models and (b) simplify access control specification and maintenance?

(3) What adjustments need to be made to SPARQL queries, to ensure that only authorised data is returned?

[4]linkeddata.org, http://linkeddata.org/tools
[5]Tabular,http://dig.csail.mit.edu/2005/ajar/ajaw/About.html
[6]Marbles, http://wiki.dbpedia.org/Marbles?v=71e
[7]Sig.ma,http://blog.sindice.com/2009/07/22/sigma-live-views-on-the-web-of-data/
[8]Sindice, http://www.sindice.com/
[9]Swoogle, http://swoogle.umbc.edu/
[10]Watson,http://watson.kmi.open.ac.uk/WatsonWUI/
[11]Talis Aspire, http://campus.talisaspire.com/
[12]BBC Programmes, http://www.bbc.co.uk/programmes
[13]Seevl, https://developer.seevl.fm/

(4) What components are required to support the specification, enforcement and administration of access control for the LDW?

1.3 Contributions

The work presented in this thesis have been published in a number of international conferences.

The initial research proposal, *DC Proposal: Knowledge Based Access Control Policy Specification and Enforcement* (Kirrane, 2011), was presented at the *International Semantic Web Conference (ISWC 2011) Doctoral Consortium*. The proposal identified the need for: a representation format capable of supporting one or more access control models; a policy language with underlying formal semantics capable of handling reasoning over access control policies; and an enforcement framework that can be used to administer and enforce both access control policies and inference rules.

Guided by the requirements identified in the research proposal, our early work focused on lifting both data and access control policies from existing systems. The inaugural paper *A Logic Programming approach for Access Control over RDF* (Lopes et al., 2012), which was presented at the *28th International Conference on Logic Programming (ICLP 2012)*, focuses specifically on an enterprise data integration use case and presents a formal access control model with built-in support for RDFS reasoning. The work relies on an extension of RDF, known as Annotated RDF, that allows domain specific meta-information (access rights in our case) to be attached to RDF triples. Domain operations are used to infer new annotations for triples derived using RDFS inference rules. Although such an approach works well when both the data and the access control policies are lifted from existing data sources, over large datasets it would not be feasible to manage permissions at a triple level. To tackle this issue, additional propagation rules are needed in order to simplify both the specification of new policies and the administration of existing policies. A follow up paper *Protect Your RDF Data* (Kirrane et al., 2013b), which was presented at the *2nd Joint International Semantic Technology Conference (JIST 2012)*, demonstrates how together annotations and rules can be used to represent a number of existing access control models. The paper builds on previous work by proposing a set of custom inference rules that can be used to reason over hierarchies of access control subjects, resources and access rights and by providing a high level overview of the components necessary for data integration and access control enforcement.

Having demonstrated how existing access control models can be represented and enforced, our focus turned to investigating the access control challenges encountered when data is represented as a graph. *Applying DAC principles to the RDF graph data model* (Kirrane et al., 2013c), which was presented at the *28th IFIP TC-11 International Information Security and Privacy Conference (SEC 2013)*, examines how Discretionary Access Control (DAC) principles, that have been successfully applied to the relational and the XML data models, can be applied to the RDF data model. A summary of access control requirements for graph data structures is provided; a list of access rights based on SPARQL query operations is proposed; a layered approach to authorisation derivation based on the graph structure and RDFSchema is recommended; and a number of rules that cater for the derivation of access rights based on class, property and instance relations are identified. A follow up paper, *Secure Manipulation of Linked Data* (Kirrane et al., 2013a), which was presented at the *12th International Semantic Web Conference (ISWC 2013)*, enhances the original triple level access control, by proposing a policy layer on top of the RDF query layer. The paper extends the original formalism to cater for authorisations based on quad patterns (where the fourth element is used to match the named graph); access rights based on SPARQL query operations; and demonstrates how generic propagation rules, conflict resolution policies and integrity constraints can together be used to ease maintenance and ensure data integrity.

We are currently working on a paper entitled *On the Correctness Criteria for Query Based Access Control over RDF*. This final paper proposes a query rewriting strategy, for complex SPARQL queries (graph patterns, filters, aggregates and subqueries), which ensures that in light of a given access control policy, as much data as possible is returned, without changing the semantics of the query. We adapt a set of correctness criteria, from relational databases, to allow for a comparison between our query rewriting strategy and an alternative data filtering approach. We demonstrate how a form of model checking (albeit not in the traditional sense) can be used to verify the correctness of the proposed query rewriting strategy. Finally we present an access control architecture for Linked Data, which we call LinDAA.

1.4 Research Impact

The access control framework and the query rewriting strategy can be used to securely integrate public and private data, using existing web infrastructure. Additionally, using the proposed access control strategy, it will be possible to use SPARQL 1.1 not only to query, but also to securely maintain Linked Data. The authorisation format and various inference rules, presented in this thesis, can be used to simplify access control specification and maintenance.

On a broader scale, the access control requirements we identify, correlate and categorise in the state of the art, serve not only as a means to compare existing access control strategies for RDF, but also as a roadmap for future research on access control for the LDW. Our analysis of the Discretionary Access Control (DAC) model and the corresponding proposal for applying DAC to distributed graph data, will provide a baseline for additional research into DAC over Linked Data. The proposed correctness criteria and model checking evaluation strategy can be used by other researchers to evaluate their access control strategies. Finally, over evaluation using the Berlin SPARQL Benchmark (BSBM) datasets can be used by others to benchmark their access control proposals.

1.5 Thesis Outline

Chapter 2 (Background) describes the necessary background information with respect to the syntax and the semantics of the RDF. We present an overview of SPARQL 1.1 and demonstrate how it can be used to both query and maintain RDF data. We introduce a fragment of RDFS, known as ρdf, which can be used to reason over RDF data. ρdf covers the essential features of RDFS and avoids vocabularies and axiomatic information, that only serve to reason about the structure of the language itself and not the data it describes. Following on from this, we highlight some of the limitations of RDFS and briefly discuss the Web Ontology Language (OWL). Finally we introduce the LDW and discuss how existing web technologies can be used to publish and consume Linked Data.

Chapter 3 (State of the Art) presents relevant access control models, standards and policy languages, and examines the different strategies for access control over RDF data that have been proposed to date. We provide a description of a number of access control models/standards and discuss how they have been applied to, or enhanced using, semantic technologies. Given that reasoning simplifies access control policy specification and maintenance, a logic based underlying formalisation is crucial for automated reasoning over access control policies. In the state of the art, we focus specifically on policy languages that use ontologies, rules or a combination of both to represent policies. In order to demonstrate the potential of the different approaches we provide a detailed description of several well known policy languages and frameworks. Following on from this, we examine the different access control strategies that have been proposed for RDF data, paying particular attention to the various reasoning and query rewriting proposals. Finally, we present a set of access control requirements for the LDW and categorise the various access control strategies accordingly.

Chapter 4 (Using Annotated RDFS for Access Control over Integrated RDF Data) investigates if triple based access control can be used to represent existing access control policies that are lifted from relational databases (***Research Question 1***). We demonstrate how XSPARQL, a transformation and query language for XML, RDB and RDF, can be used to extract both data and permissions from existing relational databases. We discuss how an extension of the RDF data model that associates contextual data with each triple, known formally as Annotated RDF, can be used to represent access control policies at a triple level, and to infer new policies for triples deduced using RDFS inference rules. In this chapter, we also investigate what rules are necessary to support a number of well known access control models, namely Role Based Access Control (RBAC), View-based Access Control (VBAC) and Attribute Based Access Control (ABAC) (***Research Question 2a***). A set of rules that allow for the propagation of permissions, based on the RDF data model and the subject, access right and resource hierarchies, are presented. In addition, a general RDF data integration and access control enforcement framework is proposed.

Chapter 5 (Applying Discretionary Access Control to Distributed RDF Data) builds on the previous chapter, by examining what rules are necessary to provide support for Discretionary Access Control (DAC) over RDF graph data (***Research Question 2a***). We examine how the graph data model differs from the relational and the tree data models, discuss how DAC can be applied to graph-based data and describe the implication such structural differences have on access control in general. A set of rules that cater for the derivation of access rights based on schema based relations (class, property and instance) are presented and a conflict resolution strategy is proposed. In this chapter, we also investigate what rules are necessary to simplify access control specification and maintenance (***Research Question 2b***). We demonstrate how the hierarchical Flexible Authorisation Framework, proposed by Jajodia et al. (2001), can be extended to provide DAC over graph data. We provide a formal definition for authorisations, propagation rules, conflict resolution policies and integrity constraints, within the context of RDF. In addition, we describe how together these components can simultaneously provide access control over interlinked RDF graphs. Finally we discuss how the extended framework can be used to enforce access control over SPARQL endpoints and detail the results of our performance evaluation.

Chapter 6 (Enforcing Access Control via Query Rewriting) investigates the security implications associated with granting partial access to RDF data, via SPARQL query rewriting (***Research Question 3***). In particular, we focus on complex SPARQL queries that include graph patterns, aggregates, subqueries, property paths and filters. We extend the enforcement algorithm presented in the previous chapter, to allow for partial results via query rewriting. We demonstrate how a set of correctness criteria, which was originally used by Wang et al. (2007), to verify relational access control policies, can be adapted to ensure the correctness of access control over RDF. We subsequently use the adapted criteria to compare our query rewriting approach against a filtering approach. In addition, we propose a benchmark, based on the BSBM dataset, that can be used to compare different strategies for access control for Linked Data. Finally, we investigate how the proposed access control mechanisms can be used to enforce access control over Linked Data (***Research Question 4***). We introduce our Linked Data Authorisation Architecture, which we call LinDAA, and discuss how it can be used to enforce access control over SPARQL endpoints, RDF documents, RDFa and RDB2RDF interfaces.

Chapter 7 (Conclusions) provides a critical assessment of the work presented in this thesis and proposes a number of directions for future work.

Chapter 2

Background

This chapter provides the background information on RDF, SPARQL, RDFS and Linked Data. We start by describing the RDF data model concepts, syntax and semantics. We demonstrate how SPARQL, the W3C recommended query language for RDF data, can be used to both query and update RDF data. We subsequently discuss the role played by RDF Schema when it comes to data modelling and reasoning. In addition, we highlight some of the limitation of RDFS and briefly introduce OWL. Finally, we discuss how these technologies, and others, can be used to realise the vision of a Linked Data Web.

2.1 The RDF Data Model

A *data model* is an abstraction used to represent real world entities, the relationship between these entities and the operations that can be performed on the data (Ullman, 1988). Data models can broadly be categorised as relational, tree and graph based. RDF, which is classified as a graph data model, is designed to capture data semantics and to present distributed data in a machine readable format.

2.1.1 Concepts and Syntax

Before introducing the core concepts of the RDF data model, it is first necessary to distinguish between *Uniform Resource Locators (URLs)*, *Uniform Resource Identifiers (URIs)* and *Internationalised Resource Identifiers (IRIs)*. As depicted in *Figure* 2.1, *URLs* are used to to identify web resources, *URIs* are used to give a unique web identifier to something that exists, and *IRIs* extend *URIs* by allowing for such web identifiers to be composed of Unicode characters.

The RDF 1.1 specification, which defines the core RDF concepts, is part of a suite of documents published as recommendations by the W3C (Cyganiak et al., 2014). *IRIs* and *literals* are used to represent information, which can be either physical or abstract in nature.

- *IRIs* are used to uniquely identify web resources. The RDF 1.0 specification (Klyne et al., 2004), which pre-dates the release of the *IRI* protocol, uses *RDF URIs* as opposed to *IRIs*. However unlike normal *URIs*, *RDF URIs* support Unicode characters.
- Literals are used to represent value data types, such as numbers, booleans and strings. Literals are composed of either two or three elements. A lexical form, a datatype *IRI* and in the case of a `languageString` datatype an optional language tag. A *simple literal* is a literal with neither a language tag nor a datatype *IRI*. By default simple literals are mapped to the `'http://www.w3.org/2001/XMLSchema#string'` datatype.

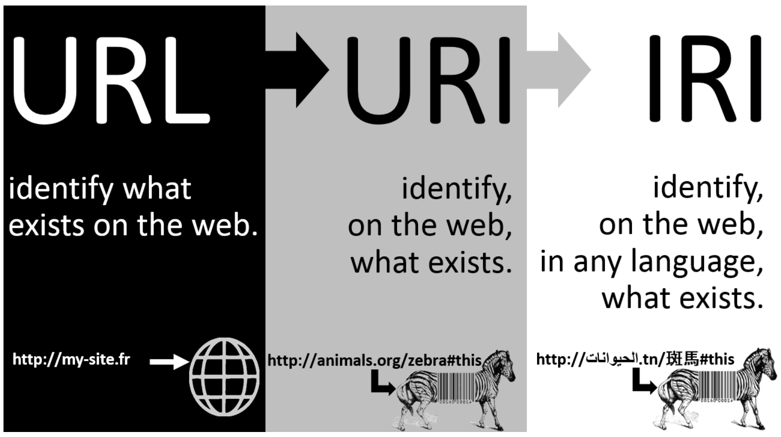

URL
URI
IRI
identify what exists on the web.
identify, on the web, what exists.
identify, on the web, in any language, what exists.
http://my-site.fr
http://animals.org/zebra#this
http://الحيوانات.tn/斑馬#this

Figure 2.1: Distinction between URLs, URIs and IRIs (Gandon, 2014)

RDF statements, formally known as *RDF triples*, are composed of *subject-predicate-object* expressions. Each triple asserts a binary relationship between two pieces of information. Whereas, *blank nodes* are used to make statements about unknown resources. Blank nodes are essentially existentially qualified variables, that are used to indicate that a resource exists, without providing explicit details about the resource.

The RDF data model was designed to facilitate data sharing and reuse. RDF *vocabularies* (otherwise known as *ontologies*) are collections of RDF triples that can be used to describe both schema and instance data. The `RDF` vocabulary is used to encode basic information pertaining to RDF, such as RDF type. Whereas, `FOAF`[1] is a well known vocabulary that is used to describe people and social relationship on the Web. Vocabularies are often placed in a common *namespaces*. For convenience *prefixes* are used as a shorthand notation for namespaces. In the examples presented in this thesis, we use the standard `rdf` and `foaf` prefixes:

```
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
foaf: <http://xmlns.com/foaf/0.1/>.
```

In addition we refer to a sample enterprise vocabulary, which is assigned the following prefix:

```
entx: <http://urq.deri.org/enterprisex/>.
```

Although RDF data can be encoded using a number of distinct representation formats (for example Notation3 (N3)[2], RDF/XML[3] or JSON-LD[4]) in this thesis we use N3 to represent triples and TriG[5], an extension of N3, which uses curly brackets to group triples into graphs identifiable by IRIs, to represent named graphs. In practice the triple and the named graph are stored as *quads*.

An RDF triple is formally defined as follows:

Definition 2.1 (RDF triple)
An RDF triple *is represented as a tuple* $\langle S, P, O\rangle \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$*, where* S *is called the* subject*,* P *the* predicate*, and* O *the* object *and* **I, B** *and* **L***, are used to represent* **IRIs***,* **blank nodes** *and* **literals** *respectively.*

[1]FOAF Vocabulary Specification, http://xmlns.com/foaf/spec/.
[2]Notation3, http://www.w3.org/TeamSubmission/n3/
[3]RDF/XML, http://www.w3.org/TR/rdf-syntax-grammar/
[4]JSON-LD, http://www.w3.org/TR/json-ld/
[5]http://www.w3.org/2010/01/Turtle/Trig/

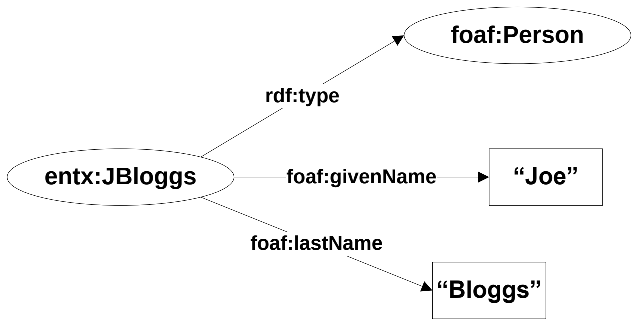


Figure 2.2: Triples represented as an RDF graph

Example 2.1 demonstrates how the `rdf` and `foaf` vocabularies can be used to indicate that `Joe Bloggs` is a person. Most RDF tools allow `rdf:type` and `a` to be used interchangeably.

Example 2.1 (RDF triple)
The following triple states that the `JoeBloggs` is a person:

```
entx:JoeBloggs rdf:type foaf:Person.
```

An *RDF graph* is a finite set of RDF triples, with subjects and objects represented as nodes and predicates represented as edges. A ground graph is a graph which does not contain blank nodes. An RDF graph is defined as follows:

Definition 2.2 (RDF graph)
Following the definition of an RDF triple*, an* RDF graph *G consists of a set of triples. The universe of G, universe(G), is the set of elements in $(\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ that occur in the triples of G and the* vocabulary *of G, voc(G), is universe(G) $\cap$ $(\mathbf{I} \cup \mathbf{L})$. We say that G is* ground *if and only if the universe(G) = voc(G), i.e. G does not contain blank nodes.*

Example 2.2 demonstrates how RDF can be used to represent additional information pertaining to `Joe Bloggs`. Whereas, *Figure* 2.2 demonstrates how these triples converge to form a graph, with IRIs represented as ovals and literals represented as rectangles.

Example 2.2 (RDF graph)
The following triples state that the `JoeBloggs` is a person whose first name is `Joe` and lastname is `Bloggs`:

```
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:givenName "Joe".
entx:JBloggs foaf:lastName "Bloggs".
```

Example 2.3 demonstrates how data pertaining to multiple employees can be stored in a single named graph. While, *Example* 2.4 demonstrates how the same information can be stored in two separate graphs. In *Figure* 2.3 we see that regardless of how the data is physically represented, RDF can logically be represented as a single graph.

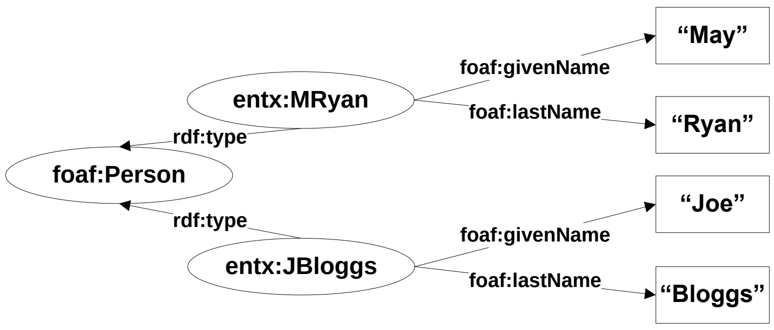


Figure 2.3: Named graph(s) represented as an RDF graph

Example 2.3 (Single named graph)
The following named graph is used to represent personal information pertaining to both `JoeBloggs` *and* `MayRyan` *in a single named graph.*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:givenName "Joe".
entx:JBloggs foaf:lastName "Bloggs".
entx:MRyan rdf:type  foaf:Person.
entx:MRyan foaf:givenName "May".
entx:MRyan foaf:lastName "Ryan".
}
```

Example 2.4 (Multiple named graphs)
The following named graphs are used to represent personal information pertaining to `Joe Bloggs` *and* `May Ryan` *in two separate named graphs.*

```
entx:EmployeeJBloggs {
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:givenName "Joe".
entx:JBloggs foaf:lastName "Bloggs".
}
entx:EmployeeMRyan {
entx:MRyan rdf:type  foaf:Person.
entx:MRyan foaf:givenName "May".
entx:MRyan foaf:lastName "Ryan".
}
```

A collection of RDF graphs, which can include a default graph and one or more named graphs is known as an RDF dataset, formally defined as follows:

Definition 2.3 (RDF dataset)
Given an RDF graph*, an* RDF dataset *DS consists of a set of graphs, with exactly one* ***default graph*** *possibly empty; and one or more* ***named graphs****, consisting of a pair* <name, RDF Graph>*, where* name $\in (\mathbf{I} \cup \mathbf{B})$.

2.1.2 Semantics

A model-theoretic semantics for RDF forms part of a separate document entitled RDF 1.1 Semantics (Hayes and Patel-Schneider, 2014). Model theory refers to a world and the conditions a world must satisfy in order to assign appropriate meaning for all expressions in a language. A particular world is called an interpretation. In the case of RDF an interpretation is a mapping from IRIs and literals into sets, and the specification of constraints with respect to both the sets and the mapping. A simple interpretation does not distinguish between different types of IRIs. A simple interpretation and truth for RDF graphs is defined as follows:

Definition 2.4 (Simple interpretation)
A simple interpretation I over a vocabulary V is a tuple $I = \langle IRes, IProp, IExt, IType, ILit \rangle$, where $\mathbf{I}$ *and* $\mathbf{L}$ *are used to represent IRIs and literals respectively, can be represented as follows:*

1. *$IRes$ is a non empty set of resources (called the domain or universe of I).*
2. *$IProp$ is a set of properties (not necessarily disjoint from $IRes$).*
3. *$IExt : IProp \rightarrow 2^{<IRes,IRes>}$, a mapping that assigns $IRes$ pairs to each IProp.*
4. *$IType$: $(\mathbf{I} \cup \mathbf{L}) \ \cap V \rightarrow IRes \ \cup \ IProp$, a mapping that maps IRIs and literals to elements in $IRes$ or $IProp$.*
5. *$ILit \subseteq IRes$ the set of all literal values.*

A simple interpretation of a ground RDF graph, called a simple model, is treated as a function from expressions (literals, IRIs, triples and graphs) to elements of the universe and truth values. If each expression evaluates to true then the interpretation of the graph evaluates to true. A simple model is defined as follows:

Definition 2.5 (Simple model)
An interpretation I is a model *of a ground graph G, denoted I a model of G, if for all elements E in G:*

1. *If E is a literal then I(E) = ILit(E).*
2. *If E is a IRI then I(E) = IType(E).*
3. *If E is a triple (s,p,o), if $I(p) \in IProp$ and <I(s),I(o)> $\in IExt(I(p))$ then I(E) = true, otherwise I(E) = false.*
4. *If E is an RDF graph, if I(E') = false for some triple E' in E then I(E) = false, otherwise I(E) = true.*

A simple model of the graph represented in *Figure* 2.2 is presented in *Example* 2.5.

Example 2.5 (Simple model)
Considering the graph from Figure 2.2, the corresponding vocabulary V consists of all of the nodes and edges of the graph. A simple interpretation I for this vocabulary would be as follows:

$IRes = \{\alpha, \beta, \texttt{"Joe"}, \texttt{"Bloggs"}, x, y, z\}$

$IProp = \{x, y, z\}$

$IExt = \{x\ \langle\alpha,\beta\rangle,\ y\ \langle\alpha,\texttt{"Joe"}\rangle,\ z\ \langle\alpha,\texttt{"Bloggs"}\rangle\}$

$IType = \{$ `entx:JoeBloggs` $\rightarrow \alpha,$
`foaf:Person` $\rightarrow \beta$,
`rdf:type` $\rightarrow x,$
`foaf:givenName` $\rightarrow y,$
`foaf:lastName` $\rightarrow z\ \}$

$ILit = \{$ `"Joe"`, `"Bloggs"` $\}$

As interpretation I verifies that all three triples of the graph are true, the graph as a whole is true.

$\langle\texttt{entx:JoeBloggs}, \texttt{foaf:Person}\rangle = \langle\alpha,\beta\rangle \in IExt(x) = IExt(\texttt{rdf:type})$

$\langle\texttt{entx:JoeBloggs}, \texttt{"Joe"}\rangle = \langle\alpha, \texttt{"Joe"}\rangle \in IExt(y) = IExt(\texttt{foaf:givenName})$

$\langle\texttt{entx:JoeBloggs}, \texttt{"Bloggs"}\rangle = \langle\alpha, \texttt{"Bloggs"}\rangle \in IExt(z) = IExt(\texttt{foaf:lastName})$

A semantic extension of a simple interpretation may extend these minimal truth conditions. However, it cannot modify or negate them. The RDF 1.1 Semantics Specification (Hayes and Patel-Schneider, 2014) provides a number of additional semantic conditions to cater for blank nodes and datatype literals. However, as the work presented in this thesis does not deal specifically with blank nodes or datatype literals these extensions are not presented. For additional details the reader is referred to the RDF 1.1. Semantics Specification (Hayes and Patel-Schneider, 2014).

2.2 Querying and Updating RDF Data using SPARQL

SPARQL, which has been standardised by the W3C, can be used to both query and update RDF data. SPARQL 1.0 (Seaborne and Prud'hommeaux, 2008) was released as an official W3C Recommendation in 2008. Its successor SPARQL 1.1 (W3C SPARQL Working Group, 2013), which extends the original query language, became an official W3C Recommendation in 2013. SPARQL 1.1 provides support for complex query operations (Gearon et al., 2013) and introduces the SPARQL update language (Harris and Seaborne, 2013). Although SPARQL 1.1 also includes enhancements to support service descriptions, federated queries, entailment reasoning and several results format, in this thesis we focus specifically on the query and update functionality. For details of these features the reader is referred to the SPARQL 1.1 overview document (W3C SPARQL Working Group, 2013).

SPARQL queries return solutions based on graph pattern matching. Triple patterns are triples that can potentially contain variables in the subject, predicate and object positions. Multiple

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

Figure 2.4: Employee named graph

```
entx:OrgStructure{
entx:MRyan entx:worksFor entx:JBloggs .
entx:JSmyth entx:worksFor entx:MRyan .
}
```

Figure 2.5: Organisation structure named graph

triple patterns, collectively know as a *graph pattern*, are used to match an RDF subgraph. The query results presented in this section are based on the data depicted in *Figure* 2.4 and *Figure* 2.5. Formally a triple pattern and a graph pattern are defined as follows:

Definition 2.6 (Triple pattern, graph pattern)
An RDF triple pattern is represented as a tuple $\langle S, P, O\rangle \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$*, where* **I, B** *and* **L***, are used to represent* **IRIs**, **blank nodes** *and* **literals** *respectively and* **V** *represents a set of* **variables** *disjoint from* $(\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$*.*

A graph pattern *is a set of* triple patterns.

Example 2.6 demonstrates how a triple pattern can be used to match an RDF subgraph.

Example 2.6 (Triple pattern)
A triple pattern *containing a variable ?s in the subject position is used to match all triples of type FOAF Person.*

```
?s rdf:type foaf:Person.
```

Considering the data presented in Figure 2.4, the bindings for variable ?s are as follows:

?s
entx:MRyan
entx:JSmyth
entx:JBloggs

Whereas, *Example* 2.7 demonstrates how a basic graph pattern can be used to match a subgraph.

Example 2.7 (Basic Graph pattern)
A basic graph pattern, containing two triple patterns, is used to return the id, name and salary of all FOAF Persons.

```
?id foaf:name ?name.
?id foaf:salary ?salary.
```

Considering the data presented in Figure 2.4, the bindings for the variables ?id ?name and ?salary, are as follows:

?id	***?name***	***?salary***
`entx:JSmyth`	`"John Smyth"`	`33000`
`entx:MRyan`	`"May Ryan"`	`33000`
`entx:JBloggs`	`"Joe Bloggs"`	`60000`

A SPARQL *graph pattern* is an extension of the basic graph pattern, which can be used to filter query results, return bindings for partially matched patterns and merge the results of two different patterns.
A SPARQL graph pattern is composed of:

- **Group graph patterns.** A set of graph patterns delimited by braces { }.
- **'.' operators.** The dot symbol is used to denote the conjunction of triples in a triple pattern.
- **`GRAPH` expressions.** When a graph keyword precedes a graph pattern the graph pattern is matched against the specified named graph.
- **`OPTIONAL` patterns.** An `OPTIONAL` keyword allows for partial graph pattern matching. If a triple pattern is declared `OPTIONAL` a result set is returned even if some of the variables are not bound.
- **`UNION` patterns.** Several alternative graph patterns can be matched to a single variable by using the `UNION` keyword to combine multiple graph patterns.
- **`FILTER` expressions.** Filters restrict solutions of a graph pattern according to a given constraint. `FILTER`s are composed of constants, elements of $(\mathbf{U} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$ (*see Definition* 2.6) comparison operators, logical connectives and built-in functions.
- **Subqueries.** Nested queries are used to limit the result set based on the results of an embedded query.
- **Property paths.** Property paths are used to match paths of arbitrary length between two graph nodes.

According to (Pérez et al., 2009) a SPARQL graph pattern is formally defined as follows:

Definition 2.7 (SPARQL graph pattern)
A SPARQL graph pattern is recursively defined

1. *a triple pattern is a graph pattern;*
2. *a graph pattern is a graph pattern;*
3. *if P and P' are graph patterns, then $(P\,.\,P')$, (P `OPTIONAL` P') and (P `UNION` P') are graph patterns;*
4. *if P is a graph pattern and F is a `FILTER` expression, then (P `FILTER` F) is a graph pattern;*
5. *if P is a graph pattern and $G \in (\mathbf{U} \cup \mathbf{V})$, then (`GRAPH` G P) is a graph pattern;*

For any pattern P, vars(P) is the set of variables occurring in P.

2.2.1 SPARQL Query Language

Considering the definition of a SPARQL graph pattern, presented in *Definition* 2.7, we describe the general syntax of a SPARQL query. Following on from this, we present the different query types supported by the SPARQL query language. Finally, we examine complex queries such as subqueries, property paths, negation and aggregates.

2.2.1.1 SPARQL Query Types

The SPARQL 1.1 query language (Gearon et al., 2013) has four distinct query forms (`ASK`, `SELECT`, `CONSTRUCT` and `DESCRIBE`). SPARQL graph patterns are used to bind variables to solutions and to generate result sets. In the case of:

- `SELECT` queries, the variables are projected;
- `ASK` queries, a boolean indicating if a given graph pattern is matched is returned; and
- `CONSTRUCT` and `DESCRIBE` queries, an RDF graph is constructed.

The `FROM`, `FROM NAMED` and `GRAPH` clauses are used to target specific named graphs. However, if the data resides in the default graph, these clauses may be omitted. The general syntax for the four distinct SPARQL query forms is defined as follows:

Definition 2.8 (SPARQL query)
Following on from the SPARQL graph pattern definition, a SPARQL query is compose of a query type (`SELECT, ASK, CONSTRUCT` and `DESCRIBE`), a `WHERE` clause, and optional `FROM`/`FROM NAMED` clauses. In the following query template `|` is used to represent or and `[ ]` are used to represent optional clauses.

`SELECT` *projections* | `ASK` | `CONSTRUCT` *{ BasicGraphPattern }* | `DESCRIBE` *projections*
`[ FROM` *NamedGraphIRI* `]`
`[ FROM NAMED` *NamedGraphIRI* `]`
`WHERE` *{ GraphPattern }*

SELECT Queries. The SPARQL `SELECT` query syntax, which is similar to SQL, consists of a `SELECT` clause identifying the variables to be bound, a `WHERE` clause which is used to match a subgraph and, optional `FROM` and `FROM NAMED` clauses that are used to target one or more graphs. The query presented in *Query* 2.1 uses a graph pattern with `id`, `name` and `salary` variables, to project the names and salaries of all FOAF Persons.

Query 2.1 (SELECT)
Given the following query:

```
SELECT ?id ?name ?salary
WHERE { GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary } }
```

The output is as follows:

?id	*?name*	*?salary*
entx:JBloggs	"Joe Bloggs"	60000
entx:MRyan	"May Ryan"	33000
entx:JSmyth	"John Smyth"	33000

ASK Query. Given `ASK` queries do not provide any variable bindings, the query syntax simply includes the `ASK` operator, a `WHERE` clause, and optional `FROM` and `FROM NAMED` clauses. As per the previous `SELECT` query, the `ASK` query presented in *Query* 2.2, is used to match a subgraph which contains the names and salaries for all FOAF Persons. If the subgraph is matched the `ASK` query returns yes, otherwise it returns no.

Query 2.2 (ASK)
Given the following query:

```
ASK
WHERE { GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary } }
```

The output is as follows:

yes

CONSTRUCT Queries. The SPARQL `CONSTRUCT` query syntax consists of a `CONSTRUCT` clause, composed of an RDF graph template, optional `FROM/FROM NAMED` clauses and a `WHERE` clause. In *Query* 2.3 a quad pattern, with a variable `?s` in the *subject* position, is used to match all FOAF Persons, and to create new triples that assert that these FOAF persons are employees.

Query 2.3 (CONSTRUCT)
Given the following query:

```
CONSTRUCT { ?s rdf:type entx:Employee }
WHERE { GRAPH entx:Employee { ?s a foaf:Person } }
```

continued overleaf ->

The output is as follows:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix entx:  <http://urq.deri.org/enterprisex#> .

entx:JSmyth a entx:Employee .
entx:JBloggs a entx:Employee .
entx:MRyan a entx:Employee .
```

DESCRIBE Queries. The query syntax consists of a `DESCRIBE` clause, with either a variable or an IRI, optional `FROM/FROM NAMED` clauses and a `WHERE` clause. In *Query* 2.4 a quad pattern with a variable `?s` in the *subject* position is used to return all information pertaining to FOAF persons that work for Joe Bloggs.

Query 2.4 (DESCRIBE)
Given the following query:

```
DESCRIBE ?s
WHERE { GRAPH entx:OrgStructure { ?s entx:worksFor entx:JBloggs } }
```

The output is as follows:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix entx:  <http://urq.deri.org/enterprise/> .

entx:MRyan  a           foaf:Person ;
        entx:salary     33000 ;
        entx:worksFor   entx:JBloggs ;
        foaf:name       "May Ryan" ;
        foaf:phone      "222-2222" .
```

2.2.1.2 Complex Graph Patterns

In the previous section, we presented the different SPARQL query forms. In this section, we take a closer look at graph pattern matching, with respect to aggregates, subqueries, negation and property paths.

Aggregates. Aggregates are functions that are applied to groups of solutions, for example `COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT` and `SAMPLE`. The query syntax is similar to a standard `SELECT` query however in the case of aggregates the `GROUP BY` clause may be used to specify solution groupings. In *Query* 2.5 `COUNT` and `AVG` are used to return the number of employees and the average salary of these employees, respectively.

Query 2.5 (Aggregate)
Given the following query:

```
SELECT ( COUNT(?s) AS ?numEmployees ) ( AVG(?salary) AS ?avgSalary )
WHERE { GRAPH ?g {
?s rdf:type foaf:Person . ?s entx:salary ?salary} }
```

The output is as follows:

?numEmployees	*?avgSalary*
3	42000.0

Subqueries. SPARQL subqueries are used to limit the result set based on the projections of an embedded query. In *Query* 2.6, we use an inner `SELECT` to return the names of all managers and their employees.

Query 2.6 (Subquery)
Given the following query:

```
SELECT DISTINCT  ?manager ?employee
WHERE { GRAPH ?g {
?x foaf:name ?employee . ?y foaf:name ?manager
{ SELECT ?x ?y WHERE { GRAPH ?g { ?x entx:worksFor ?y } } } } }
```

The output is as follows:

?manager	*?employee*
"May Ryan"	"John Smyth"
"Joe Bloggs"	"May Ryan"

Negation. In SPARQL, solutions can be removed from a result set using `FILTER EXISTS`, `FILTER NOT EXISTS` or `MINUS` expressions. Although subqueries are not classified under negation, like `FILTER EXISTS` operations, subqueries allow data to be filtered from the result set. In *Query* 2.7, we demonstrate how the `MINUS` clause can be used to filter out the names and the salaries of all FOAF Persons who earn more than 50000.

Query 2.7 (Negation)
Given the following query:

```
SELECT ?employee ?salary
WHERE { GRAPH ?g { ?x foaf:name ?employee . ?x entx:salary ?salary
MINUS { ?x entx:salary ?salary FILTER ( ?salary > 50000) } } }
```

The output is as follows:

?employee	*?salary*
"John Smyth"	33000
"Mary Ryan"	33000

Property Paths. A property path, which is represented using constructs similar to those used in regular expressions (for example, +, * and ?), is used to match a path of arbitrary length between two graph nodes. In *Query* 2.8, we demonstrate how property paths can be used not only to return the names of all managers and their employees, but also the employees that indirectly report to them.

Query 2.8 (Property path)
Given the following query:

```
SELECT DISTINCT ?manager ?employee
WHERE { GRAPH ?g1 { ?x foaf:name ?employee . ?y foaf:name ?manager
FILTER (?x != ?y)}
GRAPH ?g2 { ?x entx:worksFor* ?y } }
```

The output is as follows:

?manager	***?employee***
`"May Ryan"`	`"John Smyth"`
`"Joe Bloggs"`	`"Mary Ryan"`
`"Joe Bloggs"`	`"John Smyth"`

2.2.2 SPARQL Update

SPARQL 1.1 update caters for five update operations (`INSERT DATA`, `DELETE DATA`, `DELETE/ INSERT`, `LOAD` and `CLEAR`) and five graph management operations (`CREATE`, `DROP`, `COPY`, `MOVE` and `ADD`). The following examples are grouped according to their abstract syntax. If present the `SILENT` keyword is used to suppress any errors that are generated.

INSERT DATA and DELETE DATA The `INSERT DATA` and `DELETE DATA` operations are used to add triples to and remove triples from a graph. The general syntax for SPARQL `INSERT DATA` and `DELETE DATA` queries is as follows:

Definition 2.9 (INSERT DATA, DELETE DATA)
`INSERT DATA` | `DELETE DATA`
`[ GRAPH` *NamedGraphIRI* `]` *{ RDFGraph }*

To insert data into a named graph the `GRAPH` clause is used. If omitted the triples are inserted into the default graph. If the destination graph does not exist it is created implicitly. *Query* 2.9 demonstrates how data relating to Mike Murphy can be inserted into the employee graph.

Query 2.9 (INSERT DATA)

```
INSERT DATA
{ GRAPH entx:EmployeeDetails {
entx:MMurphy rdf:type foaf:Person .
entx:MMurphy foaf:name "Mike Murphy" .
entx:MMurphy entx:salary 45000 .
entx:MMurphy foaf:phone "444-4444"  } }
```

To delete data from a named graph the `GRAPH` clause is used. If omitted the triples are deleted from the default graph. If the delete results in an empty graph this graph may be implicitly removed. *Query* 2.10 demonstrates how data relating to Mike Murphy can be deleted from the employee graph.

Query 2.10 (DELETE DATA)
```
DELETE DATA
{ GRAPH entx:EmployeeDetails {
entx:MMurphy rdf:type foaf:Person .
entx:MMurphy foaf:name "Mike Murphy" .
entx:MMurphy entx:salary 45000 .
entx:MMurphy foaf:phone "444-4444"  } }
```

DELETE/INSERT. The `DELETE/INSERT` operation is used to remove triples from a graph, or add triples to a graph, based on bindings for a query pattern specified in a `WHERE` clause. The `WITH` clause defines the target graph. The `INSERT` and `DELETE` graph templates are updated based on the bindings for variables returned by the `WHERE` clause. It is feasible to have a `DELETE` without an `INSERT` and visa versa.

The syntax for SPARQL `DELETE/INSERT` queries is as follows:

Definition 2.10 (DELETE/INSERT)
`[ WITH` *NamedGraphIRI* `]`
`DELETE` *{ BasicGraphPattern }* `INSERT` *{ BasicGraphPattern }*
`| DELETE` *{ BasicGraphPattern }* `| INSERT` *{ BasicGraphPattern }*
`[ USING NAMED` *NamedGraphIRI* `]`
`[ WHERE` *{ GraphPattern }* `]`

Query 2.11 demonstrates how together the `INSERT` and `DELETE` operations can be used to the update the FOAF name Joe Bloggs to Joseph Bloggs.

Query 2.11 (DELETE/INSERT)
```
WITH entx:EmployeeDetails
DELETE { ?person foaf:name "Joe Bloggs" }
INSERT { ?person foaf:name "Joseph Bloggs" }
WHERE { ?person foaf:name "Joe Bloggs" }
```

LOAD. The `LOAD` operation imports all of the triples from an RDF document into an RDF graph. If the destination graph does not exist it is created, whereas if it does exist the data is simply added to the existing graph. If no destination graph is provided the triples are loaded into the default graph.

Definition 2.11 (LOAD)
`LOAD [ SILENT ]` *NamedGraphIRI* `INTO GRAPH` *NamedGraphIRI*

Query 2.12 demonstrates how data is loaded from an RDF document into an RDF graph.

Query 2.12 (LOAD)
```
LOAD <http://urq.deri.org/docs/EmployeeData> INTO GRAPH entx:
    EmployeeDetails
```

CLEAR and DROP. The `CLEAR` operation removes all triples from a given graph. While the `DROP` operation removes the existing graph and associated data. The `GRAPH` keyword is used to specify a graph based on a given IRI, the `DEFAULT` keyword is used to remove the triples from the default graph, the `NAMED` keyword is used to remove all triples in all named graphs and the `ALL` keyword is used to remove all triples from both the default graph and all of the named graphs.

Definition 2.12 (CLEAR and DROP)
`CLEAR | DROP [ SILENT ]`
`GRAPH` *NamedGraphIRI* `| DEFAULT | NAMED | ALL`

Query 2.13 demonstrates how all data can be removed from the `entx:EmployeeDetails` graph.

Query 2.13 (CLEAR)
```
CLEAR GRAPH entx:EmployeeDetails
```

CREATE. The `CREATE` operation creates a new graph with the name specified by the IRI.

Definition 2.13 (CREATE)
`CREATE [ SILENT ]`
`GRAPH` *NamedGraphIRI*

Query 2.14 simply creates a new `entx:StaffHolidays` graph.

Query 2.14 (CREATE)
```
CREATE GRAPH entx:StaffHolidays
```

COPY, MOVE and ADD. The `COPY` operation inserts all data from the source graph into the destination graph. This operation results in the data from the destination graph being removed before the move takes place. The `MOVE` operation moves all data from the source graph into the destination graph. Like the `COPY`, this operation results in the data from the destination graph being removed before the move takes place. However unlike the `COPY`, the data from the source graph is removed after all the data has been moved. The `ADD` operation inserts all data from the source graph into the destination graph. When this operation is used the destination graph is not cleared before the data is added and the source graph is not cleared after the data is added.

Definition 2.14 (COPY, MOVE and ADD)
`COPY | MOVE [ SILENT ]`
`GRAPH` *NamedGraphIRI* `| DEFAULT TO`
`GRAPH` *NamedGraphIRI* `| DEFAULT`

Query 2.15 demonstrates how data can be copied from the `entx:EmployeeDetails` graph to a new graph called `entx:EmployeeBackup` .

Query 2.15 (COPY)
```
COPY GRAPH entx:EmployeeDetails TO GRAPH entx:EmployeeBackup
```

2.3 Reasoning over RDF Data

In RDF there is a tight coupling between the schema and instance data. *RDFSchema* (RDFS) (Brickley and Guha, 2014) is a set of classes and properties used to describe RDF data. RDFS does not describe the structure of an RDF graph, but rather provides a framework used by vocabularies to denote classes, properties and relations. RDFS is composed of a set of classes that are used to define types of resources and properties that are used to describe these resources. Using the RDFS vocabulary it is possible to define class and property relations, similar to object oriented programming. The RDFS vocabulary which is defined in the following namespace, is conventionally assigned an `rdfs` prefix: `rdfs:<http://www.w3.org/2000/01/rdf-schema#>` .

2.3.1 The ρdf subset of RDFS

In this thesis, we adopt a fragment of RDFS called ρdf (Muñoz et al., 2009), which focuses on a set of RDFS properties (`rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain` and `rdfs:range`) that are designed to relate individual pieces of data. ρdf covers the essential features of RDFS and avoids vocabularies and axiomatic information that only serve to reason about the structure of the language itself and not the data it describes. The following is a summary of the different types of ρdf properties:
core properties

`rdf:type` indicates that a resource is an instance of a particular class.

`rdfs:subClassOf` defines a subsumption relation between classes.

`rdfs:subPropertyOf` defines a subsumption relation between properties.

restricting properties

`rdfs:domain` indicates that the subject of the property is an instance of the specified class.

`rdfs:range` indicates that the object of the property is an instance of the specified class.

2.3.1.1 ρdf Semantics

Muñoz et al. (2009) extend the simple interpretation and model presented in *Section* 2.1.2 so that the intended meaning of the ρdf vocabulary is unambiguous. In the model and the rules that follow we use a number of abbreviations: `type` for `rdf:type`, `sp` for `rdfs:subPropertyOf`, `sc` for `rdfs:subClassOf`, `dom` for `rdfs:domain` and `range` for `rdfs:range`. Muñoz et al. (2009) extend the simple interpretation for RDF graphs presented in *Definition* 2.4, as follows:

Definition 2.15 (ρdf Interpretation)
An interpretation I *over a vocabulary* V *is a tuple*
$I = \langle IRes, IProp, IClass, IExt, ICExt, IType, ILit \rangle$*, where:*

1. *IRes is a non empty set of resources (called the domain or universe of I);*

2. *IProp is a set of properties (not necessarily disjoint from IRes);*

3. $IClass \subseteq IRes$ *is a distinguishable subset of IRes denoting classes;*

4. $IExt : IProp \rightarrow 2^{<IRes,IRes>}$*, a mapping that assigns IRes pairs to each IProp.*

5. $ICExt : IClass \rightarrow 2^{z}$*, a mapping that assigns a set of resources z in IClass to every resource denoting a class;*

6. $IType : (\mathbf{I} \cup \mathbf{L}) \cap V \rightarrow IRes \cup IProp$*, a mapping that maps IRIs and literals to elements in IRes or IProp.*

7. *$ILit \subseteq IRes$ the set of all literal values.*

Given the extended interpretation, a model of a ground RDF graph is subsequently extended by Muñoz et al. as follows:

Definition 2.16 (ρdf Model)
An interpretation I is a model *of a ground graph G, denoted I a model of G, if and only if I is an interpretation over the vocabulary $\rho df \cup universe(G)$ that satisfies the following conditions:*

Simple:

1. *for each $(s, p, o) \in G$, $IType(p) \in IProp$ and $(IType(s), IType(o)) \in IExt(IType(p))$;*

Subproperty:

1. *IExt(IType(sp)) is transitive and reflexive over IProp;*
2. *if $(x, y) \in IExt(IType(sp))$ then $x, y \in IProp$ and $IExt(x) \subseteq IExt(y)$;*

Subclass:

1. *IExt(IType(sc)) is transitive and reflexive over IClass;*
2. *if $(x, y) \in IExt(IType(sc))$ then $x, y \in IClass$ and $ICExt(x) \subseteq ICExt(y)$;*

Typing I:

1. *$x \in ICExt(y) \implies (x, y) \in IExt(IType(type))$*
2. *if $(x, y) \in IExt(IType(dom))$ and $(u, v) \in IExt(x)$ then $u \in ICExt(y)$*
3. *if $(x, y) \in IExt(IType(range))$ and $(u, v) \in IExt(x)$ then $v \in ICExt(y)$*

Typing II:

1. *$e \in \rho df, IType(e) \in IProp$*
2. *if $(x, y) \in IExt(IType(dom))$ then $x \in IProp$ and $y \in IClass$*
3. *if $(x, y) \in IExt(IType(range))$ then $x \in IProp$ and $y \in IClass$*
4. *if $(x, y) \in IExt(IType(type))$ then $y \in IClass$*

2.3.1.2 ρdf Deductive System

Given the extended interpretation and model presented in the previous section, Muñoz et al. propose a sound and complete deductive system for ρdf. In the deduction rules that follow the premise is represented above the line and the conclusion is presented below the line.

$$\frac{P_1 ... P_n}{P}$$

Essentially a deduction rule states that if the propositions $P_1 ... P_n$ are verified to be true then we can deduce P is also true. In this thesis, we do not concern ourselves with reflexivity and therefore these rules are not presented. For additional information on reflexivity the reader is referred to Muñoz et al. (2009).

Prior to presenting the *ρdf* deductive system we first present the definition of a *map*, presented by Muñoz et al. (2009):

Definition 2.17 (Map)
Following the definition of an RDF Graph, a map *is a function* $\mu : (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}) \rightarrow (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ *preserving URIs and literals, i.e.* $\mu(t) = t$, *for all* $t \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$. *Given a graph* G, *we define* $\mu(G) = (\mu(\mathbf{s}), \mu(\mathbf{p}), \mu(\mathbf{o})) \mid (\mathbf{s}, \mathbf{p}, \mathbf{o}) \in \mathbf{G}$. *We speak of a map* μ *from* G_1 *to* G_2, *and write* $\mu : G_1 \rightarrow G_2$, *if* μ *is such that* $\mu(G_1) \subseteq G_2$. *We say that a map* μ *is a* grounding *of a graph* G, *iff* $\mu(G)$ *is a ground graph.*

Definition 2.18 (ρdf Deductive system)
In the rules that follow G *and* G' *are used to represent RDF graphs and* A, B, C, X *and* Y *are meta-variables used to represent elements in* $(\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$.

Simple:

$(a)\quad \frac{G}{G'}$ *for a map* $\theta : G' \rightarrow G \qquad (b)\quad \frac{G}{G'}$ *for* $G' \subseteq G$

Subproperty:

$$(a)\quad \frac{(A,\mathsf{sp},B),(B,\mathsf{sp},C)}{(A,\mathsf{sp},C)} \qquad (b)\quad \frac{(D,\mathsf{sp},E),(X,D,Y)}{(X,E,Y)}$$

Subclass:

$$(a)\quad \frac{(A,\mathsf{sc},B),(B,\mathsf{sc},C)}{(A,\mathsf{sc},C)} \qquad (b)\quad \frac{(A,\mathsf{sc},B),(X,\mathsf{type},A)}{(X,\mathsf{type},B)}$$

Typing:

$$(a)\quad \frac{(D,\mathsf{dom},B),(X,D,Y)}{(X,\mathsf{type},B)} \qquad (b)\quad \frac{(D,\mathsf{range},B),(X,D,Y)}{(Y,\mathsf{type},B)}$$

Implicit Typing:

$$(a)\quad \frac{(A,\mathsf{dom},B),(D,\mathsf{sp},A),(X,D,Y)}{(X,\mathsf{type},B)}$$

$$(b)\quad \frac{(A,\mathsf{range},B),(D,\mathsf{sp},A),(X,D,Y)}{(Y,\mathsf{type},B)}$$

An instantiation of the rules presented in *Definition* 2.18 involves the systematic replacement of meta-variables with elements from $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$ such that the inferred information represents well formed RDF triples.

Example 2.8 (ρdf Deductive system)
In the rules that follow the variables used in Definition *2.18 are replaced with actual subjects, predicates and objects.*

Simple:

A simple model of the graph represented in Figure *2.2 is presented in* Example *2.5.*

continued overleaf ->

Subproperty:

`rdfs:subPropertyOf` *is an instance of* `rdf:Property`*, which is used to state that all resources related by one property are also related by another:*

(a)

$$\frac{(\texttt{entx:develops}, \mathsf{sp}, \texttt{entx:implements}), (\texttt{entx:implements}, \mathsf{sp}, \texttt{entx:worksOn})}{(\texttt{entx:develops}, \mathsf{sp}, \texttt{entx:worksOn})}$$

(b)

$$\frac{(\texttt{entx:implements}, \mathsf{sp}, \texttt{entx:worksOn}), (\texttt{entx:JoeBloggs}, \texttt{entx:implements}, \texttt{entx:CRMProject})}{(\texttt{entx:JoeBloggs}, \texttt{entx:worksOn}, \texttt{entx:CRMProject})}$$

Subclass:

`rdfs:subClassOf` *is an instance of* `rdf:Property`*, which is used to state that all instances of one class are instances of another class:*

(a)

$$\frac{(\texttt{entx:Manager}, \mathsf{sc}, \texttt{entx:Employee}), (\texttt{entx:Employee}, \mathsf{sc}, \texttt{foaf:Person})}{(\texttt{entx:Manager}, \mathsf{sc}, \texttt{foaf:Person})}$$

(b)

$$\frac{(\texttt{entx:Manager}, \mathsf{sc}, \texttt{entx:Employee}), (\texttt{entx:JoeBloggs}, \mathsf{type}, \texttt{entx:Manager})}{(\texttt{entx:JoeBloggs}, \mathsf{type}, \texttt{entx:Employee})}$$

Typing:

`rdf:type` *is an instance of* `rdf:Property`*, which is used to state that a resource is an instance of a class:*

(a)

$$\frac{(\texttt{entx:develops}, \mathsf{dom}, \texttt{entx:Person}), (\texttt{entx:JoeBloggs}, \texttt{entx:develops}, \texttt{entx:CRMProject})}{(\texttt{entx:JoeBloggs}, \mathsf{type}, \texttt{entx:Person})}$$

(b)

$$\frac{(\texttt{entx:develops}, \mathsf{range}, \texttt{entx:Project}), (\texttt{entx:JoeBloggs}, \texttt{entx:develops}, \texttt{entx:CRMProject})}{(\texttt{entx:CRMProject}, \mathsf{type}, \texttt{entx:Project})}$$

Implicit Typing:

`rdfs:domain` *is an instance of* `rdf:Property`*, which is used to state that any resources that has a given property is an instance of one or more classes. While* ***`rdfs:range`*** *is an instance of* `rdf:Property` *which is used to state that the values of a property are instances of one ore more classes:*

(a)

$$\frac{\begin{array}{r}(\texttt{entx:worksOn}, \mathsf{dom}, \texttt{entx:Person}), (\texttt{entx:develops}, \mathsf{sp}, \texttt{entx:worksOn}),\\ (\texttt{entx:JoeBloggs}, \texttt{entx:develops}, \texttt{entx:CRMProject})\end{array}}{(\texttt{entx:JoeBloggs}, \mathsf{type}, \texttt{entx:Person})}$$

(b)

$$\frac{\begin{array}{r}(\texttt{entx:worksOn}, \mathsf{range}, \texttt{entx:Project}), (\texttt{entx:develops}, \mathsf{sp}, \texttt{entx:worksOn}),\\ (\texttt{entx:JoeBloggs}, \texttt{entx:develops}, \texttt{entx:CRMProject})\end{array}}{(\texttt{entx:CRMProject}, \mathsf{type}, \texttt{entx:Project})}$$

2.3.2 Beyond RDFS Inference

Generally speaking, RDF is used to express binary predicate relations. Whereas RDFS is used to define the domain and the range of these properties, hierarchies of classes and hierarchies of properties. However, using RDFS, it is not possible to represent complex sentences that include cardinality constraints or to model complex relations between classes, such as disjointness or equivalence.

The DARPA Agent Markup Language (DAML) program was an initiative, which was setup to develop a more expressive schema language and a related tool set. One of the outputs of the program was the DAML ontology language (DAML-ONT), which extends RDFS with more expressive class definitions. In parallel, a team of researchers working on a project known as OntoKnowledge, developed the Ontology Inference Layer (OIL) language. Like DAML-ONT the language extends RDFS. However, OIL focuses primarily on instances of concepts and a number of predicates (known as roles). Later both teams joined forces and released a language, known as DAML+OIL. The combined language was the starting point for the Web Ontology Language (OWL) (McGuinness and van Harmelen, 2004), which was released by the W3C's as a recommendation in 2004. An overview of DAML, OIL and DAML-OIL is presented in Gomez-Perez and Corcho (2002).

OWL is related to a family of logics known as Description Logics (DL). Like RDFS, OWL uses classes and properties (commonly called roles) and instances (known as individuals). However, OWL provides for:

- A rich set of relations between classes, roles and individuals (for example, `owl:sameAs`, `owl:equivalentClass`, `owl:differentFrom`).
- A number of logical operations (for example, `owl:unionOf`, `owl:intersectionOf` and `owl:complementOf`.
- Several cardinality constraints (for example, `owl:someValuesFrom`, `owl:allValuesFrom`, `owl:cardinality`, `owl:minCardinality` and `owl:maxCardinality`).

When it comes to reasoning, in an attempt to balance expressivity and efficiency, the specification presents three sub-languages (OWL Full, OWL DL and OWL Lite), otherwise known as *species* of OWL.

- OWL Full, which includes both OWL DL and OWL Lite, is the most expressive. It is the only OWL language, which supports RDFS. However, it is in general undecidable and it is not supported by the majority of software vendors.
- OWL DL, which includes OWL Lite, is less expressive than OWL Full. A number of restrictions are imposed: it is not permissible to use `rdfs:Class` or `rdfs:Property`; there must be a clear separation between individuals, classes, roles and datatypes; and limitations are applied to several roles. As a result OWL DL is decidable and has a worst case computational complexity of NExpTime. In addition, it is fully supported by most software tools.
- OWL Lite, which has a worst case computational complexity of ExpTime, is the least expressive. Several restrictions, in addition to those specified for OWL DL, are imposed: it is not possible to use `rdfs:Class` or `rdfs:Property`; cardinality constraints are limited to 0 and 1; in certain situations unnamed classes cannot be used; and a number of additional limitations are applied to roles.

The second version of the language OWL2 was released as a W3C recommendation in 2012 (W3C OWL Working Group, 2012). OWL2 comes in two flavours (OWL2 Full and OWL2 DL). OWL2 DL is composed of three profiles (OWL2EL, OWL2QL and OWL2RL) that are based on well used DL constructs. OWL2 extends OWL with additional datatypes, additional annotations and relaxes the strict separation between classes, properties and annotations. For specific details on OWL, the reader is referred to the OWL (McGuinness and van Harmelen, 2004) and OWL2 (W3C OWL Working Group, 2012) specification documentation.

2.4 Linked Data

The term *Linked Data Web (LDW)* is used to describe a World Wold Wide where structured data is directly linked with other relevant data using machine accessible formats. According to Heath and Bizer (2011), the LDW enables things, otherwise known as resources, to be linked using URIs, in a similar way to how web documents are interlinked using the HyperText Markup Language (HTML) hypertext reference (HREF) attribute. In recent years, we have seen significant advances in the technology used to both publish and consume Linked Data, namely RDF, RDFS, OWL and SPARQL.

2.4.1 Linked Data Principles

Underpinning the LDW is a set of best practices for publishing and interlinking structured data, know as the *Principles of Linked Data*. These principles are defined by Berners-Lee, Tim (2006) as follows:

" 1. Use URIs as names of 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 term *thing* is used to refer to both real world entities and abstract concepts, commonly referred to as resources. The LDW builds on the existing web infrastructure, by using HyperText Transfer Protocol (HTTP) URIs to identify things, as well as documents. However, URIs only support a subset of the American Standard Code for Information Interchange (ASCII) character set. Later the W3C introduced Internationalised Resource Identifiers (IRIs), that provides support for the richer Unicode character set. Although the principles defined by (Berners-Lee, Tim, 2006) refer to URIs, as there is a mapping from IRIs to URIs, it is also possible to use IRIs. The latest versions of the RDF, RDFS and SPARQL specifications have all chosen to migrate from URIs to IRIs.

However, it is not enough to simply use URIs to refer to things. According to the Linked Data principles, it should be feasible to use the URI to return a description of the resource (commonly referred to as dereferencing). As URIs often represent real world entities, it is common practice to use different URIs to represent the resource and the document that describes it. Two different strategies, that can be used to dereference URIs exist, namely 303 redirects and Hash URI's. In the case of 303 redirects, when a client attempts to dereference a resource, the server responds with a `303 See Other`, and a URI for the document that describes this resource. The client subsequently uses this new URI to retrieve the description of the resource. Whereas, in the case of hash URIs a # separator is used to append an *identifier*, which identifies the resource, to the end of the URI. Prior to attempting to dereference the resource, the client strips off the # and the identifier, making it possible to distinguish between the physical resource and the document that describes it.

The final principle refers to the linking of URI's. Just like the web of documents uses reference links to enable humans and machines to navigate web pages, the web of data is constructed in a similar fashion. By using RDF to describe resources, it is possible not only to link structured data, but also to describe complex relations between resources in a machine readable format.

2.4.2 Publishing Linked Data

When it comes to publishing Linked Data, a number of alternative strategies exists.

- Static data, or data that does not change frequently, can be represented in RDF documents that are served by regular web servers.
- RDF data can be embedded in HTML documents, using RDFa (Adida et al., 2013), a W3C Recommendation which enables RDF markup to be embedded in HTML attributes.
- RDF stores generally allow RDF data to be exposed as Linked Data via interfaces know as SPARQL endpoints. A SPARQL endpoint accepts queries and sends responses using the HTTP protocol.
- Information stored in relational databases can be translated into RDF using relational data to RDF converters. The W3C recommends two complementary RDB2RDF standards, *Direct Mapping of Relational Data to RDF* (Arenas et al., 2012) and *RDB to RDF Mapping Language (R2RML)* (Das et al., 2012). *Direct Mapping* defines a simple transformation which can also be used to materialize RDF graphs or define virtual graphs, which can be queried by SPARQL or traversed by an RDF graph API. Whereas *R2RML* is used to express custom mappings from relational databases to RDF datasets. Unlike *Direct Mapping*, where the RDF graph directly reflects the structure of the database, R2RML caters for the definition of highly customized views over relational data.

2.4.3 Consuming Linked Data

In a similar fashion to the web of documents, Linked Data can be accessed using Linked Data browsers, search engines and crawlers. When it comes to consuming Linked Data, Heath and Bizer (2011) propose three different patterns.

- The *Crawling Pattern*, which retrieves RDF data by starting with a seed URI and uses the dereferenced data to find more links. All data is stored locally and an integrated view is provided to the application.
- The *On-The-Fly-Dereferencing Pattern* also uses dereferencing, however in this instance data is consumed at query time.
- The *Query Federation Pattern* involves querying a fixed set of data sources via SPARQL endpoints.

The choice of pattern depends on the number of data sources to be accessed, the response time demanded by the application, and importance placed on up to date data.

2.5 Summary

In this chapter, we provided an overview of the RDF data model and presented a simple interpretation of a ground graph according the RDF model theoretic semantics. Following on from this, we introduced the SPARQL query and update languages. We described the four SPARQL query forms and we examined a number of different query constructs, such as subqueries, property paths, negation and aggregates. In addition, we discussed how SPARQL update can be used to manage graph data. We subsequently discussed the role played by RDFS when it comes to data modelling and reasoning, and presented a deductive system for a subset of RDFS, known as ρdf. We briefly introduced a more expressive language, known as OWL. Finally, we discussed the principles of Linked Data and presented existing strategies for publishing and consuming Linked Data.

Chapter 3

State of the Art

Generally speaking the term *access control* is used to refer to the *model*, which is the blueprint used to guide the access control process; the *policy language*, which defines both the syntax and the semantics of the access control rules; and the *framework*, which is a combination of the access control model, the language and the enforcement mechanism. At its most basic, an access control rule (otherwise known as an authorisation) can be represented as a tuple $\langle S, R, AR \rangle$ where S denotes the *subject* (entities requesting access to resources), R denotes the *resource* (entities to be protected) and AR represents the *access rights* (permissions and prohibitions often based on actions pertaining to the resource). Sets of access control rules are collectively referred to as an *access control policy.* The decision to grant or deny access is based on two distinct processes, *authentication* and *authorisation. Authentication* involves the verification of identity (you are who you say you are). Whereas, *authorisation* is the process of granting or denying access to system resources based on identity.

Security in general and access control in particularity have been extensively studied by both the database and the information system communities, among others. Although various access control models, policy languages and enforcement frameworks have been proposed, given that access control models tend not to be domain specific, it is not surprising that they have had a strong influence on access control research within the Semantic Web community. Early work on access control policy specification and enforcement, within the Semantic Web community, focused on representing existing access control models and standards using semantic technology; proposing new access control models suitable for open, heterogeneous and distributed environments; and recommending languages and frameworks that can be used to facilitate access control specification and maintenance. Later the focus moved to access control for the RDF data model and access control propagation, based on the semantic relations between policy entities. In recent years the focus has shifted to access control policy specification and enforcement over Linked Data.

The remainder of the chapter is structured as follows: *Section* 3.1 presents relevant access control models and standardisation efforts and discusses how they have been applied to, or enhanced, using semantic technology. *Section* 3.2 describes several well known policy languages and frameworks. *Section* 3.3 describes the different access control strategies that have been proposed for RDF data. Finally *Section* 3.4 presents a set of access control requirements and examines the suitability of the presented access control models, languages and frameworks for protecting *Linked Data.*

3.1 Access Control Models and Standards

In this section, we describe a number of well known access control models, emerging access control models and relevant standardisation efforts. In each instance, we provide a description of the access control model/standard and discuss how it has been applied to, or enhanced using, semantic technologies.

3.1.1 Access Control Models

Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role Based Access Control (RBAC) are the most prominent access control models found in the literature, and used in practice. View Based Access Control (VBAC) is a complementary access control model which grants access to sets of entities, logically structured as views. In addition, several researchers have proposed new access control models, that were deemed more suitable for the open, heterogeneous and distributed architecture of the web. Primary research efforts, involve using properties (relating to the subject, resource or the environment) as opposed to identities, in order to determine if access to resources should be permitted. Attribute Based Access Control (McCollum et al., 1990) and Context Based Access Control (Rubart, 2005) are the predominant works in this area.

3.1.1.1 Mandatory Access Control

MAC limits access to resources using access control policies determined by a central authority (Samarati and de Vimercati, 2001). The central authority is responsible for classifying both *subjects* and *resources* according to *security levels*. Resources are assigned labels that represent the security level required to access the resource, and only subjects with the same security level or higher are granted access.

MAC was originally developed for military applications and therefore it is best suited to closed environments, where a great deal of control is required (Bertino and Sandhu, 2005). Given the open, heterogeneous and distributed nature of the web, it is not surprising that MAC has not gained much traction among Semantic Web researchers.

The primary focus has been on demonstrating how MAC can be supported by Semantic Web policy languages. Kodali et al. (2004) describe a unifying framework which can be used to represent a number of different access control models, MAC being one of them. Whereas, Yagüe et al. (2003) demonstrate how their attribute based access control model, can be used to represent MAC, DAC and RBAC. Details of both of these models are provided in *Section* 3.1.1.7.

3.1.1.2 Discretionary Access Control

DAC policies associate one or more *subjects* with sets of *access rights* pertaining to one or more *resources*. Like MAC, DAC restricts access by means of a central access control policy, however the users are allowed to override the central policy and can pass their access rights on to other users, known formally as delegation (Sandhu and Samarati, 1994).

As with MAC, both Kodali et al. (2004) and Yagüe et al. (2003) demonstrate how the access control models they propose can support Linked Data, DAC being one. Details of both of these models are provided in *Section* 3.1.1.7. According to Weitzner et al. (2006a), the web needs discretionary, rule based access control. Although the authors describe a motivating scenario and present a potential solution, they focus primarily on the general architecture of the system, as opposed to investigating how discretionary access control can be modelled or enforced.

3.1.1.3 Role Based Access Control

RBAC restricts access to *resources* to groups of *users*, with common responsibilities or tasks, generally referred to as *roles*. In RBAC, users are assigned to appropriate roles and access to resources is granted to one or more roles, as opposed to users directly (Sandhu et al., 1994). The term *session* is commonly used to refer to the set of roles the user is currently assuming (their active roles). Whereas, *role deactivation* is generally used to refer to the process whereby a user is removed from a role. Depending on the use case, roles may be organised to form either a hierarchy or a partial order. Such structures are used to simplify access control specification and maintenance. Access control constraints, are commonly used to enforce conditions over access control policies. For RBAC, these constraints take the form of both static and dynamic *separation of duty* (a user cannot be assigned to two roles simultaneously) and prerequisites (a user can only be assigned to a role if they have already been assigned another required role).

When it comes to traditional access control models, much of the research effort within the Semantic Web community, focuses on modelling RBAC using ontologies. In order to provide the reader with an overview of the different possibilities, we present four different strategies for representing RBAC in OWL.

RBAC entities as classes. Di et al. (2005) provide a basic modelling for RBAC concepts and constraints using OWL. `User`, `Role`, `Permission` and `Session` entities are represented as classes. While, the following properties are used to represent relationships:

- `hasRole` (assigns a user to a role);
- `hasPermission` (associates a role with a permission);
- `belongTo` (maps a session to a single user); and
- `hasActiveRole` (maps a set of roles to a session).

Two additional properties are used to model separation of duty and prerequisite constraints:

- `conflictRole` (indicates that there is a conflict between two roles); and
- `prerequesiteRole` (specifies that one role is dependent on another).

Roles as classes versus roles as instances. Finin et al. (2008a) discuss the advantages and disadvantages of two different approaches for representing RBAC policies in OWL. The first approach which represents *roles as classes* and the second approach which represents *roles as instances*. In both instances `Subject`, `Object` and `Action` entities are represented as classes. The authors propose two disjoint `Action` subclasses, `PermittedAction` and `ProhibitedAction`, which are used to associate permissions and prohibitions with users.

- When ***roles are represented as classes*** a general `Role` is defined as a class and specific *roles* are defined as subclasses of the `Role` class. The `rdf:type` property is used to associates users with roles and an `activeForm` property is used to indicate if a role is currently active. The role hierarchy is modelled using the `rdfs:subclassOf` property. While, the `owl:disjointWith` property is used to model separation of duty constraints.
- When ***roles are represented as instances*** a general `Role` is defined as a class and *roles* are defined as instances of the `Role` class. Role hierarchies are represented using the `owl:TransitiveProperty`, which is used to describe a transitive relationship between properties. Two properties `role` and `activeRole` are used to assign roles to users. Whereas, two additional properties `ssod` and `dsod` are used to represent static and dynamic separation of duty.

When *roles are represented as instances* the modelling is simple and more concise. Whereas, when *roles are represented as classes*, it is possible to determine subsumption relationships according to a users active role and the role hierarchy, using standard description logic subsumption. As OWL is monotonic, using either approach, it is not possible to remove assertions from the knowledge base, as such role deactivation cannot be supported. Although the authors indicate that model checking can be used to verify the effectiveness of the proposed approaches, no concrete details are supplied.

Using the eXtensible Access Control Markup Language for RBAC. Ferrini and Bertino (2009), demonstrate how the eXtensible Access Control Markup Language (XACML), an attribute based access control language and framework proposed by the Advanced Open Standards for the Information Society (OASIS), and OWL can be used to specify and enforce RBAC. Policies are specified using the XACML vocabulary, whereas role hierarchies and constraints are represented using OWL. The proposed modelling is a combination of the two approaches, proposed by Finin et al. (2008a).

- Like the *roles as instances approach*, *roles* are represented as instances of a generic `Role` class and *subjects* as instances of a generic `Subject` class. Two properties `subRoleOf` and `supRoleOf`, are used to represent the relationship between role instances. The `owl:inverseOf` property is used to indicate the relationship between the `subRoleOf` and `supRoleOf` and the `owl:TransitiveProperty` is used to model the transitive relationship between roles. A `hasRole` property is used to indicate that a subject has been assigned to a give role.
- Like the *roles as classes approach* (Finin et al., 2008a), the `owl:disjointWith` property and an `activeRole` property are used to model separation of duty and active roles respectively.

A request is composed of one or more attributes pertaining to the subject, action, resource or the environment. In order to support RBAC the user submits their role as a subject attribute. On receipt of the request the system extracts the role from the subject attribute. The description logic reasoner is used to retrieve additional roles that can be inferred from the OWL ontology. These roles are subsequently fed into the XACML engine. Finally the XACML engine consults the XACML policy in order to determine if access should be granted. As the policies are not modelled in OWL it is possible to support role deactivation. However, with the existing modelling it is not possible to exploit reasoning over policy resources or access rights.

Using the Common Information Model for RBAC. Alcaraz Calero et al. (2010) demonstrate how the Common Information Model (CIM) (a standard vocabulary used to represent information technology objects and the relationship between them) can be used to represent RBAC. The authors provide a mapping between RBAC entities and CIM entities that are represented using OWL. The primary entities are mapped as follows:

- subject is mapped to the CIM `Identity`;
- privilege is simply a CIM `Privilege`; and
- object is associated with an entity inheriting from CIM `ManagedElement`.

Information is asserted into the knowledge base using two different properties:

- `AuthorisationSubject` which maps a subject to a privilege; and
- `AuthorisationTarget` which associates a privilege with an object.

The following properties, which are declared transitive (using `owl:TransitiveProperty`), are used to represent hierarchies:

- `MemberOfGroup` is used to represent role and group hierarchies; and
- `Aggregations` and `Dependencies` are used to model object hierarchies.

Constraints are represented as rules using the Semantic Web Rule Language (SWRL). For example, in order to model a separation of duty constraint, a rule is defined which checks if there are any common instances between roles and if so generates an exception. The authors describe how a SPARQL enabled reasoner, such as Pellet, can be used to query the OWL policy, in order to determine if access should be granted or denied.

Several authors (Kagal et al., 2003a; Yagüe et al., 2003; Corradi et al., 2004b; Kodali et al., 2004; Toninelli et al., 2006) argue that RBAC is not suitable for systems where it is not possible to assign permissions in advance or where permissions change frequently. Nonetheless, the access control models they propose, either extend RBAC with additional functionality or include RBAC.

3.1.1.4 View Based Access Control

VBAC (Griffiths and Wade, 1976) is used in relational databases to simultaneously grant access to one or more relations, tuples, attributes or values. A similar approach is used in Object Oriented Access Control (OBAC) (Evered and Bögeholz, 2004), where access rights are granted to sets of application objects.

Using rules to create views. Li and Cheung (2008) use rules to generate views of data based on relations between ontology concepts. For additional details see *Section* 3.3.1.3.

Using SPARQL to create views. Gabillon and Letouzey (2010) describe how RDF data can be logically organised into views using SPARQL `CONSTRUCT` and `DESCRIBE` queries. Inspired by access control in relational databases, the authors propose an access control model which can be used to grant/deny access to named graphs or SPARQL views. For additional details see *Section* 3.3.3.2.

Using policies to limit access to views. Costabello et al. (2012a) propose a policy language that can be used to restrict access to named graphs using query rewriting. For additional details see *Section* 3.3.3.2.

3.1.1.5 Attribute Based Access Control

ABAC was designed for distributed systems, where the subject may not be known to the system, prior to the submission of a request. ABAC grants or denies access to *resources*, based on properties of the *subject* and/or the *resource*, known as *attributes*.

Using rules for ABAC. Stermsek et al. (2004) discuss how attributes can be used to specify access control directly using rules and indirectly using roles. In the former, the requesters attributes are compared against a policy, which indicates the attributes necessary to access a resource. Whereas in the latter, the requesters attributes are used to determine the access rights or roles the subject should be mapped to. The authors, describe three different mechanisms that can be used to obtain the attributes pertaining to a subject:

(i) subjects can send all of their attributes to the server, with the initial request;

(ii) the server can request particular attributes from the subject, once the initial request is submitted; and

(iii) given the subject may be cautious about giving out the requested credentials to an unknown entity, both the server and the client could exchange policies and attributes (a process commonly known as trust negotiation).

Although a high level architecture is described, the formal representation of the model is left to future work.

A Metadata-Based Access Control design pattern. Priebe et al. (2004), inspired by software design patterns, present a Metadata-Based Access Control (MBAC) pattern, which aims to encapsulate best practice with respect to attribute based access control. In the presented pattern, the authors indicate that, *subjects* and *objects* should be modeled as sets of attribute/value pairs. While, authorisation *subjects* and authorisation *resources* should be described in terms of required attributes and corresponding values. Follow up work, by Priebe et al. (2006), describes how attribute based access control policies specified using XACML can be modelled using OWL. The XACML policies together with a reasoning engine, allows for deductive reasoning based on the OWL vocabulary.

Combining roles and attributes. Cirio et al. (2007) propose an access control model which combines role and the attribute based access control. Rather than associating permissions with attributes directly, the permissions are associated with roles and attributes are added to roles. In the presented modelling `roles`, `resources`, `privileges` and `actions` are represented as classes, similar to the roles as classes approach presented by Finin et al. (2008a). The authors define the following predicates:

- `hasPrivelege` (associates a role with a privilege);
- `notTogetherWith` (used to express dynamic separation of duty);
- `performsAction` (maps privileges to actions); and
- `usesResource` (maps privileges to resources).

The authors used the `rdfs:domain` and `rdfs:range` properties to assert knowledge instead of using them as constraints. Such an approach simplifies policy specification as concepts can be defined implicitly. Using description logic, it is not possible to specify policies based on relations between instances. Therefore, the authors use two additional predicates, `requiresTrue` and `requiresFalse`, to specify run time constraints based on user attributes. Constraints are represented using SPARQL `ASK` queries and are evaluated at runtime using a custom policy decision point, which wraps a description logic reasoner.

3.1.1.6 Context Based Access Control

CBAC uses properties, pertaining to users, resources and the environment, to grant/deny access to resources. In light of new and emerging human computer interaction paradigms, such as ubiquitous computing and the internet of things, access control based on context has been graining traction in recent years.

Physical versus Logical Context. Corradi et al. (2004a) and Montanari et al. (2005) propose a context based access control model and framework, called UbiCOSM. The proposed access control model uses context to group policies. The authors distinguish between physical and logical context. The former relates to the physical location denoted by geographical coordinates. Whereas, the latter refers to logical properties pertaining to the users and resources. In the proposed modelling, user identities and roles are specified using logical properties. *Context* is represented using a generic `context` class and the following properties are used to associate instances with classes:

- `context_Name` (identifier for the context);
- `context_Type` (either physical or logical); and
- `context_Activation_Condition` (used to associate specify contextual constraints).

Resources are represented using a generic `resource` class and instances are specified using the following properties:

- `resource_Name` (identifier for the context); and

- `resource_Description` (used to associate context with resources).

While, *permissions* are represented using a generic `permission` class and instances are specified using the following properties:

- `Name` (identifier for the permission);
- `Target` (the resource the permission is applied to);
- `Action` (the operation pertaining to the resource); and
- `Kind` (which is used to specify either a positive or a negative).

Access control policies are composed of mappings between *context* and *permissions*. Complex policies are generated using *conjunction*, *disjunction* and *negation* operators. When a user requests access to a resource, the UbiCOSM enforcement framework retrieves the relevant policies and generates a view based on the users permissions. If the user possesses the context necessary to access the resource, access is granted. Otherwise access is denied.

Context relating to the subject, object, transaction and environment. Shen and Cheng (2011) propose a semantic-aware context-based access control model (SCBAC) and demonstrate how together ontologies and rules can be used to generate authorisations. The authors provide both a context ontology and a policy ontology, which can be used to specify positive and negative authorisations and obligations. Like Corradi et al. (2004a) and Montanari et al. (2005), context provides a level of indirection between *subjects* and *permissions*. The authors propose four different types of context, that are relevant from a access control perspective:

- *Subject contexts* (properties pertaining to a subject);
- *Object contexts* (information relating to resources);
- *Transaction contexts* (either current or past information relating to a particular action);
- *Environment contexts* (other contextual information not relating directly to the subject, the resource or the action, for example time of day).

An authorisation permits/prohibits an action, based on sets of contexts supplied by the user. `Actions` are used to represent operations that a subject wishes to perform. `Permission Assignments` are used to associate contexts with actions. Rules are used to insert new authorisations, based on contextual information, into a knowledge base. An access request is represented as a tuple $\langle U, P, C, R\rangle$ where user U, requests privilege P on resource R, in light of a given context C. Access is enforced by representing access requests as SPARQL queries that are executed over the knowledge base. However, given OWL is monotonic, it is not clear how changes to contextual information are handled in the proposed approach.

3.1.1.7 Combination of Traditional Models

A couple of authors have proposed access control strategies which can be used to represent a number of different access control models.

A unifying access control framework. Kodali et al. (2004) describe a unifying framework which can be used to represent and enforce access control policies adhering to the MAC, DAC and the RBAC access control paradigms. The authors demonstrate how a tuple $\langle S, P, O\rangle$, where S, P and O represent *subject*, *permissions* and *objects* respectively, can be used to model authorisations, pertaining to each of the access control models. To this end they propose a basic ontology with:

- an `AC Models` class and `Disc Access Control`, `Mandatory Access Control` and `Role-Based Access Control` subclasses;

- a `Subject` class and `DAC-Subject`, `MAC-Subject` and `RBAC-Subject` subclasses; and
- an `Object` class and `DAC-Object`, `MAC-Object` and `RBAC-Object` subclasses.

In addition, a general `Constraint` class and `Time-dependent Constraints`, `Obligatory Constraints` and `Context-dependent Constraints` subclasses are defined.

Supporting RBAC, DAC and MAC using attributes. Yagüe et al. (2003) attest that access control mechanisms that use attributes, to represent user credentials, are more general than MAC, DAC and RBAC, and as a result can be used to represent such policies. The authors propose an access control model which determines access based on the semantics of the resource. Access control policies are specified in terms of attributes and access is permitted or denied depending on requester attributes, that are submitted using certificates that have been verified by a trusted certification entity. No specific details are provided on how their access control model can be used to support MAC, DAC and RBAC.

3.1.2 Relevant Standardisation Efforts

In recent years, there have been a number of standardisation efforts, by both the W3C and OASIS, in relation to access control for web data. In this section, we provide a high level overview of the relevant standards and describe how they have been adapted or enhanced using semantic technology.

3.1.2.1 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML)[1], is an OASIS standard, which is used to represent attribute based access control policies (Rissanen, 2013). XML was chosen as the representation formalism for the policy language as it:

(i) can be used to represent information in both a human and machine readable manner;

(ii) can easily be adapted to represent different access control requirements; and

(iii) is widely supported by software vendors.

The specification provides an XML schema which can be used to represent attribute based access control policies. The root of an XACML policy is a `Policy` or a `PolicySet` (used to represent multiple policies). Policies are composed of sets of `Rules` that are in turn composed of sets of a `Targets` (conditions relating to `Subjects`, `Resources` and `Actions`) and an `Access Decision` (permit, deny or not applicable). An XACML `Request` is represented as a tuple $\langle subject, resource, action, environment\rangle$. Where *subject* represents the entity requesting access, *resource* denotes the object to be protected, *action* defines the type of access and *environment* represents the requesters attributes. The XACML framework is composed of the following components:

- a *policy decision point* (which evaluates policies and returns a response);
- a *policy enforcement point* (which is responsible for making decision requests and enforcing the decisions);
- a *policy information point* (which obtains attributes pertaining to subjects, resources and the environment); and
- a *policy administration point* (which enables policies and sets of policies to be created and updated);

[1]XACML,`https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

Deductive reasoning over XACML policies. Priebe et al. (2006, 2007) present an extension to XACML, which enables deductive reasoning over attribute ontologies, specified using OWL. In order to support reasoning, the authors propose two additional architecture components: an *ontology administration point* and an *inference engine*. If the attributes presented by the requester are not explicitly stated in the access control policy, the system attempts to infer the required access rights from the policy attributes, requester attributes and the attributes ontology, using the *inference engine*.

Chen and Stuckenschmidt (2010) also extend XACML with OWL deductive reasoning capabilities. Access control policies are specified using XACML and access control is enforced via query rewriting. The authors use SPARQL filters to both permit and deny access to instance data. Reasoning over the data, represented in the filters, is delegated to reasoners which support OWL and SPARQL.

Extending XACML policies with Context. Franzoni et al. (2007) demonstrate how XACML can be extended to consider access control based on contextual properties pertaining to either the user or the application. In addition to standard access control policies, specified using XACML, the authors propose fine grained access control policies, which are used to specify the *instances* of a *concept* that a user is permitted to access. The proposed fine grained access control policies are enforced over RDF data, by rewriting a SeRQL query (an alternative to SPARQL), so that it limits access to the instances that have been permitted.

Supporting RBAC using XACML. Ferrini and Bertino (2009) describe an extension of XACML, which uses a combination of XACML and OWL, in order to support RBAC. Like Priebe et al. (2006, 2007), access control policies are specified using XACML, therefore it is possible to take advantage of OWL's out of the box reasoning capabilities. However rather than modelling relationships between attributes, the authors model role hierarchies and cardinality constraints.

3.1.2.2 Web Identity and Discovery

Web Identity and Discovery (WebID)[2], which is supported by a W3C community group, is a mechanism used to uniquely identify and authenticate a person, company, organisation or other entity, by means of a URI (Sambra et al., 2014). Essentially a WebID is a HTTP URI which is used to represent an agent. According to the WebID Incubator Group (Sporny et al., 2011), a WebID should:

(i) be under the control of the entity it describes;

(ii) be linkable on the web;

(iii) describe the entity is represents;

(iv) enable authentication and access control;

(v) respect the privacy of the entity it describes; and

(vi) rely solely on HTTP and the Linked Data web.

A description of the agent is provided in an RDF document, known as a WebID profile, which can be dereferenced using 303 or Hash URI's (*see Section* 2.4.1). The WebID-TLS protocol (where TLS stands for Transport Layer Security) specifies how together the WebID profile and public key certificates, can be used to authenticate users (Inkster et al., 2014). The user places their WebID profile document URI in the *Subject Alternative Names* field of their certificate. Once

[2]WebID, `http://www.w3.org/wiki/WebID`

the certificate has been generated the user adds the public key details to their WebID profile document. A service wishing to authenticate the user, needs to verify that the public key of the certificate it receives matches the public key specified in the WebID profile.

Hollenbach et al. (2009) use FOAF and the Secure Sockets Layer (SSL) to determine if the public key in the users FOAF profile matches that of the certificate. When a requester attempts to authenticate, the system extracts the public key from the certificate. The system subsequently verifies the signature and if successful, a SPARQL query is run against the FOAF profile, in order to determine if it contains a matching public key. In Berners-Lee et al. (2009), the authors describe their vision of a read-write web of data and present a proof of concept. Like Hollenbach et al. (2009), the authors discuss how WebID, together with FOAF+SSL, can be used for authentication. Although Stermsek et al. (2004) do not use the term WebID, they describe how attributes pertaining to a subject (commonly known as credentials), can be associated with public keys and attached to digital certificates.

3.1.2.3 Web Access Control

WebAccessControl (WAC)[3] is an RDF vocabulary and an access control framework, which demonstrates how together WebID and access control policies specified using the WAC vocabulary, can be used to enforce distributed access control. WAC authorisations grant *agents*, *access* to *resources*. Agents are specified using the `agent` and `agentClass` properties and resources are specified using the `accessTo` and `accessToClass` properties. Whereas, `Read`, `Write`, `Append` and `Control` access rights are represented as classes. Once the user has been authenticated using WebID, the system checks if a policy exists which grants the user access to the requested resource. If no such policy exists, then the system checks for classes that are granted access. For each class the system dereferences the URI and checks if the users WedID is a type of the given class. If yes, then the user is granted access to the system.

Using WebID and WAC. In addition to using WebID for authentication, Hollenbach et al. (2009) use the WAC vocabulary to specify access control policies. The authors provide a mapping between permissions and HTTP operations and demonstrate how together WebID and WAC can be used to grant/deny access to web resources. In order to verify if the user has the permissions required to perform the requested HTTP operation, a SPARQL query is executed against the access control policy. If access is denied, a 403 response is returned from the server.

Extending the WAC vocabulary. Both Villata et al. (2011) and Sacco and Passant (2011b) extend the WAC to cater for access control over the RDF data model. Using the extended vocabularies, it is possible to associate access control with individual RDF resources (subjects, predicates and objects) and also collections of RDF resources (named graph). In addition, the authors extend the vocabulary to cater for a broader set of access privileges (`create`, `read`, `write`, `update`, `delete`, `append` and `control`).

3.1.2.4 Platform for Privacy Preferences

The *Platform for Privacy Preferences (P3P)*[4], is a W3C recommendation, which enables websites to express their privacy preferences in a machine readable format. Like *XACML*, the specification provides an XML Schema, which can be used to specify policies. In addition, the specification details how privacy policies can be associated with webpages/websites and describes how P3P

[3]WAC,`http://www.w3.org/wiki/WebAccessControl`
[4]P3P,`http://www.w3.org/TR/P3P/`

policies can be used in conjunction with HTTP. Organisations wishing to specify machine readable privacy policies can publish their privacy policies using the P3P syntax. A reference to the policy can be added to a well known location (for example, `/w3c/p3p.xml`), which can be specified using the HTML `link` tag, or alternatively can form part of the HTTP Response. P3P agents can be built into browsers, plug-ins or proxy servers. The agent is responsible for fetching the servers privacy preferences and taking some action. This action can vary from simply displaying a symbol, to comparing the servers privacy preferences to those of the client and taking some form of action. A related specification, which describes *A P3P Preference Exchange Language (APPEL)* (Cranor et al., 2002), presents a vocabulary, which is used by individuals (as opposed to websites) to express their privacy preferences.

Extending the P3P vocabulary. Kolari et al. (2005) propose an extension to P3P, to cater for more expressive privacy preferences. The authors adopt a policy language, based on Semantic Web technology, known as Rei. As Rei is a general policy language, it can easily be used to represent existing P3P policies in a manner which supports reasoning based on context. As access rights in Rei are based on deontic logic, it is possible to model, not only positive and negative permissions, but also positive and negative obligations.

Representing P3P policies using OWL. Garcia and Toledo (2008) demonstrate how P3P policies can be represented in OWL. The authors detail how the Web Services Policy Framework (WS-Policy) can be used to allow service providers to expose these OWL based privacy policies.

3.2 Policy Languages and Frameworks

Policy languages can be categorised as either general or specific. In the former, the syntax caters for a diverse range of functional requirements (access control, query answering, service discovery, negotiation, to name but a few), whereas the latter focuses on just one functional requirement. Two of the most well-known access control languages, KAoS (Bradshaw et al., 1997, 2003; Uszok et al., 2003b) and Rei (Kagal and Finin, 2003; Kagal et al., 2003b), are in fact general policy languages. Natural language, programming languages, XML and ontologies can all be used to express policies. XML and ontologies are two popular choices for representing policy languages as they benefit from flexibility, extensibility and runtime adaptability. However, ontologies are better suited to modelling the semantic relationships between entities. In addition, the common framework and vocabulary used by ontologies, to represent data structures and schemas, provides greater interpretability and interoperability. Regardless of the language chosen, a logic based underlying formalisation is crucial for automatic reasoning over access control policies. Therefore, the work presented in this section is limited to policy languages that use ontologies, rules or a combination of both to represent policies. As the objective is to provide the reader with an overview of each approach, we present a detailed description of well known frameworks in each category. For a broader comparison of policy languages, the author is referred to a survey by Bonatti and Olmedilla (2007), which evaluates twelve different policy languages against a set of criteria, that are deemed necessary for ensuring security and privacy in a Semantic Web context.

3.2.1 Ontology Based Approaches

Using OWL ontologies it is possible to specify access control vocabularies that can easily be adopted by others. Additionally, access control policies specified using different vocabularies can easily be merged. By using an ontology based approach, it is possible to perform deductive reasoning (deriving the consequent) and abductive reasoning (affirming the consequent) over access control policies, with standard description logic reasoners. The former is often used to infer new policies based on relationship between access control entities, whereas the latter is used in access control administration in order to determine the access rights required to meet a given policy. In this section, we examine KAoS (Bradshaw et al., 1997, 2003; Uszok et al., 2003b) a general policy language which adopts a pure ontological approach.

3.2.1.1 KAoS

KAoS (Bradshaw et al., 1997, 2003; Uszok et al., 2003b) is an open distributed architecture, which allows for the specification, management and enforcement of a variety of policies. In initial versions of the language, policies were represented using DAML. However, the authors later moved to OWL, the successor of DAML (Uszok et al., 2004c). As both DAML and OWL are based on description logic, using the KAoS language it is possible to define class and property hierarchies, along with inference rules. Although KAoS was originally designed to enable interoperability between complex web agents (software that acts on behalf of humans) (Bradshaw et al., 1997; Suri et al., 2003), it was later applied to web services (Uszok et al., 2004a,d,c,b) and grid computing (Johnson et al., 2003; Uszok et al., 2004a).

Core entities. The authors define a set of core vocabularies, known as KAoS policy ontologies, that are used to describe:

- `actors` (both humans and artificial agents);
- `actions` (various system operations such as accessing, communication and monitoring);
- `resources` (entities associated with actions);
- `policy types` (authorisations and obligations); and
- `policies` (positive and negative constraints).

Specification of policies. A policy is used to express either an *authorisation* or an *obligation*, on the part of one or more *actors*, with respect to *actions* relating to *resources*. Policies are represented as instances of the aforementioned policy types. The language is not meant to be exhaustive, but rather to provide a basis, which can be further extended, to cater for use case specific classes, instances and rules. In order to simplify policy administration and enforcement, actors and resources are organised into domains, that can be nested indefinitely. Domains and subdomains are used to represent complex relationships between classes and instances, such as organisation structures.

Enforcement of policies. The authors propose a general policy and domain services framework, which consists of the following components: a *policy administration tool*, *directory services*, *guards*, *enforcers* and a *domain manger*. The *policy administration tool*, known as KPAT, is a user friendly interface that allows administrators, who are unfamiliar with DAML and OWL, to either specify new or maintain existing policies. *Guards* are responsible for enforcing platform independent policies. While, *enforcers* are responsible for enforcing policies that are platform dependent. The *domain manger* is used to manage domain membership and to distribute policies to guards. This component is also responsible for notifying the *guards* of any policy updates. Given that the actual enforcement may depend not only on the action to be performed, but also on the application, it may be necessary for the developer to implement platform specific code. In order to simplify the integration of these custom enforcement mechanisms with the KAoS framework a number of interfaces are provided as a guide for developers.

In a follow up paper (Uszok et al., 2003a), the authors discuss how description logic can be used to support policy administration, exploration and disclosure. Administration is primarily concerned with subsumption based reasoning and the determination of disjointness. Using deductive reasoning it is possible to identify and resolve conflicts at design time. Whereas, exploration and disclosure is supported using instance classification capability. Using abductive reasoning it is possible to both test constraints, and to return relevant constraints given one or more properties.

In Uszok et al. (2003a), the authors propose a general algorithm for conflict resolution and harmonisation, which can be used even when the entities (`actors`, `actions`, `resources` and `policies`) are specified at different levels of abstraction. The proposed conflict resolution strategy is based on policy priorities and timestamps. In the event of a conflict the algorithm takes the policy with the lowest precedence and subdivides it until the conflicting part has been isolated. The conflicting policy is removed, and non conflicting policies are generated and feed into the knowledge base.

3.2.2 Rule Based Approaches

One of the benefits of a rule based approach is that it is possible to support access control policies that contain instance dependencies or variables. Like ontology based approaches, access control policies are defined over ontology entities. Therefore access control policies specified using different vocabularies can easily be integrated. In this section, we examine two different rule based languages and enforcement frameworks, Rei (Kagal and Finin, 2003; Kagal et al., 2003b) and Protune (Bonatti et al., n.d.; Bonatti and Olmedilla, 2005, 2007). Although Rei is a general policy language, it is primarily concerned with the specification and enforcement of policies in ubiquitous environments. Protune, is also a general policy language, however the authors focus primarily on trust negotiation and policy explanations

3.2.2.1 Rei

Rei (Kagal and Finin, 2003; Kagal et al., 2003b) is a Semantic Web policy language and distributed enforcement framework, which is used to reason over policies, that are specified using RDFS or Prolog rules. As OWL has a richer semantics than RDFS, the authors later provided an OWL representation for their policy language (Denker et al., 2005; Kagal et al., 2006). Like KAoS, Rei is a general policy language which can be applied to agents and web services (Denker et al., 2005; Kagal and Berners-lee, 2005). Although Rei policies can be represented using RDFS or OWL the authors adopt a rule based enforcement mechanism, in contrast to the description logic enforcement mechanism adopted by KAoS.

Core entities. The authors propose a number of core ontologies, that are used to describe:

- `entities` (users, agents, services and resources);
- `access rights` (derived from the following speech acts: delegation, revocation, request, cancel, promise and command);
- `actions` (mapping between access rights to entities);
- `conditions` (one or more access rights applied to entities, combined using disjunction, conjunction and negation);
- `policy types` (permissions, prohibitions, obligations and dispensations); and
- `policies` (policy rules and meta-policies).

Specification of policies. Although Rei provides support for four distinct *policy types*, there is a direct mapping between permissions and prohibitions in Rei and the positive and negative authorisations in KAoS, and also between obligations and dispensations in Rei and the positive and negative obligations in KAoS. Given requesters might not be known to the system in advance, constraints are specified in terms of *credentials* (attributes that are used to identify entities). By choosing to represent access rights as speech acts Rei is able to support not only a wide range of policies but also the delegation and revocation of policies. A policy is composed of a set of rules, based on the four policy types, that are used to associate conditions with actions. A `has` predicate is used to associate *permissions* and *obligations* with *entities*. Like KAoS, the core ontologies can be further extended to meet the requirements of specific use cases.

Enforcement of policies. The Rei policy framework, called Rein (Rei and N3), presented in Kagal and Berners-lee (2005) and Kagal et al. (2006), consists of the following components:

- *a set of ontologies*, used to represent Rein policy networks (resources, policies, meta-policies and the Rein policy languages) and access requests; and
- *a reasoning engine*, that uses both explicit and derived knowledge to determine if a request should be granted or denied.

The authors propose a distributed enforcement architecture, whereby each entity is responsible for specifying and enforcing their own policies. Rein is capable of acting as a server or a client. In server mode, Rein retrieves the relevant policies; requests the credentials necessary to access the resource; and verifies the credentials against the policies. Whereas in client mode, the server returns a link to a policy which the client must satisfy; Rein generates a proof that the requester can satisify the policy; and forwards the proof to the server. In order to cater for scenarios where part of the policy is private and part is public, the authors propose a hybrid approach, where Rein acts both as a client and a server.

Using Rein it is possible to combine and reason over different access control policies, meta-policies and policy languages. Policies are expressed using either RDFS or OWL, and inference over both data resources and policies is performed using an N3 reasoner, known as Cwm[5]. N3 was originally used as a representation syntax for RDF, however it was later extended to allow for variables and nested graphs. Cwm extends N3 with inference rules and built-in functions, making it possible to express relationships between graphs, specify both existential and universal constraints and to represent implication. Although the authors demonstrate how the Rei vocabulary can be used to specify policies, these policies could in fact be represented using alternative ontologies.

In Kagal et al. (2003b), the authors discuss how conflict resolution can be achieved using meta-policies. Priority policies are used to indicate dominance between policies or policy rules. While, precedence policies are used to specify a default grant or deny, for *policies*, sets of *actions* or sets of *entities* satisfying specific conditions. In order to guarantee that a decision can always be reached, the authors propose a partial order between meta-policies. Given Rei allows for *policies* to contain variables, conflicts need to be resolved at run-time, as opposed to design time, which is the case with KAoS.

3.2.2.2 Protune

Protune (Bonatti and Olmedilla, 2005, 2007) is a policy language which was proposed by the Research Network of Excellence on Reasoning on the Web, known as REWERSE[6]. Like Rei, Protune adopts a rule based approach to policy enforcement. The authors identify *usability* as one of the primary factors for a policy aware web. To this end, Protune was designed to support

[5]Cwm, http://www.w3.org/2000/10/swap/doc/cwm.html
[6]REWERSE, http://rewerse.net/

both trust negotiation and policy explanations. Lightweight ontologies are used to represent concepts, the relationships between these concepts and details of the evidences needed to prove their truth. Protune is an extension of two other well known policy languages, the Portfolio and Service Protection Language (PSPL) (Bonatti and Samarati, 2000) and PeerTrust (Gavriloaie et al., 2004). PSPL is a model and framework, which uses rules to support policy filtering, policy exchange and information disclosure. Whereas, PeerTrust is a language and a framework, which uses semantic annotations and access control rules, in order to cater for automated trust negotiation and access control.

Specification of policies. Protune policies are specified using rules and meta-rules (essentially horn clauses with some syntactic sugar), which provide support for both deductive and abductive reasoning. The former is used in order to enforce policies, whereas the latter is used in order to retrieve information about the policy conditions that need to be satisfied. Protune provides three predicate categories (*decision* predicates, *provisional* predicates and *abbreviation* predicates).

- *Decision predicates* are used to specify the outcome of a policy.
- *Provisional predicates* are used to represent the conditions the requester must satisfy. By default the system supports two conditions: requests for credentials and request for declarations. Both credentials and declarations are used to assert facts about the requester, however credentials are certified by a third party, whereas declarations are not.
- *Abbreviation predicates*, which are composed of one or more provisional predicates, are used to represent abstractions of the conditions listed in the body of the rule, simplifying policy specification and maintenance.

It is however possible to extend the language, with custom predicate categories. Ontologies are used to associate evidences (descriptive requirements of what is needed to meet the conditions) with access conditions. Evidences of this nature facilitate negotiation. *Metapolicies* are used to assign sensitivity levels to predicates, controlling when actions are executed and constructing new provisional predicates. Metapolicies, containing details of the action and the actor in charge of executing an action, are used to define new provisional predicates.

Enforcement of policies. The enforcement framework is composed of three separate components, a *negotiation handler*, an *execution handler* and an *inference engine*.

- The *negotiation handler* is responsible for sending conditions to the requester and providing responses to conditions that were requested.
- The *execution handler* is used to interact with external systems and data sources.
- The *inference engine* is tasked with both enforcing policies (deduction) and retrieving evidences (abduction).

Like Rei, Protune can be used as a client, as a server, or both. Protune explanations are provided by a component known as protune-x, which supports four different types of queries:

- *How-to queries* (provide a description of the policy).
- *What-if queries* (give foresight into potential policy outcomes).
- *Why queries* (give explanations for positive negotiations outcomes).
- *Why-not queries* (give explanations for negative outcomes).

Protune is developed in Java with a Prolog reasoning component, which is compiled into Java byte code. In Bonatti and Olmedilla (2007), the authors perform a performance evaluation of the negotiation algorithm, using test data which was automatically generated.

Based on the evaluation results the authors concluded that the systems scales well beyond that which would be required for realistic policies.

3.2.3 Combined Ontology and Rule Based Approaches

A hybrid approach to policy specification and enforcement can be used to exploit the out of the box deductive capabilities, of an ontology based approach, and the runtime inference capabilities, of a rule based approach. In this section, we describe Proteus (Toninelli et al., 2005) which uses a combined approach to policy enforcement. We also examine an alternative approach, presented by Kolovski et al. (2007) which demonstrates how description logic based access control policies can be extended with defeasible logic rules.

3.2.3.1 Proteus

Proteus (Toninelli et al., 2005) uses a hybrid approach to semantic policy specification. The authors examine early versions of KAoS and Rei, and highlight the strengths and weaknesses of both ontology based and logic based policy languages and frameworks. Like KAoS the authors use ontologies to model both domain information and policies. Such an approach allows for conflict resolution and harmonisation at design time. Like Protune, policy descriptions are used to facilitate partial policy disclosure and policy negotiation. Like Rei, the authors adopt a rule based approach in order to support dynamic constraints and run time variables. For example, to support access control based on dynamic context pertaining to the requester or the environment.

Core entities. The context-aware adaptive policy model, which they call Proteus (Toninelli et al., 2006, 2007), is composed of several core entities:

- `resource state` (the status of the various entities to be protected);
- `actors` (represented as roles, identities or security credentials);
- `environment` (the surrounding conditions and nearby resources);
- `protection context` (name-value pairs pertaining to resources, actors and the environment); and
- `policies` (a mapping from resources to protection contexts).

Specification of policies. *Policies* are represented as classes and contextual information, relating to the user, are represented as instances. Description logic deduction is used to determine the policies that are relevant for the instance data supplied. However, using description logic reasoning it is not possible to cater for contextual properties that are based on property paths or that are associated with variables. In order to handle reasoning of this nature, the authors propose context aggregation and context instantiation rules. Such rules are represented as horn clauses, with predicates in the head and ontological classes and properties in the body.

Enforcement of policies. A subsequent paper by Toninelli et al. (2009), presents the Proteus policy framework, which is composed of the following core components: a *policy installation manager*, a *reasoning core*, a *policy enforcement manager* and a *context manager*.

- The *policy installation manager* is responsible for loading ontologies, access control policies, contextual information and quality constraints.
- The *reasoning core* performs reasoning over policies, context and quality constraints in order to determine which policies are currently active.
- The *policy enforcement manager* intercepts action requests, collects relevant contextual information and interacts with the *reasoning core* in order to determine if access should be granted or denied.

- The *context manager* collects state information pertaining to system entities and forwards this contextual information to the *reasoning core*.

The authors provide details of their prototype which is implemented in Java with a Pellet reasoner. The proposed solution supports incremental reasoning via an OWL application programming interface and SPARQL queries. Their evaluation shows that incremental reasoning over increasing assertions remains constant. Furthermore in light of increasing assertions, queries with quality constraints are evaluated in constant time and queries without quality constraints demonstrate linear growth.

3.2.3.2 Kolovski et al. (2007)

Kolovski et al. (2007) propose a description logic formalisation for XACML. The authors demonstrate how together description logic and defeasible logic rules, known as defeasible description logic (Governatori, 2004), can be used to understand the effect and the consequence of sets of access control policies.

Enforcement of policies. Although the actual framework is not presented, the following subset of policy services are described:

- **Constraints.** Like Finin et al. (2008b,a) the proposed solution caters for role cardinality and separation of duty;
- **Comparison.** Policies or sets of policies can be compared in order to determine if one is equivalent to or logically contains the other;
- **Verification.** Like Bonatti and Olmedilla (2007), this component checks if the policy satisfies a given property;
- **Incompatibility.** This component provides details of policies that cannot be active at the same time;
- **Redundancy.** This component checks hierarchies to ensure that all policies are reachable; and
- **Querying.** Given a set of attributes, this component searches for relevant policies.

The proposed XACML analysis prototype was implemented on top of Pellet (an open source description logic reasoner). The authors compare their first order logic analysis tool against a propositional logic analysis tool, using identical test data (the access control policy for the *Continue Conference Manager System*). Verification of the policies using their first order logic analysis tool took .42 seconds, whereas the propositional tool took less than a millisecond. The authors attributed this to the fact that the Pellet is optimised for reasoning over more expressive formalisms and conclude that the proposed approach is still acceptable for practical purposes.

3.3 Access Control for RDF

In the previous sections, we saw how access control models and standards can be represented/extended using semantic technology and we examined a number of different policy languages. In this section, we present the different access control mechanisms that have been used to protect RDF data. In particular, we focus on the different mechanisms used to specify access control policies, access control reasoning strategies based on the RDF data model and the different query rewriting strategies.

3.3.1 Specification of Access Control for RDF

Over the years several researchers have focused on the modelling and the enforcement of access control over RDF data. A number of authors (Reddivari et al., 2005; Jain and Farkas, 2006; Abel et al., 2007; Flouris et al., 2010) define access control policies based on RDF patterns, that are mapped to one or more RDF triples. Li and Cheung (2008); Gabillon and Letouzey (2010) and Costabello et al. (2012a) inspired by existing relational database access control strategies, propose a view based access control model for distributed RDF data. Whereas, Sacco et al. (2011) and Costabello et al. (2012a) both propose access control ontologies and enforcement frameworks that rely on SPARQL `ASK` queries to verify if the requester possesses the credentials necessary to access a resource.

3.3.1.1 Triple Patterns

Reddivari et al. (2005) define a set of actions required to manage an RDF store and demonstrate how access control rules can be used to permit or prohibit the requested actions. The actions are organised into four categories:

- `adding` (the insertion of explicit triples, implicit triples and sets of triples);
- `deleting` (the deletion of explicit triples, implicit triples and sets of triples);
- `updating` (directly replacing one triple with another); and
- `querying` (returning triples or using triples to return answers to queries).

Policies are defined using Prolog facts and rules and compiled into Jena rules. Two predicates `permit` and `prohibit` are used to grant and deny access rights based on the aforementioned actions, to one or more triples using triple patterns. Authorisations can be further constrained using conditions relating to policies, triples and agents.

- *Policy specific conditions* relate to the access control policies, for example a user can only add instances if they added the class.
- *Triple specific conditions* correspond to the triple specified in the authorisation, for example if an authorisation governs a triple then all triples associated with a subProperty relation are governed by the same policy.
- *Agent specific conditions* use properties of the user to limit the authorisation, for example it is possible to limit access to users who are managers in a specific company division.

The proposed RDF Store Access Control Policies (RAP) framework checks the policy to ensure that the action is permitted, temporarily allows the action, and afterwards checks the policy to ensure that the inferences are allowed. The authors propose default and conflict preferences that can simply be set to either permit or deny.

Flouris et al. (2010) also use RDF triple patterns to expose or hide information represented as RDF. Although the authors go beyond simple graph patterns by allowing the graph pattern to be constrained by a `WHERE` clause, no consideration is given to either data or policy inference. Like Reddivari et al. (2005), the authors propose a default policy and a conflict resolution strategy. They formally define the semantics of the individual access control statements and the entire access control policy, and present the different possible interpretations for the default semantics and the conflict resolution. A flexible system architecture that demonstrates how the access control enforcement framework can be used with disparate RDF repositories and query languages is presented. The system was implemented using Jena ARQ, Jena SDB with a Postgresql back-end and Sesame. The authors reported a linear increase in processing times over increasing document size, statements in an access control policy, and triple pattern constraints in the `WHERE` clause.

Both Jain and Farkas (2006) and Abel et al. (2007) also use triple patterns to specify access control policies. Information on their approaches is presented under reasoning (*Section* 3.3.2) and query rewriting (*Section* 3.3.3), respectively.

3.3.1.2 Views and Named Graphs

Gabillon and Letouzey (2010) highlight the possible administration burden associated with maintaining access control policies that are based on triple patterns. They propose the logical distribution of RDF data into SPARQL views and the subsequent specification of access control policies, based on existing RDF Graphs or predefined views. The policy language contains the following entities:

- `subjects` (identified using attributes pertaining to users or processes);
- `objects` (represented as either RDF graphs or RDF views); and
- `actions` (that are tightly coupled with the SPARQL `SELECT`, `ASK`, `CONSTRUCT` and `DESCRIBE` query types).

Access control policies are specified using contextual information pertaining to the user, resources or the environment. The body of the rule is a possibly empty condition, or a combination of conditions connected via conjunction or disjunction. The head of the rule is an authorisation. The authors describe an enforcement framework, whereby users define security policies for the RDF graph/views that they own. Users may delegate rights to other users by specifying an authorisation which grants `construct` and `describe` privileges to the RDF Graph or View. Although the authors acknowledge the need for conflict resolution, they do not propose a conflict resolution strategy.

3.3.1.3 Ontology Concepts

Sacco and Passant (2011b,a) and Sacco et al. (2011) demonstrated how an extension of the Web Access Control vocabulary known as the Privacy Preferences Ontology (PPO) can be used to restrict access to an RDF resources, statements and graphs. An access control policy is composed of:

- a restriction in the form of an RDF resource, statement or graph;
- a condition which provides specific details of the restriction, for example `hasProperty`, `hasLiteral`;
- an access privilege (either `read`, `write` or `both`); and
- a SPARQL `ASK` query that must be satisfied by the requester.

The authors describe the formal semantics of the PPO and present a detailed description of their Privacy Preferences Manager (PPM), which can be used to enforce access control using SPARQL `ASK` queries. A follow-up paper Sacco and Breslin (2012) extends the original PPO and PPM to allow access to be restricted based on a dataset or a particular context. The authors also provide support for conflict resolution, more expressive authorisations (in the form of negation and logical operators), and a broader set of access privileges (`create`, `read`, `write`, `update`, `delete`, `append` and `control`). In Sacco et al. (2013), the authors demonstrate how both the PPO and the PPM can be used to cater for fine grained access control on a mobile device. A number of shortcomings of their original enforcement algorithm are identified and a more efficient algorithm which utilises pre-indexing, query analysis and results filtering is presented and evaluated.

An alternative access control vocabulary called the Social Semantic SPARQL Security for Access Control (S4AC) is presented in Villata et al. (2011). Like Sacco and Passant (2011b) they extend the WAC to cater for fine grained access control over RDF data. Their proposal is tightly integrated with several social web and web of data vocabularies, namely: SIOC[7], SCOT[8], NiceTag[9], TIME[10], FOAF[11], Dublin Code[12] and RELATIONSHIPS[13]. The authors define access control policies for named graphs, which can also be used to grant/deny access to sets of triples. S4AC provides support for logical operators and a broad set of access privileges (`create`, `read`, `write`, `update`, `delete`, `append` and `control`) from the offset. Like Sacco and Passant (2011b), SPARQL `ASK` queries are used to determine if the requester has the permissions necessary to access a resource. The authors propose the disjunctive evaluation of policies, thus circumventing the need for a conflict resolution mechanism. Follow up work by Costabello et al. (2012a,b) describes how an access control framework, called Shi3ld, can be used to enforce access control over SPARQL endpoints in a pluggable manner. Although limited implementation details are provided, an evaluation over the Berlin SPARQL Benchmark dataset is performed. The authors conclude that in general there is a small overhead associated with access control. However, when access is granted to a small number of named graphs the query executes faster than the respective query without access control.

In a follow-up paper (Costabello et al., 2013) the authors extend the Shi3ld framework to cater for access control for a Linked Data Platform (LDP)[14] (Speicher et al., 2014). Resources refer simply to Linked Data resources that are queries, created, modified and deleted via HTTP requests processed by a LDP. Two alternative frameworks are presented, one which contains an embedded SPARQL engine and a SPARQL-less solution. In the first scenario, Shi3ld remains unchanged. Whereas in the second scenario, authorisations cannot contain embedded SPARQL queries and therefore are evaluated using subgraph matching. In their evaluation the authors compare all three frameworks, using the Billion Triple Challenge 2012 Dataset[15]. Based on their performance evaluation the authors conclude that access control over SPARQL endpoints is marginally slower than access control over LDP resources and that their SPARQL-less solution exhibits a 25% faster response time.

3.3.2 Reasoning over RDF Access Control Policies

Inference is a process whereby new data is derived from data which is known or assumed to be true. In *Section* 3.2, we discussed how deduction and abduction can be used to simplify both policy specification and maintenance. However, inference can also be used to deduce information, which users should not have access to, commonly known as the *inference problem.* Both Thuraisingham (2007) and Nematzadeh and Pournajaf (2008) highlight the need for security mechanisms to protect against such unauthorised inference. Although this is not a new problem, the authors highlight the fact that with advances in current data integration and mining technologies, the problem is further magnified. According to Qin and Atluri (2003), if the semantic relationship between entities is not taken into account it may be possible to infer information

[7]http://rdfs.org/sioc/spec/
[8]http://scot-project.net/
[9]http://ns.inria.fr/nicetag/2010/09/09/voc.html
[10]http://www.w3.org/TR/2006/WD-owl-time-20060927/
[11]http://xmlns.com/foaf/spec/
[12]http://dublincore.org/documents/dcmi-terms/
[13]http://vocab.org/relationship/
[14]LDP, `http://www.w3.org/2012/ldp/wiki/Main_Page`
[15]BTC2012, `http://km.aifb.kit.edu/projects/btc-2012/`

which has been restricted, or access control policies may not exist for the inferred information making this information inaccessible.

Relational database administrators have tackled the problem using constraints, that are specified when the database is designed, or updated and enforced when the data is queried. A number of different reasoning strategies have been proposed for RDF data. Qin and Atluri (2003); Javanmardi et al. (2006a); Ryutov et al. (2009) and Amini and Jalili (2010) propose strategies for propagation of access rights based on authorisation subjects, access rights and resources. Whereas, Jain and Farkas (2006); Kim et al. (2008) and Papakonstantinou et al. (2012) demonstrate how access rights can be inferred for new triples deduced based on RDFS inference rules. In contrast, Bao et al. (2007) present a number of use cases where it is desirable to grant access to data which has been inferred from unauthorised data. In order to meet this need, the authors present a privacy preserving reasoning strategy.

3.3.2.1 Propagation of Authorisations

Early proposals for the propagation of authorisations focused on reasoning over ontology concepts (Qin and Atluri, 2003). Subsequent work by Javanmardi et al. (2006b) and Javanmardi et al. (2006a) focused not only on ontology concepts but also on reasoning over ontology properties and ontology instances. An alternative strategy which was proposed by Ryutov et al. (2008, 2009), takes a more abstract approach by propagating policies based on nodes and edges in a semantic network.

Reasoning over Concepts. Qin and Atluri (2003), extend XML based access control to take into account the semantic relationships between the concepts that need to be protected. The authors propose a Concept Level Access Control (CLAC) model, which allows for reasoning over concepts appearing in authorisation *subjects*, *permissions* and *objects*. Access control policies are represented using an OWL based vocabulary, which they call the Semantic Access Control Language (SACL) and data instances are defined in domain ontologies. Two properties `SACL:higherLevelThan` and `SACL:lowerLevelThan` are used to specify a partial order between authorisation subjects and permissions. The proposed access control propagation is based on six domain independent relationships (`superclass/subclass`, `equivalence`, `partof`, `intersection`, `union` and `complement`). In the case of `equivalence`, `partof`, `union` and `subclass` positive policies are propagated from subject to object and negative policies are propagated from object to subject. Where there is an `intersection` between two concepts, only negative policies are propagated. In the case of `complement` relations neither positive nor negative authorisations are propagated. The authors acknowledge the need for conflict resolution and simply propose a *negation takes precedence* handling mechanism. Although a motivating scenario is described, no implementation details or evaluation are provided.

Reasoning over concepts, properties and individuals. The Semantic Based Access Control Model (SBAC) proposed by Javanmardi et al. (2006b) and Javanmardi et al. (2006a), builds on the work presented in Qin and Atluri (2003), by catering for access control policy propagation, not only based on the semantic relations between ontology *concepts*, but also based on the relations between *concepts*, *properties* and *individuals*. Like Qin and Atluri (2003), OWL vocabularies are used to represent the authorisation *subjects*, *permissions* and *objects*, however the authorisations themselves are specified using rules. The authors propose the propagation of access rights based on seven different types of inference, from:

- *concept to concept* (where classes are deemed related based on some vocabulary, for example `rdfs:subClass`, `owl:equivalentClass`);

- *concept to individual* (where an entity is a type of class, for example if employee is a class and JoeBloggs `rdf:type` employee);
- *individual to individual* (using properties such as `owl:sameAs` it is possible to propagate entities that represent the same thing);
- *property to concept* (if access is granted to the property access should be granted to the classes governed by `rdfs:domain` and `rdfs:range`);
- *property to property* (where properties are deemed related based on some vocabulary, for example `rdfs:subProperty`, `owl:equivalentProperty`);
- *property to individual* (where an entity is a type of property, for example if roles is a property and manager is of `rdf:type` role); and
- *concept to property* (where access is granted to a concept it should also be granted to all properties relating to that concept).

The authors describe how the aforementioned semantic relations can be reduced to subsumption relations and propose a general propagation strategy for subsumption relations among *subjects*, *permissions* and *objects*. In the case of subjects and objects, both positive and negative access rights propagate from subsumee to subsumer. However, in the case of *permissions* positive access rights propagate from subsumee to subsumer, while negative access right propagate from subsumer to subsumee. Although an architecture is presented by Javanmardi et al. (2006a), very little detail on the actual enforcement mechanism is supplied. Their evaluation, which is performed over increasing datasets, highlights that their custom reasoner over the reduced ontology performs considerably better than a Pellet reasoner over the standard ontology.

Follow-up papers by Ehsan et al. (2009) and Amini and Jalili (2010) build on previous work, by providing for an access control model with formal semantics and an enforcement framework, which is suitable for distributed semantic aware environments (for example Semantic Web, Semantic Grid and Semantic Cloud Computing). Policy rules, in both the conceptual and individual levels, are specified using a combination of deontic and description logic, which they refer to as $MA(DL)^2$. The prototype consists of:

- a user interface developed using the Google Web Toolkit;
- a data reasoner implemented in Jena; and
- a tableaux reasoner implemented in Prolog.

The authors present the results of a performance evaluation over increasing policy rules, where the decision to grant or deny access is based on ground policies, inferred policies and the proposed conflict resolution strategy. The authors conclude that real-time reasoning is expensive. Therefore they suggest:

(i) using the parallelisation facilities of the tableaux system;
(ii) adopting a proof based approach where the requester presents authorisation rules that demonstrate they can access the requested resource; and
(iii) materialisation of inferred relations in advance.

Reasoning based on the semantic network. Ryutov et al. (2008, 2009) propose a policy language which can be used to specify access control in terms of the semantic relationships between the nodes and edges of a graph. The policy language is composed of the following entities:

- `Subjects` (the union of users and groups);
- `Permissions` (operations to be performed on resources);
- `Resources` (rdf resources);

- `Links` (rdf properties);
- `Types` (rdf classes);
- `Objects` (the union of resources, links, properties and types); and
- `Entities` (the union of subjects and objects).

The following predicates are used to describe the structure of the semantic network:

- `link` is used to specify a relationship between two entities;
- `property` is used to attach a literal property to an entity;
- `type` is used to specify an instance of a class; and
- `permit` is used to denote a subject is assigned permission to a resource.

In order to cater for policy propagation, two directed acyclic graphs are used to represent the relationship between users and groups (using a `memberOf` property) and between objects and bundles (using a `partOf` relation). Propagation policies are defined to allow for policy propagation based on both the `partOf` and `memberOf` relations. The authors propose a conflict resolution algorithm, which is based on the semantic network. When a policy explicitly refers to a node, the policy distance is zero. Whereas, when a policy is implicitly assigned, based on a propagation policy, the distance is determined by counting the number of nodes in the path from the node with the explicit policy. The smaller the distance, the more specific the policy. If multiple policies exist at different distances, the most specific policy takes precedence. If multiple explicit conflicting policies exist, conflicts are resolved using logical conjunction. In the case of implicit policies, conflicts are resolved using logical disjunction. The authors also propose safety and consistency policies that are used to prevent undesirable access control policy specification and propagation, for example resources that nobody can access. A specific implementation which limits access to semantic objects, files and executables is presented, however no formal evaluation is performed.

3.3.2.2 RDFS Inference

When it comes to RDFS inference, there are two different strands of research. The first infers access rights for triples that are inferred using RDFS entailment rules (Jain and Farkas, 2006; Kim et al., 2008). Whereas the second uses RDFS entailment rules to propagate permissions for triples that already exist (Papakonstantinou et al., 2012).

Infer access right for new triples. Jain and Farkas (2006), demonstrate how RDFS entailment rules can be used not only to infer new RDF triples, but also to infer access control annotations for those triples. The authors use RDF triple patterns and associated security classifications, known as security labels, to limit access to RDF statements. They define a subsumption relationship between patterns and stipulate that subsuming patterns must be as restrictive as the subsumed patterns. In addition, they define a partial order between security labels, which is used to determine the security classification of triples inferred via RDFS entailment rules. If more than one pattern maps to a statement the most restrictive or the lowest upper bound takes precedence. The authors provide formal definitions for each of the RDF security objects and define an algorithm to generate security labels for both explicit and inferred triples based on a security policy and a conflict resolution strategy. Limited details of the implementation are supplied and no evaluation is performed.

Kim et al. (2008) demonstrate how together authorisations and RDFS inference rules can be used to generate new authorisations. An authorisation is defined as a four tuple $\langle sub, obj, sign, type \rangle$, where *sub* refers to the access control subject; *obj* is represented an RDF triple pattern; *act* is the operation; *sign* indicate if access is granted or denied; and *type* which is either *R* to indicate that the authorisation should be propagated or *L* if it should not. The authors discuss how `rdfs:subClass`, `rdfs:subProperty` and `rdf:type` inference can be used to infer new authorisation objects (triple patterns) and consequently new authorisations. In addition, they examine the different scenarios which might result in access to data being both permitted and prohibited. If the authorisation that is prohibited is more specific than the authorisation that is permitted based on the subclass/subproperty hierarchy then access should be denied. In order to determine if there is a conflict, only authorisations with a superclass or superproperty that is negative need to be checked.

Infer and propagate access right for new triples. Papakonstantinou et al. (2012) propose a strategy for associating access control annotations results using RDFS inference and propagation policies. In the proposed strategy access to triples is granted/denied based on annotations that are assigned to triples. In the presented modelling both the triples and the corresponding annotations are represented as quads. Annotations can be:

- directly associated with triples;
- inferred using RDF inference rules;
- propagated using RDF inference rules; or
- assigned a default label.

The authors demonstrate how the RDFS `subClass`, `subProperty` and `type` inference rules can be used to assign annotations to inferred triples. However, they do not dictate how the access control annotations assigned to the premises should be combined, but rather propose an abstract operator which can be adapted to suit particular use cases. In addition, the authors demonstrate how the RDFS `subClass`, `subProperty` and `type` inference rules can be used to propagate permissions to existing triples. As per inference rules existing annotations and propagated annotations are inferred by means of a domain operator. The triples, access control labels and details of the triples that are used to derive an annotation are stored in a relational database. The database is composed of a *Quad* table, a *LabelStore* table and a *Map* table. The Map table is used to associate a unique id with each URI and literal. When labels are associated with triples directly, the annotation is represented in the *label* attribute of the *Quad* table. However in the case of inferred or propagated labels, a reference is placed in the *LabelStore* for each of the triples that are used to generate the annotation. The authors evaluate their prototype over both ProgresSQL and MonetDB relational databases. Based on their performance evaluation of both the inference and propagation rules, the authors concluded that more efficient storage and indexing schemes are required.

3.3.2.3 Reasoning over restricted data

When it comes to reasoning over restricted data, there is a general consensus that any information that can be inferred from restricted data should also be restricted. An alternative viewpoint is presented by Bao et al. (2007). The authors focuses on a number of use cases where it is desirable to grant access to information that has been inferred from restricted data:

(i) a calendar showing the existence of an appointment without revealing specifics;

(ii) a booking engine sharing partial hotel details; and

(iii) a pharmacy confirming that the patients drugs are reimbursable without disclosing details.

The open world assumption is used to ensure that users cannot distinguish between information which does not exist and information with is inaccessible. The authors stipulate that, the knowledge base should not lie, the answers given should be independent of any previous answers and it should not be possible to infer any restricted data. The proposed strategy is based on the notion of conservative extension. Essentially the reasoner keeps a history of the answers to all previous queries. For each subsequent query, the history is consulted, in order to verify that unauthorised information cannot be inferred by the requester.

3.3.3 Partial Query Results

A number of the access control mechanisms for RDF data that we have examined, demonstrate how their access control can be enforced on top of SPARQL queries. However, the solutions we have examine thus far either grant or deny access to the entire query. In this section, we examine the different strategies that can be used to ensure that access to unauthorised data is not permitted. Such approaches can be used to grant access in the general case and deny access to specific data.

3.3.3.1 Data Filtering

Dietzold and Auer (2006) examine access control requirements for an RDF Store from a semantic wiki perspective. The authors propose access control policy specification at multiple levels of granularity (*triples*, *classes* and *properties*). In addition, they define three atomic actions (`read`, `insert` and `delete`) for both individual triples and sets of triples. Authorisations are used to generate a virtual model of the data, upon which user queries are executed. Authorisations are used to associate filters (SPARQL `CONSTRUCT` queries) with users and resources. When a requester submits a query, a virtual model is created based on the matched authorisations. The query is executed against the virtual model, which only contains data the requester is authorised to access.

Muhleisen et al. (2010) describe a Policy-enabled Linked Data Server (PeLDS), which uses WebID to authenticate users. The policy language caters for the specification of access control policies for particular *triple patterns*, *resources* or *instances*, using SWRL rules. An OWL ontology is used to identify the rule types (`single concept` and `triple pattern`) and supported actions `query` and `update`. Negation is not supported in the presented modelling. When a requester submits a query, the system uses their WebID to determine the data instances that the user has been granted access to and generates a temporary named graph containing authorised data. The requesters query is subsequently executed against the temporary named graph and the results are returned to the user.

3.3.3.2 Query Rewriting

To date a number of query writing strategies have been proposed. Different strategies involve creating bindings for variables and adding them to the query `WHERE` clause in the case of positive authorisations, or the query `MINUS` clause in the case of negative authorisations (Abel et al., 2007). Binding user attributes to path expressions (Franzoni et al., 2007). Using `FILTER` expressions to filter out inaccessible data (Chen and Stuckenschmidt, 2010; Oulmakhzoune et al., 2010). Limiting the query to a specific named graph (Costabello et al., 2012a) or rewriting a view so that it considers propagation rules and both instance and range restrictions (Li and Cheung, 2008).

Bindings added to where or minus clause. Abel et al. (2007) propose the evaluation of access control policy constraints at both the query layer and the data layer. Contextual conditions that are not depended on RDF data are evaluated by a policy engine. Whereas the query is expanded to include the contextual conditions that are dependent on RDF data.

Such an approach requires the substitution of variables to ensure uniqueness, however in doing so they are able to leverage the highly optimized query evaluation features of the RDF store. In the presented modelling, both positive and negative authorisations are composed of sets of *contextual predicates*, *path expressions* and *boolean expressions*. Queries are assumed to have the following structure `SELECT/CONSTRUCT` ***`RF`*** `FROM` ***`PE`*** `WHERE` ***`BE`***, where ***`RF`*** represents the result form (projections in the case of `SELECT` queries and `triples` in the case of `CONSTRUCT` queries); ***`PE`*** denotes the path expression; and ***`BE`*** corresponds to one ore more boolean expressions connected via conjunction or disjunction operators. An authorisation is deemed applicable if the triple pattern the policy is protecting, is part of either the ***`PE`*** or the ***`BE`***, and the corresponding contextual predicates, path expressions and boolean expressions are satisfied. The authors propose a query rewriting algorithm, which constructs bindings for authorisation path expressions and contextual predicates. For positive authorisations the bindings are appended to the query `WHERE` clause. Whereas, for negative authorisations the bindings are added to a `MINUS` clause, which in turn is appended to the query. The authors conclude that the proposed rewriting strategy, which was evaluated over a Sesame database, increases linearly with additional `WHERE` clauses.

Binding user attributes to path expressions. Franzoni et al. (2007) propose a query rewriting strategy, which is used to grant/deny access to ontology instances. The authors rewrite queries to take into account contextual information, pertaining to the user or the environment. A fine grained access control (FGAC) policy is defined as a tuple $\langle target, \langle property, attribute, operator \rangle \rangle$ where:

- *target* is the resource that the policy relates to;
- *property* is a path expression, which either directly or indirectly relates to the resource;
- *attribute* is the user attributes, that are bound to the path expression variables; and
- *operator* is the filter condition.

The authors propose a two tiered approach to access control enforcement. Access control policies are used to determine if access should be granted or denied. FGAC policies are only applied if access is granted. If the query contains one ore more FGAC policy targets, the query is rewritten to include the path expression and a `WHERE` clause, which is composed of an expression generated from the variables in the path expression, the attributes of the requester and the operator.

Optionals and filters. Chen and Stuckenschmidt (2010) present a query rewriting strategy, which can be used to restrict access to data represented using ontologies. The authors focus on restricting access to instance data. Access control policies are used to deny access to specific *individuals* or to grant/deny access to instances associated with a given *class* or *property*. When access is prohibited to specific *individuals*, a `FILTER` expression is generated, which ensures that none of the query variables bind to the prohibited individuals. When access is granted to *predicates* or *classes*, a `FILTER` expression is generated, which binds the relevant variables in the query to the specified predicate or class. Whereas, when access is prohibited to *predicates* or *classes*, the matching triple patterns are made `OPTIONAL` and a `FILTER` expression is generated, which ensures that the corresponding variables do not bind to the specified predicate or class, and variables that are `!BOUND` are not returned.

Oulmakhzoune et al. (2010) propose a query rewriting strategy for SPARQL queries. In the presented modelling, both positive and negative authorisations are composed of sets of filters that are associated with simple conditions or involved conditions.

Given a SPARQL query the algorithm examines each individual *basic graph pattern* (BGP). In the case of:

- *simple conditions*, when authorisations permit/deny access to a single triple pattern, the following query rewriting strategy is applied:
  - If all authorisations that match the triple pattern, permit access to the triple pattern, no action is required.
  - If all authorisations prohibit access to the triple pattern, the triple pattern is deleted.
  - Otherwise, if the BGP is converted to an `OPTIONAL` BGP, and the authorisation `FILTER` expression is added to the query.
- *involved conditions*, where authorisations permit/deny access for a given *predicate*, the following query rewriting strategy is applied:
  - In the case of positive authorisations, if the query contains a triple pattern which matches the predicate of the authorisation, the `FILTER` condition is added. Alternatively both the triple pattern and the corresponding `FILTER` are added.
  - In the case of negative authorisations, if the query contains a triple pattern which matches the predicate of the authorisation, and the object is a variable, both the `FILTER` condition and a `!BOUND` expression are added.
  - Alternatively the triple pattern, the corresponding `FILTER` condition and a `!BOUND` expression are added.

Named graph added to the query. Costabello et al. (2012a) restrict access to named graphs using query rewriting. An access control policy is a tuple $\langle ACS, AP, S, R, AEC\rangle$, where ACS is a set of access conditions (specified using SPARQL `ASK` queries); AP is a set of access privileges (`CREATE`, `READ`, `UPDATE` or `DELETE`); S denotes the subjects to be protected; R represents the named graphs to be protected; and AEC is the evaluation context specified using name value pairs (verified using SPARQL `BINDINGS`). In addition to the SPARQL query that the user wishes to execute, the user provides their access credentials, in the form of a SPARQL `UPDATE` query, which contains contextual data. The enforcement framework stores the contextual data in a named graph and retrieves the authorisations that match the query type. In order to determine if access is permitted, the `ASK` query and the `BINDINGS`, that are specified in the authorisation, are executed against the users contextual graph. If the `ASK` query returns true then the query is rewritten to include the corresponding named graph.

Expanding views based on propagation rules and instance and range restrictions. Li and Cheung (2008) propose a query rewriting strategy for views generated from ontological relations. An access control policy is defined as a tuple $\langle s, v, sign\rangle$, where s denotes the subject, v represents a set of concepts, relations and filters and $sign$ is used to indicate permissions and prohibitions. Both views and queries that are also generated from sets of concepts, relations and filters are represented using rules. Propagation policies are used to generate implicit authorisations from explicit authorisations, based on subsumption relations between access control subjects and subsumption relations between concepts, appearing in the body of the view. The proposed query rewriting strategy involves: (i) retrieving the policies applicable to the subject, taking into account the subject propagation rules; (ii) expanding each of the concepts in the body of the view based on the concept propagation policies; and (iii) applying the relevant range and instance restrictions to the query based on the expanded view.

3.4 Access Control Requirements for Linked Data

More recently, the focus has shifted to the specification and enforcement of access control for Linked Data. Although Sacco et al. (2011) and Costabello et al. (2013) describe how their frameworks can be used in conjunction with Linked Data, each of the access control mechanisms we have examined thus far could potentially be used to enforce access control over Linked Data. However, as it currently stands there is still no formal recommendation for access control over Linked Data. Therefore, in this section we provide a summary of existing requirements, and present an overview of the general policy languages presented in *Section* 3.2 and the different access control strategies described in *Section* 3.3.

The requirements we use are derived from several papers that examine access control for RDF from a number of perspectives. Yagüe et al. (2003) examine the different layers of the Semantic Web and how the technologies and concepts can be applied to access control. Both Damiani et al. (2005) and Weitzner et al. (2006b) focus on the access control mechanisms that are required to support new access control paradigms where user privacy is a key requirement. De Coi et al. (2008) and Bonatti and Olmedilla (2007) investigate the interplay between trust, access control and policy languages. While, Ryutov et al. (2009) focus more on the data model, investigating access control requirements from a graph perspective, as opposed to the traditional hierarchical approach. In this section, we correlate and categorise the RDF access control requirements identified by each of the aforementioned researchers.

3.4.1 Specification

Generally speaking, access control policy specification requirements relate to the types of policies that can be expressed, and the interoperability of the chosen representation format. An overview of each of the requirements relating to access control specification is presented below and a summary of existing proposals is depicted in *Table* 3.1. One requirement, which we have chosen not to include is ***monotonicity***. According to Bonatti and Olmedilla (2007), the addition of new evidences and policies should not negate any of the previous conclusions. However, given the need to support negative access control policies, and also changes in contextual constraints, we would argue that access control should in fact be ***non-monotonic***.

Granularity (Ryutov et al., 2009; Amini and Jalili, 2010). Ryutov et al. (2009) adopt a graph perspective, stating that it should be possible to specify access control rules for nodes (entities) and edges (semantic relationships between entities). Whereas, Amini and Jalili (2010) adopt an ontological view, stating that it is necessary to specify policies for both ontology concepts and individuals. Existing access control strategies for RDF, resources are specified at several different levels of granularity. Namely, triples (Dietzold and Auer, 2006; Papakonstantinou et al., 2012; Sacco and Passant, 2011b), named graphs (Costabello et al., 2012b; Gabillon and Letouzey, 2010; Sacco and Passant, 2011b), views (Li and Cheung, 2008), triple patterns (Jain and Farkas, 2006; Kim et al., 2008; Muhleisen et al., 2010; Reddivari et al., 2005), graph patterns with filters (Abel et al., 2007; Chen and Stuckenschmidt, 2010) and graph patterns without filters (Flouris et al., 2010), classes and properties (Dietzold and Auer, 2006), ontology concepts (Bonatti and Olmedilla, 2007; Amini and Jalili, 2010; Ehsan et al., 2009; Javanmardi et al., 2006a; Kagal and Finin, 2003; Kolovski et al., 2007; Oulmakhzoune et al., 2010; Qin and Atluri, 2003; Sacco and Passant, 2011b; Toninelli et al., 2009; Uszok et al., 2003b), ontology individuals (Amini and Jalili, 2010; Ehsan et al., 2009; Franzoni et al., 2007; Javanmardi et al., 2006a; Sacco and Passant, 2011b) and graph nodes and edges (Ryutov et al., 2009). In the vast majority of cases, access is either granted or denied. However, a number of researchers have investigated returning partial results to the users. Such strategies either involve dataset filtering (Dietzold and Auer, 2006) or query rewriting using filters (Abel et al., 2007; Chen and Stuckenschmidt, 2010; Flouris et al., 2010; Franzoni et al., 2007; Muhleisen et al., 2010) or named graphs (Costabello et al., 2012b). In *Table* 3.1 the granularity of the authorisations and the different strategies used to cater for partial query results are represented as *Granularity* and *Partial Results* respectively.

Underlying Formalism (De Coi et al., 2008; Amini and Jalili, 2010; Bonatti and Olmedilla, 2007). Access control languages should be based on formal semantics, as it decouples the meaning of the policies from the actual implementation. The majority of researchers either adopt formalisms based on logic programming (Bonatti and Olmedilla, 2007; Kagal and Finin, 2003; Toninelli et al., 2009) or different flavors of description logic (Amini and Jalili, 2010; Bao et al., 2007; Chen and Stuckenschmidt, 2010; Javanmardi et al., 2006a; Kolovski et al., 2007; Toninelli et al., 2009). While, Kagal and Finin (2003) demonstrate how logic programming can be combined with deontic logic, Amini and Jalili (2010) demonstrate how description logic can be combined with deontic logic and Kolovski et al. (2007) combine description logic and defeasible logic. Whereas, Ryutov et al. (2009) adopt a many sorted first order logic formalism.

Reasoning (Damiani et al., 2005; Ryutov et al., 2009; Amini and Jalili, 2010). It should be possible to propagate policies based on the semantic relations between authorisation subjects, objects and access rights. Using ontologies, rules or a combination of both, it is possible to perform deductive reasoning and abductive reasoning over access control policies (Amini and Jalili, 2010; Bonatti and Olmedilla, 2007; Chen and Stuckenschmidt, 2010; Kagal and Finin, 2003; Kolovski et al., 2007; Muhleisen et al., 2010; Toninelli et al., 2009; Uszok et al., 2003b). In addition, a number of authors have proposed propagation strategies based on RDFS entailment (Franzoni et al., 2007; Jain and Farkas, 2006; Kim et al., 2008; Qin and Atluri, 2003; Reddivari et al., 2005) and hierarchies, partial orders or ontological relations between RDF resources (Javanmardi et al., 2006a; Papakonstantinou et al., 2012; Ryutov et al., 2009). Unlike the other authors, who use reasoning to either infer or to propagate access control policies, Bao et al. (2007) demonstrates how it is possible to reason over restricted data without releasing any restricted information. However, the interplay between the various reasoning strategies proposed, and the access control requirements arising from concrete use cases remains an open issue.

Condition Expressiveness (De Coi et al., 2008; Bonatti and Olmedilla, 2007). When it comes to access control, it should be feasible to specify conditions under which a request will be permitted or prohibited. However, many of the general policy languages highlight the need to also cater for obligation and dispensation policies (Amini and Jalili, 2010; Kagal and Finin, 2003; Toninelli et al., 2009; Uszok et al., 2003b).

Attributes, Context & Evidences (Damiani et al., 2005; De Coi et al., 2008; Bonatti and Olmedilla, 2007; Amini and Jalili, 2010). As the requester many be unknown to the system prior to submitting a request, access should be based on properties pertaining to the requester, commonly know as attributes, instead of traditional identities (Amini and Jalili, 2010; Bonatti and Olmedilla, 2007; Costabello et al., 2012b; Franzoni et al., 2007; Gabillon and Letouzey, 2010; Kagal and Finin, 2003; Kolovski et al., 2007; Reddivari et al., 2005; Ryutov et al., 2009; Sacco and Passant, 2011b; Toninelli et al., 2009). It should also be feasible to dynamically activate policies based on context (Abel et al., 2007; Amini and Jalili, 2010; Bonatti and Olmedilla, 2007; Costabello et al., 2012b; Franzoni et al., 2007;

Gabillon and Letouzey, 2010; Kagal and Finin, 2003; Papakonstantinou et al., 2012; Sacco and Passant, 2011b; Toninelli et al., 2009; Uszok et al., 2003b). Context can relate to the requester, the system or the environment. Attributes and context should be communicated by means of digital certificates, known as evidences. The de facto standard for submitting evidences is WebID (Muhleisen et al., 2010; Costabello et al., 2012b; Sacco and Passant, 2011b). However, a number of researchers have also proposed using OpenID (Bonatti and Olmedilla, 2007; Muhleisen et al., 2010; Sacco and Passant, 2011b) .

Heterogeneity & Interoperability (Yagüe et al., 2003). Access control for open distributed environments, such as the web, needs to be able to support a wide variety of disparate policies, resources and users. One of the primary goals of standardisation is to maximize interoperability. As each of the access control strategies we examine use open standards, such as RDF, RDFS, OWL and SPARQL, regardless of the specific use case they are suitable for access control over Linked Data. Using ontologies it is possible to specify access control vocabularies that can easily be adopted by others. In addition, OWL predicates such as `owl:sameAs` and `owl:disjointFrom` can be used to merge different access control vocabularies.

Table 3.1: Specification requirements

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Interoperability
Abel et al.	graph patterns & filters	bindings & filters	-	-	permissions±	context	RDF & SPARQL & SeRQL
Amini and Jalili & Ehsan et al.	ontology concepts & individuals	-	DL & deontic logic	deduction & abduction	permissions± obligation±	attributes & context	OWL
Bao et al.	ontology concepts	-	DL *SHIQ*	privacy preserving	-	-	OWL
Bonatti and Olmedilla	ontology concepts	-	LP	deduction & abduction	permissions±	attributes & context & OpenID	RDF
Chen and Stuck-enschmidt	graph patterns & filters	bindings & filters	DL	deduction & abduction	permissions±	-	OWL & SPARQL
Costabello et al.	named graphs	named graphs	-	-	permissions±	attributes & context & WebID	RDF & SPARQL
Dietzold and Auer	triples, classes & properties	data filtering	-	-	permissions	-	RDF
Flouris et al.	graph patterns	bindings & filters	-	-	permissions±	-	RDF & SPARQL
Franzoni et al.	ontology individuals	bindings & filters	-	RDFS entailment	permissions±	attributes & context	RDFS & SPARQL
Gabillon and Letouzey	named graph & views	-	-	-	permissions±	attributes & context	RDF & SPARQL
Jain and Farkas	triple patterns	-	-	RDFS entailment	permissions±	-	RDFS

continued overleaf

Table 3.1 – continued from previous page

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Heterogeneity & Interoperability
Javanmardi et al.	ontology concepts & individuals	-	DL *SHOIN*	subjects, predicates & objects subsumption	permissions±	-	OWL
Kagal and Finin	ontology concepts	-	LP	deduction & abduction	permissions± obligation±	attributes & context	OWL
Kim et al.	triple pattern	-	-	RDFS entailment	permissions±	-	RDFS
Kolovski et al.	ontology concepts	-	DL *SHOIN* & defeasible logic	deduction & abduction	permissions±	attributes	OWL
Li and Cheung	views	propagation	-	-	permissions±	-	RDF
Muhleisen et al.	triple patterns	data filtering	-	deduction & abduction	permissions+	WebID & OpenID	OWL
Oulmakhzoune et al.	ontology concepts	bindings & filters	-	-	permissions±	-	RDF & SPARQL
Papakonstantinou et al.	triples	-	-	RDFS entailment	permissions±	context	RDFS & SPARQL
Qin and Atluri	concept	-	-	ontology concept relations	permissions±	-	RDF
Reddivari et al.	triple patterns	-	-	RDFS entailment	permissions±	attributes	RDFS
Ryutov et al.	nodes & edges	-	many sorted first order logic	subject & object subsumption	permissions±	attributes	RDF

continued overleaf

Table 3.1 – continued from previous page

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Heterogeneity & Interoperability
Sacco and Passant	resource, triple & graph	-	-	-	permissions±	attributes & context & WebID & OpenID	RDF & SPARQL
Toninelli et al.	ontology concepts	-	DL & LP	deduction & abduction	permissions± obligation±	attributes & context	OWL
Uszok et al.	ontology concepts	-	DL	deduction & abduction	permissions± obligation±	context	OWL

3.4.2 Enforcement

Access control enforcement requirements refer to constraints that are placed on the policy language or mechanisms that assist the requester to complete their request. An overview of the requirements is presented below and a snapshot of existing support for said requirements is presented in *Table* 3.2.

Negotiation (Damiani et al., 2005; De Coi et al., 2008; Bonatti and Olmedilla, 2007). In other to protect user privacy, it should be possible for both the service provider and the requester to define polices and exchange credentials until and agreement has been reached. The process is commonly known as negotiation. The P3P recommendation and the APPEL vocabulary have been designed to support automatic negotiation between clients and servers. The access control mechanisms proposed by Amini and Jalili (2010), Bonatti and Olmedilla (2007) and Toninelli et al. (2009) all cater for access control negotiation.

Explanations (De Coi et al., 2008; Bonatti and Olmedilla, 2007). Rather than simply granting or denying access, the policy should also provide details of how the decision was reached. Such explanations would be of benefit to both the requester and the policy owner, making it easier for the requester to understand what is required of them and for the policy owner to troubleshoot potential problems. Bonatti and Olmedilla (2007), Ryutov et al. (2009) and Toninelli et al. (2009) all provide policy explanations. However, both Ryutov et al. (2009) and Bonatti and Olmedilla (2007) provide a means to execute queries over policies in order to obtain additional information.

Conflict Resolution (Amini and Jalili, 2010). Conflicts between both explicit and implicit policies should be resolved automatically. A number of different conflict resolution strategies have been proposed. In the event of a conflict some authors simply default to grant/deny (Abel et al., 2007; Costabello et al., 2012b; Flouris et al., 2010; Gabillon and Letouzey, 2010; Papakonstantinou et al., 2012; Qin and Atluri, 2003), use priorities to determine dominance (Kolovski et al., 2007) or use metapolicies as a flexible means to resolve conflicts (Kagal and Finin, 2003; Reddivari et al., 2005). A number of authors propose conflict resolution algorithms based on several different measures (Amini and Jalili, 2010; Bao et al., 2007; Jain and Farkas, 2006; Javanmardi et al., 2006a; Kim et al., 2008; Ryutov et al., 2009). Whereas, others try to isolate individual data items that are in conflict and propose harmonisation strategies (Li and Cheung, 2008; Toninelli et al., 2009; Uszok et al., 2003b).

3.4.3 Administration

In this section, we examine a number of access control requirements that are necessary to simplify the specification and maintenance of access control policies. An overview of the requirements is presented below and a summary of current support is presented in *Table* 3.3. Although a number of researchers indicate that they provide some level of support for these requirements, generally speaking research efforts seem to focus more on the specification and enforcement mechanisms, and very little detail is supplied.

Delegation (Bonatti and Olmedilla, 2007). It should be feasible to temporarily transfer access rights to other users. In relational databases, users are granted sole ownership of the tables and views that they create. They can subsequently grant access rights to other database users. Amini and Jalili (2010), Bonatti and Olmedilla (2007), Gabillon and Letouzey (2010) and Kagal and Finin (2003) indicate that they support the delegation of

Table 3.2: Enforcement requirements

	Negotiation support	Explanation	Conflict Resolution
Abel et al.	-	-	default
Amini and Jalili &Ehsan et al.	bidirectional policies	-	algorithm
Bao et al.	-	-	algorithm
Bonatti and Olmedilla	bidirectional policies	queries	-
Chen and Stuckenschmidt	-	-	-
Costabello et al.	-	-	default
Dietzold and Auer	-	-	-
Flouris et al.	-	-	default
Franzoni et al.	-	-	-
Gabillon and Letouzey	-	-	default
Jain and Farkas	-	-	algorithm
Javanmardi et al.	-	-	algorithm
Kagal and Finin	-	-	meta policies
Kim et al.	-	-	algorithm
Kolovski et al.	-	-	priorities
Li and Cheung	-	-	harmonisation
Muhleisen et al.	-	-	-
Oulmakhzoune et al.	-	-	-
Papakonstantinou et al.	-	-	default
Qin and Atluri	-	-	default
Reddivari et al.	-	-	meta policies
Ryutov et al.	-	user interface	algorithm
Sacco and Breslin	-	-	-
Toninelli et al.	bidirectional policies	descriptions	harmonisation
Uszok et al.	-	-	harmonisation

access rights. However, the suitability of existing revocations strategies, for the RDF graph model, warrants further research.

Consistency & Safety (Ryutov et al., 2009). In order to ensure the access control system is complete and accurate, insertion and deletion of policies should be controlled. It should not be possible to elevate your own privileges or to assign permissions that would make data inaccessible to everyone. Although a number of researchers indicate that their frameworks support consistency and safety constraints, very little information is provided. Amini and Jalili (2010), Bao et al. (2007) and Jain and Farkas (2006) ensure consistency and safety as part of their administration algorithms. Chen and Stuckenschmidt (2010) and Ryutov et al. (2009) suggest that meta policies can be used to ensure consistency and safety. Given the diversity of access control models, policies and reasoning strategies that have been proposed for RDF, additional research is required in order to determine potential issues with access control policies and propose suitable handling mechanisms.

Usability (Damiani et al., 2005; Amini and Jalili, 2010). The specification and the maintenance of access control policies should be as simple as possible. Administration facilities that support ease of both specification and maintenance of policies have been provided by a number of researchers (Jain and Farkas, 2006; Ryutov et al., 2009; Sacco and Passant, 2011b; Uszok et al., 2003b). Given the complexity associated with reasoning over

Table 3.3: Administration requirements

	Delegation	Consistency & Safety	Usability	Understandability
Abel et al.	-	-	-	-
Amini and JaliliEhsan et al.	case study	algorithm	-	-
Bao et al.	-	algorithm	-	-
Bonatti and Olmedilla	language	-	-	ProtuneX explanation
Chen and Stuckenschmidt	-	meta policies	-	-
Costabello et al.	-	-	-	-
Dietzold and Auer	-	-	-	-
Flouris et al.	-	-	-	-
Franzoni et al.	-	-	-	-
Gabillon and Letouzey	construct & describe	-	-	-
Jain and Farkas	-	algorithm	RACL admin module	-
Javanmardi et al.	-	-	-	-
Kagal and Finin	speech acts	-	-	-
Kim et al.	-	-	-	-
Kolovski et al.	-	-	-	analysis services
Li and Cheung	-	-	-	-
Muhleisen et al.	-	-	-	-
Oulmakhzoune et al.	-	-	-	-
Papakonstantinou et al.	-	-	-	-
Qin and Atluri	-	-	-	-
Reddivari et al.	-	-	-	-
Ryutov et al.	-	meta policies	RAW policy editor	RAW permission check
Sacco and Breslin	-	-	privacy preference manager	-
Toninelli et al.	-	-	-	-
Uszok et al.	-	-	KAoS Policy Admin Tool	policy disclosure

graph data, advanced data analytics and visualisation techniques are needed to highlight the effects of advanced policies, constraints and deduction rules.

Understandability (Ryutov et al., 2009; Amini and Jalili, 2010). It should be easy to understand the interplay between policies. Earlier we saw that a handful of researchers associate policy explanations with policies. Similarly, only a select few provide systems that enable administrators to verify the interplay between policies (Bonatti and Olmedilla, 2007; Kolovski et al., 2007; Ryutov et al., 2009; Uszok et al., 2003b). Given the dynamic nature of context based access control and the various deduction and propagation strategies, further research on automating the explanations and presenting the results in a manner which is digestible by humans is necessary.

3.4.4 Implementation

Implementation requirements generally refer to non-functional requirements. As with any software system, non-functional requirements hold the key to the adoption of a tool or technology. Although a number of authors indicate that the solutions they propose are flexible or extensible,

Table 3.4: Implementation requirements

	Effectiveness	Distributed Use case	Flexibility & Extensibility
Abel et al.	query rewriting performance	-	SeRQL, Protune
Amini and Jalili &Ehsan et al.	enforcement performance	case study	Prolog, GWT, Jena, JIP, Protege
Bao et al.	-	-	-
Bonatti and Olmedilla	explanation performance	demo	Java, TuProlog
Chen and Stuckenschmidt			Jena
Costabello et al.	enforcement performance BSBM & BTC datasets	mobile use case	Java, Corese-KGRAM
Dietzold and Auer	-	-	RDF, SPARQL
Flouris et al.	annotation performance	-	Java, Jena, Sesame, Progress
Franzoni et al.	-	-	Java, SeRQL, Sesame
Gabillon and Letouzey	enforcement performance	-	Java, Tomcat, Sesame
Jain and Farkas	-	-	Java, Jena, Jess
Javanmardi et al.	policy reasoning	-	PELLET, SWRL
Kagal and Finin	-	use cases	Java, Prolog
Kim et al.	-	-	-
Kolovski et al.	policy reasoning Continue dataset	-	Pellet
Li and Cheung	-	-	-
Muhleisen et al.	enforcement performance BSBM dataset	demo	Joseki, Jena, Pellet
Oulmakhzoune et al.	- discussion	-	-
Papakonstantinou et al.	enforcement reasoning performance	-	PostgreSQL, MonetDB
Qin and Atluri	-	-	-
Reddivari et al.	query performance	-	Java, Jena, RDQL
Ryutov et al.	-	-	Java
Sacco and Breslin	enforcement performance	mobile use case	Java
Toninelli et al.	-	-	-
Uszok et al.	-	use cases	Java

seldom do researchers evaluate these claims. *Table* 3.4 provides an overview of the technologies adopted and indicates the evaluations performed.

Effectiveness (Ryutov et al., 2009). In order to work in practice, access control enforcement and administration needs to be efficient. A number of authors have presented performance evaluations of their access control enforcement (Amini and Jalili, 2010; Costabello et al., 2012b; Gabillon and Letouzey, 2010; Muhleisen et al., 2010; Reddivari et al., 2005; Sacco and Passant, 2011b), query rewriting (Abel et al., 2007), annotation (Flouris et al., 2010) , explanation (Bonatti and Olmedilla, 2007) or reasoning (Javanmardi et al., 2006a; Kolovski et al., 2007; Papakonstantinou et al., 2012) algorithms. However, there is still no clear access control benchmark, that can be used to compare different approaches. One suggestion would be to build a set of access control scenarios and extend the BSBM dataset generator to cater for solution benchmarking.

Distributed (Amini and Jalili, 2010). In order to ensure scalability, it should be possible to cater for the distributed specification and enforcement of access control policies. A number of researchers have examined how their proposed solution can be applied to use cases requiring distributed access control mechanisms. Amini and Jalili (2010) describe a case study on distributed semantic digital library. Bonatti and Olmedilla (2007) and Muhleisen et al. (2010) developed demos in order to demonstrate how their policy languages can be used in a distributed setting. Whereas, the access control languages and enforcement frameworks proposed by Costabello et al. (2012b); Kagal and Finin (2003); Sacco and Passant (2011b) and Uszok et al. (2003b) are motivated by distributed uses cases. However, the adaption of current distributed query processing techniques to cater for access control over Linked Data has not be explored to date.

Flexibility & Extensibility (Yagüe et al., 2003; Damiani et al., 2005; Bonatti and Olmedilla, 2007). The system should be capable of handling frequent changes to policies, user, access rights and resources. In addition, in order to provide support for different scenarios and future enhancements, the enforcement frameworks should be flexible and extensible. As each of the access control strategies we examined uses one or more open standards (*see Table* 3.1), they are by design flexible and extensible. An overview of the technologies used in each of the access control proposals we have examined, is presented in *Table* 3.4.

3.5 Summary

In this chapter, we provided an overview of relevant access control models (MAC, DAC, RBAC, VBAC, ABAC, CBAC) and standardisation efforts (XACML, WebID, WAC, P3P, APPEL), and described how they have been either enhanced by/applied to RDF. We examined a number of well known policy languages in detail, focusing in particular on ontology based, rule based and combined ontology and rule based access control enforcement. We subsequently examined the different strategies that have be used to specify access control over RDF (triple patterns, views, named graphs and ontologies). In addition, we discussed the interplay between access control, RDF and reasoning and described how query rewriting can be used to limit access to restricted data. Finally, we derived a set of requirements for Linked Data, based on several papers that examine access control for RDF from a number of perspectives. We subsequently used these requirements to classify existing strategies for access control over RDF. Based on this analysis we have identified a number of gaps with respect to access control for Linked Data, which is address in this thesis:

- When it comes to access control specification, enforcement and administration, to date much of the focus has been on using semantic technology to specify access control policies or proposing policies languages, that can be used to specify and reason over access control for RDF resources. Although considerable research has been conducted into the exposure of relational data as RDF, using Relational Database to RDF (RDB2RDF) (Arenas et al., 2012) and Relational Database to RDF Mapping Language (R2RML) (Das et al., 2012), none of the authors investigate how access rights, that were placed on the original relational data, can be lifted and enforced over the corresponding RDF data. In *Chapter* 4 we will demonstrate how RDB2RDF technology can be used not only to extract data from relational databases but also to extract permissions. Both the data and the permissions are represented using an extension of the RDF data model that allows contextual data to be associated with each triple, known formally as Annotated RDF.

- Although the delegation of access rights to others is a core access control administration requirement, to date it has gained very little traction within the Semantic Web community. As DAC allows users to delegate their permissions to others, it is particularly suitable for managing access control over distributed data. *Chapter* 5 examines how the graph data model differs from the relational and the tree data models, discusses how Discretionary Access Control (DAC) can be applied to graph-based data and describes the implication such structural differences have on access control in general.
- When it comes to access control administration and enforcement, a number of different categories of rules are necessary: to grant/deny access to data at multiple level of granularity; to support reasoning over access control policies; to ensure both the consistency and safety of the access control policies; and also to support conflict resolution. To date, a number of different rules have been proposed in each of these categories. Given the diversity of the existing proposals, and the fact that when it comes to access control *no one size fits all*, there is a need for a general rules framework, which can be used to cater for the specification, administration and enforcements of access control policies over Linked Data. In *Chapter* 5 we demonstrate how the hierarchical Flexible Authorisation Framework, proposed by Jajodia et al. (2001), can be extended to provide DAC over graph data.
- Several authors propose access control strategies that can be used in conjunction with RDF query languages, to return partial query results by filtering out unauthorised data. One of the limitations of existing proposals is that they do not specifically consider complex SPARQL 1.1 queries (such as, aggregates, negation, subqueries and property path) and SPARQL 1.1 update queries. As such, there is a need for an access control mechanism which can be used to rewrite such complex queries so that they behave as if the unauthorised data is not present in the dataset. *Chapter* 6 investigates the security implications associated with granting partial access to RDF data, via SPARQL 1.1 query rewriting.
- In order to work in practice, both access control enforcement and administration need to be effective from both a performances and a correctness perspective. Although a number of authors have conducted access control performances evaluations using the BSBM dataset, when it comes to access control for RDF data there is currently no general access control benchmark. In addition, there is a pressing need for a general mechanism, which can be used to verify the correctness of proposed access control mechanisms. In *Chapter* 6 we propose a benchmark, based on the BSBM dataset, that can be used to compare different strategies for access control for Linked Data. In addition, we demonstrate how a set of correctness criteria, which was originally used by Wang et al. (2007) to verify the correctness of relational access control policies, can be adapted to ensure the correctness of access control over RDF.
- Given that RDF data can be served as RDF documents; embedded in HTML documents; exposed via SPARQL endpoints; or translated from relational data; there is an need for an authorisation architecture, which can be used to provide access control for Linked Data irrespective of how the data is represented or consumed. In *Chapter* 6, we discuss how together our Flexible Authorisation Framework and our Query Rewriting algorithms can be used to enforce access control over Linked Data.

Chapter 4

Using Annotated RDFS for Access Control over Integrated RDF Data

Enterprises rely on stand-alone systems, commonly known as Line Of Business (LOB) applications, to perform day-to-day activities efficiently. For example, interactions with clients are recorded in a Customer Relationship Management (CRM) application, employee information is maintained in a Human Resources (HR) application and project documentation is stored in a Document Management System (DMS). These systems, although independent, often contain different information regarding the same entities. For example, if an organisation needs to know the projects commissioned by a customer, the employees that worked on those projects and the revenue that was generated, they need to obtain information across these systems. However, such integration is not a simple task, not only due to the heterogeneity of the systems, but also due to the presence of access control mechanisms in each system. In fact, since much of the information within the enterprise is highly sensitive, this integration step could result in information leakage to unauthorised individuals. RDF is a flexible format which can be used to represent integrated data, however it does not provide any mechanisms to safeguard against information leakage.

In this chapter, we demonstrate how RDB2RDF technology, which can be used to extract information from relational databases into RDF, can be used to extract access control information from LOB applications. We define a mechanism to enforce access control over the resulting RDF graph, which we implement via logic programming. The solution we present builds upon an extension of the RDF data model (called Annotated RDF), which supplements the RDF model with context information, while providing backwards compatibility. Our approach provides a representation for the access control policies at a triple level and also caters for permission propagation, via logic inference rules. In order to cater for situations where information is represented directly as RDF we examine the suitability of the proposed access control mechanism for the specification of new access control policies. For this task we adopt a bottom up approach, examining the access control requirements based on existing software engineering and database access control models.

In relation to access control for integrated RDF data, we make the following contributions. We: (i) define an annotation domain that models access control permissions in RDF; (ii) discuss how RDB2RDF technology can be used to extract both data and permissions from relational databases; (iii) propose a set of rules that are necessary for the enforcement of existing access control models over RDF; and (iv) present a data integration and access control enforcement framework for RDF. In addition, we detail our implementation of the proposed enforcement framework and examine the overall performance of our prototype over real enterprise data.

The remainder of the chapter is structured as follows: in *Section* 4.1 we provide the necessary background information on the annotated RDF (aRDF) model. *Section* 4.2 formalises the access control annotation domain and details our implementation of the domain in logic programming. *Section* 4.3 examines the predominant software engineering and database access control models and details the rules necessary to propagate access control policies over RDF data, based on these access control models. The framework and our specific implementation are described and evaluated in *Section* 4.4. Finally, we discuss the related work in *Section* 4.5 and we summarise and present future directions in *Section* 4.6.

4.1 aRDFS Background

In this section we provide the necessary background information regarding the semantics of Annotated RDFS. We start by presenting the data model, giving an overview of RDF and its extension towards Annotated RDFS which draws inspiration from Annotated Logic Programming (Kifer and Subrahmanian, 1992). We then present the extension of the RDF Schema (RDFS) inference rules for the annotated case and the extension of the SPARQL query language for querying Annotated RDFS, AnQL. Finally, we present the current Annotated RDFS and AnQL prototype which is implemented in SWI-Prolog.

4.1.1 The aRDFS Data Model

An RDF triple has the intuitive meaning that the *subject* is connected to the *object* by the *predicate* relation (Definition 2.1). Several extensions were proposed to introduce meta-information into the RDF data model. For example, Gutierrez et al. (2007) define temporal RDF, which allows for the allocation of a validity interval to an RDF triple. Whereas, Straccia (2009) presents fuzzy RDF in order to attach a confidence or membership value to a triple. These and other approaches can be represented within a common framework, called Annotated RDF (Udrea et al., 2010) and further extended to include RDFS inference rules by Zimmermann et al. (2012). Annotated RDFS introduces the notion of an *annotation domain* into the RDF model and defines an extension of the RDFS inference rules that, by relying on the $\otimes$ and $\oplus$ operations defined by the annotation domain (Definition 4.1), can be specified in a *domain independent* fashion.

Definition 4.1 (Annotation domain)
Let L be a non-empty set, whose elements are considered the annotation values. *We say that an* annotation domain *for RDFS is an idempotent, commutative semi-ring $D = \langle L, \oplus, \otimes, \bot, \top \rangle$, where $\oplus$ is $\top$-annihilating. That is, for $\lambda, \lambda_1, \lambda_2, \lambda_3 \in L$: $\oplus$ is idempotent, commutative, associative; $\otimes$ is commutative and associative; $\bot \oplus \lambda = \lambda$, $\top \otimes \lambda = \lambda$, $\bot \otimes \lambda = \bot$, and $\top \oplus \lambda = \top$; $\otimes$ is distributive over $\oplus$, i.e. $\lambda_1 \otimes (\lambda_2 \oplus \lambda_3) = (\lambda_1 \otimes \lambda_2) \oplus (\lambda_1 \otimes \lambda_3)$; An annotation domain $D = \langle L, \oplus, \otimes, \bot, \top \rangle$ induces a partial order $\preceq$ over L defined as: $\lambda_1 \preceq \lambda_2$ iff $\lambda_1 \oplus \lambda_2 = \lambda_2$.*

Example 4.1 (Annotation domain)
The Fuzzy Annotation domain is defined as
$D_{[0,1]} = \langle [0, 1], \max, \min, 0, 1 \rangle$. Using this domain we can specify that `:joeBloggs` *is a part-time employee of* `:westportCars` *as follows:*

$$(\texttt{:joeBloggs}, \texttt{:worksFor}, \texttt{:westportCars}) : 0.5$$

For the definitions of other domains, such as the temporal and provenance domains, the reader is referred to Zimmermann et al. (2012). Further to the above annotation domain definition, we extend RDF towards annotated RDFS:

Definition 4.2 (Annotated triple, annotated graph)
An annotated triple *is an expression $\tau\colon \lambda$, where τ is an RDF triple and λ is an* annotation value*. An* annotated RDFS graph *is a finite set of annotated triples.*

The entailment between two Annotated RDFS graphs, $G \models H$ is defined by a model-theoretic semantics presented in Zimmermann et al. (2012).

4.1.2 Inference Rules

RDF Schema (RDFS) consists of a predefined vocabulary that assigns specific meaning to certain IRIs, allowing a reasoner to infer new triples from existing ones. A set of inference rules can be used to provide a sound and complete reasoner for RDFS (Ter Horst, 2005). These rules can be extended to support Annotated RDFS reasoning, in a domain-independent fashion, simply by relying on the $\otimes$ and $\oplus$ operations (presented in Definition 4.1). Such rules can be represented by the following meta-rules:

Rule 4.1 ($\otimes$ Domain operator)
If a classical RDFS triple τ can be inferred by applying an RDFS inference rule to triples $\tau_1, \ldots \tau_n$ (denoted $\{\tau_1, \ldots, \tau_n\} \vdash_{\mathsf{RDFS}} \tau$), the same triple can be inferred in the annotated case with annotation term $\bigotimes_i \lambda_i$, where λ_i is the annotation of triple τ_i.

$$\frac{\tau_1\colon \lambda_1,\ \ldots,\ \tau_n\colon \lambda_n, \{\tau_1, \ldots \tau_n\} \vdash_{\mathsf{RDFS}} \tau}{\tau\colon \bigotimes_i \lambda_i}\ .$$

Rule 4.2 ($\oplus$ Domain operator)
The $\oplus$ operation is in turn used to combine information about the same statement. If the same triple is inferred from different rules or steps in the inference, the following rule is applied:

$$\frac{\tau\colon \lambda_1,\ \tau\colon \lambda_2}{\tau\colon \lambda_1 \oplus \lambda_2}\ .$$

It is also possible to specify a custom set of rules in order to provide application specific inferencing. A number of custom rules for managing permissions in the access control domain are presented in *Section* 4.3.

4.1.3 AnQL: Annotated Query Language

The proposed query language for Annotated RDFS, AnQL (Lopes et al., 2010), is an extension of SPARQL, the W3C recommended query language for RDF, which also takes into consideration features from the SPARQL 1.1 language revision. In *Definition* 2.1, ***I, B*** and ***L***, are used to represent ***IRIs***, ***blank nodes*** and ***literals*** respectively. Consider $\mathbf{V}$ a set of variables disjoint from $(\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$. In SPARQL, a *triple pattern* consists of an RDF triple with optionally a variable $v \in \mathbf{V}$ as the subject, predicate and/or object. Sets of triple patterns are called *basic graph patterns* (BGP) and BGPs can be combined to create generic *graph patterns*. The semantics of SPARQL is based on the notion of *basic graph pattern* matching, where a *substitution* is a partial function $\mu\colon \mathbf{V} \to (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$.

For the extension of SPARQL towards the AnQL query language, we propose a specific annotation domain instance of D of the form $\langle L, \oplus, \otimes, \bot, \top \rangle$. Let $\mathbf{A}$ denote the set annotation variables, disjoint from $(\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$ and λ be an annotation value from $\mathbf{L}$ or an annotation variable from $\mathbf{A}$, called an *annotation label.* For a SPARQL triple pattern τ, we call $\tau\colon \lambda$ an *annotated triple pattern* and sets of annotated triple patterns are called *basic annotated patterns* (BAP). Similar to SPARQL, BAPs can be combined to create an *annotated graph pattern* and for further details we refer the reader to (Lopes et al., 2010).

Definition 4.3 (AnQL query)
An AnQL query *is defined as a triple $Q = (P, G, V)$ where: (i) P is an annotated graph pattern; (ii) G is an annotated RDF graph; and (iii) $V \subseteq \mathbf{VA}$ is the set of variables to be returned by the query. Given an annotated graph pattern P, we further denote by $var(P) \subseteq \mathbf{V}$ and $avar(P) \subseteq \mathbf{A}$ the set of variables and annotation variables respectively present in a graph pattern P.*

As presented in Example 4.2, the annotated graph pattern P is specified following the `WHERE` keyword, while the variables are specified after the `SELECT` keyword.

Example 4.2 (AnQL query)
Considering the fuzzy domain presented in Example 4.1, we can pose the following query:

```
SELECT ?v ?av WHERE { ?v a :Company ?av }
```

where `?v` *is a variable from* $\mathbf{V}$ *and* `?av` *is an annotation variable from* $\mathbf{A}$*.*

The semantics of AnQL BAP matching is defined by extending the notion of SPARQL *basic graph pattern* matching to cater for annotation variables and their mapping to annotation values. For any substitution μ and variable v, $\mu(v)$ corresponds to the value assigned to v by μ. For a BAP P, $\mu(P)$ represents the annotated triples that correspond to P except that any variable $v \in vars(P) \cup\ avars(P)$ is replaced with $\mu(v)$.

Definition 4.4 (BAP matching)
Let P be a BAP and G an Annotated RDFS graph. We define the evaluation *of P over G, denoted $[\![P]\!]_G$, as the list of substitutions that are* solutions *of P, i.e. $[\![P]\!]_G = \{\mu \mid G \models \mu(P)\}$, according to the model-theoretic definition of entailment presented by Zimmermann et al. (2012).*

The semantics of arbitrary annotated graph patterns is defined by an algebra that is built on top of this *BAP matching.* For further details we refer the reader to (Lopes et al., 2010). A combined overview of Annotated RDFS and AnQL is provided by Zimmermann et al. (2012).

4.1.4 Implementation

The system architecture of our prototype implementation, based on SWI-Prolog's Semantic Web library (Wielemaker et al., 2008), is sketched in *Figure* 4.1. The main component of the system consists of the *Reasoner / AnQL Query Engine*, which is composed of a forward-chaining reasoner engine with a fix-point semantics that calculates the closure of a given *Annotated RDF Graph* (Zimmermann et al., 2012) and an implementation of the AnQL query language. This main component can be tailored to a specific *Annotation Domain* and to include different *Inference Rules* describing how triples and their annotation values are propagated. Such inference rules can be specified, in domain independent fashion, by using a high-level language that ab-

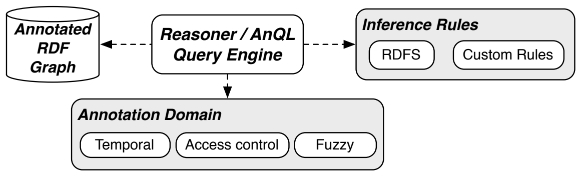


Figure 4.1: Annotated RDFS implementation schema

stracts the specific details of each domain. An example of an Annotated RDFS rule is presented below.

Example 4.3 (Annotated RDFS inference rule)
The following rule provides subclass inference in the RDFS ruleset:

```
rdf(O, rdf:type, C2, V) <==  rdf(O, rdf:type, C1, V1),
                             rdf(C1, rdfs:subClassOf, C2, V2),
                             infimum(V1, V2, V).
```

where the `rdf/4` predicate is used to represent the annotated triples and the `infimum/3` predicate corresponds to the implementation of the $\otimes$ domain operation (Definition 4.1).

More information and downloads of the prototype implementation can be found at `http://anql.deri.org/`.

4.2 Access Control Annotation Domain

Following the definitions presented in *Section* 4.1 , in this section we formalise our access control annotation domain. We start by defining the entities and annotation values and then present the $\otimes$ and $\oplus$ domain operations. Finally, we describe the implementation of the presented access control annotation domain.

4.2.1 Entities and Annotations

For the modelling of the Access Control Domain (ACD) consider, in addition to the previously presented sets of IRIs $\mathbf{I}$, blank nodes $\mathbf{B}$, and literals $\mathbf{L}$, a set of credential elements $\mathbf{C}$. The elements of $\mathbf{C}$ are used to denote *usernames*, *roles*, and *groups*. To denote *attributes*, we propose a set $\mathbf{T}$ of pairs of form (k, v), represented as key–value pairs where $k \in \mathbf{U}$ and $v \in \mathbf{L}$. For example, "$(:age, 30)$" or "$(:institute, DERI)$" are elements of $\mathbf{T}$.[1] We allow shortcuts to represent intervals of integers, for example, "$(:age, [25, 30])$" is used to indicate that all entities with attribute "$:age$" between 25 and 30 are allowed access to the triple.

Considering an element $e \in (\mathbf{C} \cup \mathbf{T})$, e and $\neg e$ are *access control elements*, where e is called a *positive* element and $\neg e$ is called a *negative* element. Here we are using $\neg e$ to represent strong negation. In our access control domain representation, $\neg e$ indicates that e will be specifically *denied* access. An *access control statement* S consists of a set of access control elements and an *Access Control List* (ACL) consists of a set of access control statements. An access control statement S is *consistent* if and only if, for any element $e \in (\mathbf{C} \cup \mathbf{T})$, only one of e and $\neg e$ may appear in S. This restriction avoids *conflicts*, where a statement is attempting to both *grant* and *deny* access to a triple. Furthermore, we can define a partial order between access control

[1]In these examples, the default URI prefix is `http://urq.deri.org/enterprise#`.

statements S_1 and S_2, as $S_1 \leq S_2$ *iff* $S_1 \subseteq S_2$. This partial order can be used to eliminate *redundant* access statements within an ACL. If a user is granted access by statement S_2, he will also be granted access by statement S_1 (and thus S_2 can be removed). Finally, an ACL is *consistent* if and only if all statements therein are consistent and not redundant. In our domain representation, only consistent ACLs are considered as annotation values. Intuitively, an annotation value specifies which entities have read permission to the triple, or are denied access when the annotation is preceded by $\neg$.

Example 4.4 (Access control list)
Assume a set of entities $\mathbf{C} = \{jb, js, hr, it\}$, where jb and js are employee usernames and hr and it are shorthand for humanResources and informationTechnology, respectively. The following annotated triple:

$$\tau : [[it], [hr, \neg js]]$$

states that the entities identified with it or hr (except if the js credential is also present) have read access to the triple τ.

An ACL A can be considered as a non-recursive Datalog with negation (nr-datalog$^{\neg}$) program, where each of the access control statements $S \in A$ corresponds to the body of a rule in the Datalog program. The head of each Datalog rule is a reserved element $access \notin (\mathbf{C} \cup \mathbf{T})$ and the evaluation of the Datalog program determines the access permission to a triple given a specific set of credentials. The set of user credentials is assumed to be provided by an external authentication service and consists of elements of $(\mathbf{C} \cup \mathbf{T})$ which equates to a non-empty ACL representing the entities associated with the user. As expected, we assume that this ACL consists of only one positive statement, i.e. the ACL will contain one statement with all the entities associated with the user and does not contain any negative elements.

Example 4.5 (Datalog representation of an ACL)
Taking into account the annotation example presented above. The nr-datalog$^{\neg}$ program corresponding to the ACL $[[it], [hr, \neg js]]$ is:

$$\begin{array}{l} access \leftarrow it. \\ access \leftarrow hr, \neg js. \end{array}$$

The set of credentials of the user session*, provided by the external authentication system eg. $[[js, it]]$, are facts in the nr-datalog$^{\neg}$ program.*

Further domain specific information, for example, the encoding of hierarchies between the credential elements, can be encoded as extra rules within the nr-datalog$^{\neg}$ program. These extra rules can be used to provide *implicit* credentials to a user, allowing the access control to be specified based on credentials that the authentication system does not necessarily assign to a user.

Example 4.6 (Credential hierarchies)
If the entity emp represents all the employees within a specific company, and that jb and js correspond to employee usernames (as presented in Example 4.4), the following rules can be added to the nr-datalog¬ program from Example 4.5:

$$emp \leftarrow js.$$
$$emp \leftarrow jb.$$

These rules ensure that both jb and js are given access when the credential emp is required in an annotation value.

Although in the above example we use these rules to express hierarchies between entities in reality any form of nr-datalog¬ rules are allowed. In *Section* 4.3, we demonstrate how rules can be used to provide support for a number of traditional access control models.

4.2.2 Operations

We now turn to the annotation domain operations $\otimes$ and $\oplus$, that as presented in *Section* 4.1.2, that allow for the combination of annotation values catering for RDFS inferences. A naive implementation of these domain operations may produce ACLs that are not consistent (and would not be considered valid annotation values). To avoid such invalid ACLs, we rely on a normalisation step that ensures the result is a valid annotation value, by checking for redundant statements and applying a conflict resolution policy if necessary. If an annotation statement contains a positive and negative access control element for the same entity, e.g $[jb, \neg jb]$, there is a *conflict*. There are two different ways to resolve conflicts in the annotation statements: (i) apply a *brave* conflict resolution strategy (allow access); or (ii) *safe* conflict resolution strategy (deny access). This is achieved during the normalisation step, through the *resolve* function, by removing the appropriate element: $\neg jb$ for brave conflict resolution or jb for safe conflict resolution. In our current modelling, we are assuming safe conflict resolution. The normalisation process is defined as follows:

Definition 4.5 (Normalise)
Let A be an ACL. We define the reduction of A into its consistent form, denoted norm(A), as:

$$normalise(A) = \{resolve(S_i) \mid S_i \in A \text{ and } \nexists S_j \in A, i \neq j \text{ such that } S_i \leq S_j\} \quad .$$

$\oplus_{ac}$ **Operator.** The $\oplus$ operation is used to combine annotations when the same triple is deduced from different inference steps (Rule 4.1). For the access control domain, the $\oplus_{ac}$ operation involves the union of the annotations and the subsequent normalisation operation. The result of this operation intuitively creates a new nr-datalog¬ program consisting of the union of all the rules from the original nr-datalog¬ programs. Formally the $\oplus$ operation is defined as follows:

Definition 4.6 ($\oplus_{ac}$ Operator)

$$A_1 \oplus_{ac} A_2 = normalise\,(A_1 \cup A_2) \quad .$$

The Following example demonstrates how the $\oplus_{ac}$ operation can be used to combine two ACLs:

Example 4.7 ($\oplus_{ac}$ Operator)
Consider the following triples:

$$\tau_1 = (\texttt{:johnSmith}, \texttt{:salary}, \texttt{40000})\colon\ [[js]].$$

$$\tau_2 = (\texttt{:johnSmith}, \texttt{:salary}, \texttt{40000})\colon\ [[hr]].$$

Combining these triples with the $\oplus_{ac}$ operation (by applying Rule *4.1) would result in access being granted to all the entities that are allowed to access the premises:*

$$(\texttt{:johnSmith}, \texttt{:salary}, \texttt{40000})\colon\ [[js], [hr]] \quad .$$

$\otimes_{ac}$ **Operator.** The $\otimes$ operation is in turn used to derive new annotations when new triples are inferred (Rule 4.2). The $\otimes_{ac}$ operation involves the merge of both annotations, normalisation and conflict resolution. This equates to restricting access to inferred statements to only those entities that have access to the both the original statements. Thus, the $\otimes$ operation corresponds to:

Definition 4.7 ($\otimes_{ac}$ Operation)

$$A_1 \otimes_{ac} A_2 = normalise\,(\{S_1 \cup S_2 \mid S_1 \in A_1 \text{ and } S_2 \in A_2\}) \quad ,$$

where $S_1 \cup S_2$ represents the set theoretical union.

Unlike the $\oplus_{ac}$ operation, the $\otimes_{ac}$ may produce conflicts in the annotation statements. For example, the application of the $\otimes_{ac}$ operation with the Annotated RDFS dom rule is as follows:

Example 4.8 ($\otimes_{ac}$ Operation)
Given the following triples:

$$\tau_1 = (\texttt{:westportCars}, \texttt{:netIncome}, \texttt{1000000})\colon\ [[hr, \neg jb]].$$

$$\tau_2 = (\texttt{:netIncome}, \texttt{rdfs:domain}, \texttt{:Company})\colon\ [[it, jb]].$$

The annotation resulting from applying the $\otimes_{ac}$ operation should provide access to the resulting triple only to entities which are allowed to access all the premises. Thus we can infer, not only that :westportCars *is of type* :Company*, but also the appropriate annotation value:*

$$(\texttt{:westportCars}, \texttt{rdf:type}, \texttt{:Company})\colon\ [[hr, it, \neg jb]] \quad .$$

Please note that the aforementioned conflict resolution mechanism simplifies $[\neg jb, jb]$ to $[\neg jb]$.

Top and Bottom Annotations. The smallest and largest annotation values in the access control domain, $\bot_{ac}$ and $\top_{ac}$ respectively, correspond in turn to an empty nr-datalog$^{\neg}$ program and another that provides access to all entities $e \in (\mathbf{C} \cup \mathbf{T})$: $\bot_{ac} = []$ and $\top_{ac} = [[]]$. The $\bot_{ac}$ annotation value element indicates that the annotated triple is not accessible to any entity, since no annotation statements will provide access to the triple, and an annotation value of $\top_{ac}$ states that the triple is *public*, since any credential contained in the user session will trivially provide access to the triple. Intuitively, the $\top_{ac}$ annotation is translated into the nr-datalog$^{\neg}$ program containing only the "access" fact, while $\bot_{ac}$ corresponds to an empty program. However, for practical reasons, it might be necessary to assume a "super-user" role, for instance, represented as the reserved element "su", which will be allowed access to all triples and therefore would be used as the $\bot_{ac}$ annotation.

Access Control Annotation Domain. The access control annotation domain is formally defined as follows:

Definition 4.8 (Access control annotation domain)
Let $\mathbf{F}$ *be the set of annotation values over* $(\mathbf{C} \cup \mathbf{T})$*, i.e. consistent ACLs.*
$D_{ac} = \langle \mathbf{F}, \oplus_{ac}, \otimes_{ac}, \bot_{ac}, \top_{ac} \rangle$.

The presented access control domain modelling can be easily extended to handle other permissions, like *update*, and *delete* by representing the annotation as an n-tuple of ACL $\langle P, Q, \ldots \rangle$, where P specifies the formula for *read* permission, Q for *update* permission, etc. In this extended domain modelling, the domain operations can also be extended to operate over the corresponding elements of the annotation tuple. A *create* permission has a different behaviour as it can not be attached to any specific triple as it is a graph-wide permission and thus is out of scope for this modelling. Support for graph based access control is presented in *Chapter* 5.

4.2.3 Prolog Implementation

Given the annotated RDFS implementation described in *Section* 4.1.4, the access control annotation domain implementation consists of a Prolog module that is imported by the reasoner. This module defines the domain operations $\otimes_{ac}$ and $\oplus_{ac}$, represented as the predicates `infimum/3` and `supremum/3` respectively. Although it is possible to represent the annotations using reification, which requires four statements to represent the triple (one to assign an identifier to a statement and three additional statements to associate a subject, predicate and object with the identifier) and an additional statement to associate an annotation with the identifier, such an approach would increase the size of the database making both querying and reasoning more expensive from a performance perspective. An alternative approach is to represent annotation values using *lists* (in this case lists of lists), following the notions presented in the previous section.

The implementation of the $\oplus_{ac}$ operation involves concatenating the list representation of both annotations and then performing the normalisation operation. As for the $\otimes_{ac}$ operation, we follow a similar procedure to the $\oplus_{ac}$ operation, with the additional step of applying either the *brave* or the *safe* conflict resolution method. The evaluation of the nr-datalog$^{\neg}$ program can be performed based on the representation of the annotation values, by checking if the list of credentials of a user is a superset of any of the positive literals of the statements of our annotation values and also that it does not contain any of the negative literals of the statement.

An example of RDF data annotated with access control information is presented in *Figure* 4.2, where the salary information is only available to the respective employee. In this figure we are

```
@prefix : <http://urq.deri.org/enterprise#> .
:westportCars rdf:type :Company  "[[jb]]".
:westportCars :netIncome 1000000 .
:joeBloggs :worksFor :westportCars .
:joeBloggs :salary 80000 "[[jb]]".
:johnSmith :worksFor :westportCars .
:johnSmith :salary 40000 "[[js]].
```

Figure 4.2: RDF triples annotated with access control permissions

representing the RDF triples and annotation element using the nquads RDF serialisation.[2] Using AnQL, the extension of the SPARQL query language described in *Section* 4.1.3, it is possible to perform queries that take into consideration the access control annotations. An example of an AnQL query over this data is presented in the following example:

Example 4.9 (AnQL query)
This query specifies that we are interested in the salary of employees that someone with the permissions $[[jb, hr, it]]$ *is allowed to access.*

```
SELECT * WHERE { ?p :salary ?s "[[jb, hr, it]]" }
```

The answers for this query (when matched against the data from Figure *4.2) under SPARQL semantics, i.e. if the annotation was omitted, would be:*

$$\{\{\texttt{?p} \to \texttt{:joeBloggs}, \texttt{?s} \to 80000\}, \{\texttt{?p} \to \texttt{:johnSmith}, \texttt{?s} \to 40000\}\} \quad .$$

However, when the domain annotations are present, an AnQL query engine must also perform the following check: $[[jb, hr, it]]$ *satisfies the nr-datalog*$^{\neg}$ *program* λ*, where* λ *is the program represented by the annotation of each matched triple, thus yielding only the following answer:*

$$\{\{\texttt{?p} \to \texttt{:joeBloggs}, \texttt{?s} \to 80000\}\} \quad .$$

4.3 Representing Traditional Access Control Models

To date Semantic Web researchers have adopted a top-down approach to RDF access control specification, modelling access control based on RDF data structures. Our approach is rather to extract, model and enforce existing access control mechanisms over RDF data. Therefore, we adopt a bottom up approach examining the access control requirements of existing software systems. In *Section* 4.3.1 we provide an overview of access control terminology and in *Section* 4.3.2 we describe several prominent access control models. In addition, we examine how existing access control models can be used to protect RDF data. We propose a set of access control rules that are required in order to propagate and enforce access control policies, based on these models, over RDF data.

4.3.1 Access Control Terminology

An *Access Control Model* provides guidelines on how access to system resources and data should be restricted. Whereas an *Access Control Policy* (ACP) details the actual authorisations and

[2]http://sw.deri.org/2008/07/n-quads/

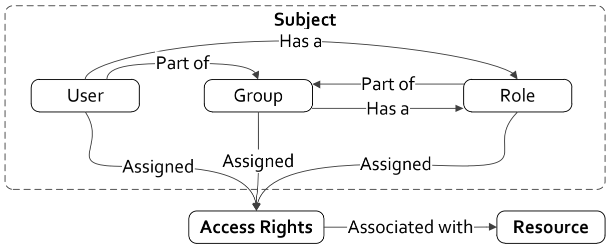


Figure 4.3: Components of an access control statement

Table 4.1: Categorisation of access control models

Enterprise	*Data Models*	*Distributed Systems*
MAC	VBAC	ABAC
DAC	OBAC	
RBAC		

access restrictions to be enforced. Such permissions and prohibitions are specified as individual *Access Control Statements* (ACS), represented as a tuple: $\langle S, R, AR\rangle$ where S denotes the `subject`, R denotes the `resource` and AR represents the `access rights`. *Figure* 4.3 depicts the relationship between the access control terms that are used in this chapter. A `Resource` is used to denote the information to be protected (for example, database records, application objects, website IRIs). *Users* represent individuals requesting access to resources. We note that such individuals may or may not be known in advance of the information request. *Groups* are collections of users or roles with common features (for example, contributors, supervisors and management). *Roles* are used to assign a set of access rights to a set of individuals and groups, such as by department (for example, human resources, sales and marketing) or by task (for example, insurance claim processing, reporting and invoicing). The term `Subject` is an umbrella term used to collectively refer to users, roles and groups. `Access rights` are generally defined as permissions and prohibitions applied to resources and granted to subjects.

4.3.2 Traditional Access Control Models

A high level categorisation of the models discussed in this section is presented in *Table* 4.1. In keeping with our enterprise data integration use case, we focus on access control models commonly used in enterprises; models applicable to other data representation formats; and those that are relevant for distributed systems. An overview of the models is presented below. As the models will be discussed in the next section we label them M_x where x is the name of the access control model.

M_{MAC}**: Mandatory Access Control (MAC) (Samarati and de Vimercati, 2001).** In this model access to resources is restricted through mandated policies determined by a central authority (for example, mainframes and lightweight directory services).

M_{DAC}**: Discretionary Access Control (DAC) (Samarati and de Vimercati, 2001).** Similar to MAC, access to resources is constrained by a central access control policy, however in contrast to MAC users are allowed to override the central policy (for example, employees may be allowed to grant others access to their own resources).

M_{RBAC}: **Role Based Access Control (RBAC) (Sandhu, 1998).** Generally speaking, RBAC involves grouping a set of access rights that describe the responsibilities or tasks that can be performed by a role (for example, manager, sales person and clerk). RBAC is the most commonly used access control model in enterprise applications.

M_{VBAC}: **View-based Access Control (VBAC) (Griffiths and Wade, 1976).** In relational database systems VBAC is used to simultaneously grant access to one or more tables, tuples or fields. A similar approach is used in Object Oriented Access Control (OBAC) (Evered and Bögeholz, 2004) where access rights are granted to application objects.

M_{ABAC}: **Attribute Based Access Control (ABAC) (McCollum et al., 1990).** In distributed environments where the requester is unknown prior to the submission of the request, an alternative means of granting access is required. In ABAC (McCollum et al., 1990), access control decisions are based on attributes (for example, *employer=storm*, *policyNumber=565656*), in the form of digitally signed documents that are sent by the requester to the server.

4.3.3 Support for Existing Access Control Models

In this section we discuss how each of the access control models, presented in *Section* 4.3.2 can be supported.

M_{MAC} **and** M_{DAC}. Given both models relate to the entire ACP as opposed to an ACS, a permission management module is necessary to enable system administrators to specify standard access control policies. In the case of DAC users are allowed to manage access control policies and to delegate their permissions to others. Whereas in the case of MAC all policies are set by a central authority. Given the former is more suitable for distributed environments, such as the web, in *Chapter* 5 we examine however DAC is applied to the relational and the XML data models and discuss how it can be used in conjunction with the RDF data model.

M_{RBAC}. RBAC can be represented using a single value element notation. The following quad demonstrates how a single annotation can be used to simultaneously cater for users, roles and groups.

```
:WestCars1 a :Project "([[:mary, :manager, :salesDept ]])"
```

M_{ABAC}. A key–value pair element representation is needed to support attributes. The annotation elements outlined below are used to demonstrate how a user's current employer can be represented using key–value pairs.

```
:WestCars1 a :Project "([[(:employer,:storm)]])"
```

M_{VBAC}. In RDB2RDF it is common to transform each database record into several triples that share a common subject. In our data integration scenario M_{VBAC} can be catered for by using RDB2RDF to apply the access control directly to each triple generated during the translation. For example, the following two triples could be generated from two separate columns of a database table:

```
:WestCars1 a :Project "([[:employee]])".
:WestCars1 :Client :WestCarsLtd "([[:employee]])".
```

4.3.4 Simplifying Administration using Rules

Table 4.2 provides an overview of the rules required by each of the models. The table also includes a column labelled *CRUD*. Although access rights are not a model per se, we have included them

Table 4.2: Overview of access control rules

	M_{MAC}	M_{DAC}	M_{RBAC}	M_{VBAC}	M_{ABAC}	*CRUD*
Rule 4.3				$\surd$		
Rule 4.4			$\surd$			
Rule 4.5	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	
Rule 4.6	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	
Rule 4.7						$\surd$

here as a specific rule is required to infer access rights based on permission hierarchies.

These rules can be used to associate permissions with data, that may or may not be extracted from existing systems. Propagation chains can be broken by explicitly specifying permissions on a particular triple. However, the rules may need to be extended to consider provenance as it may not be desirable to propagate permissions to related data from different sources.

In each of the rules, we represent variables with the ? prefix. In general $?S$, $?P$, and $?O$ refer to data variables while $?\lambda$ and $?E$ refer to annotation variables and annotation element variables respectively. The premises (represented above the line) correspond to a conjunction of quads, possibly with variables for any position. Whereas the conclusion (represented below the line) corresponds to a single quad where the variables are instantiated from the premises. We also assume the premises may include functions, for example, the *member* function is true, if and only if, an access control element is included in a provided access control list. The $\oplus_{ac}$ operation is used to combine the access control information associated with two triples. This operation is used to combine complete lists as well as to combine single access control elements with access control lists, which intuitively adds the new element to the list.

Resource Based Access. The most basic access control rule involves granting a subject access rights to a resource. Normally the access is explicit and therefore no inference is required. However in order to provide support for M_{VBAC} we need the ability to associate access rights with several triples. As such, we need a rule to propagate access rights to all triples with the same RDF subject. Given the following triples:

```
:Invoice1 a :Document "([[:john]])"
:Invoice1 :located "/dms/projs/docs"
```

we can infer that:

```
:Invoice1 a :Document "([[:john]])"
:Invoice1 :located "/dms/projs/docs" "([[:john]])"
```

Rule 4.3 (Propagation based on the subject)
Assuming that we have access rights associated with one triple. We can use this rule to propagate λ_1 access rights to all triples with the same subject.

$$\frac{?S\ ?P_1\ ?O_1\ ?\lambda_1,\quad ?S\ ?P_2\ ?O_2\ ?\lambda_2}{?S\ ?P_2\ ?O_2\ (?\lambda_2 \oplus_{ac} ?\lambda_1)}$$

Hierarchical Subjects Inheritance. In M_{RBAC} access control subjects are organised hierarchically and lower-level subjects inherit the access rights of higher-level subjects. For example, in a role hierarchy access rights allocated to the `manager` role will be inherited by all individuals in the organisation that have been granted the `manager` role.

Given the following triples:

```
:Invoice1 a :Document "([[:manager]])"
:john :inheritsFrom :manager
```

we can infer that:

```
:Invoice1 a :Document "([[:manager],[:john]])"
```

Rule 4.4 (Propagation based on subject hierarchies)
If $?E_1$ and $?E_2$ are access rights and $?E_2$ inherits from $?E_1$ then triples with access rights $?E_1$ should also have $?E_2$.

$$\frac{?S\ ?P\ ?O\ ?\lambda_1,\ ?E_2\ \texttt{:inheritsFrom}\ ?E_1, member(?E_1, ?\lambda_1)}{?S\ ?P\ ?O\ (?\lambda_1 \oplus_{ac} ?E_2)}$$

Hierarchical Subjects Subsumption. Like *Rule* 4.3, access control subjects can be organised to form a hierarchy. However in this instance we are talking about an organisation structure as opposed to a role hierarchy, in which case higher-level subjects will subsume the access rights of lower-level subjects. For example, managers implicitly gain access to resources that their subordinates have been explicitly granted access to. Given the following triples:

```
:Invoice2 a :Document "([[:john]])"
:mary :hasSubordinate :john
```

we can infer that:

```
:Invoice2 a :Document "([[:john], [:mary]])"
```

If an $?E_2$ has subordinate $?E_1$, then $?E_2$ will be granted access to any triples $?E_1$ has access to. Since this format of rule differs from *Rule* 4.3 only in the vocabulary that connects the resources, the same rule can be reused if we replace `:inheritsFrom` with `:hasSubordinate`.

Hierarchical Resources Inheritance. Similar principles can also be applied to resources. In several of the access control models resources are organised into a hierarchy and lower-level resources inherit the access rights of higher-level resources. For example, document libraries are often organised into a hierarchy. Given the following triples:

```
:dmsProjs a :DocumentLibrary "([[:employee]])"
:dmsProjsRpts a :DocumentLibrary
:dmsProjsRpts :isPartOf :dmsProjs
```

we can infer that:

```
:dmsProjRpts a :DocumentLibrary "([[:employee]])"
```

Rule 4.5 (Propagation based on resource hierarchies)
Here the rule states that if an $?S_2$ is part of $?S_1$ then the access rights of triples with $?S_1$ i.e. λ_1 should be propagated to $?S_2$.

$$\frac{?S_1\ ?P_1\ ?O_1\ ?\lambda_1,\ ?S_2\ ?P_2\ ?O_2\ ?\lambda_2,\ ?S_2\ \texttt{:isPartOf}\ ?S_1}{?S_2\ ?P_2\ ?O_2\ (?\lambda_2 \oplus_{ac} ?\lambda_1)}$$

Resources Categorisation. However resources can also be categorised by type, for example, views or objects in M_{VBAC}, and all resources of a particular type can inherit the access rights that have been assigned to that type. Access rights placed on a report object can be propagated to all objects of type report. Given the following triples:

```
:Report a :Document "([[:employee]])"
:Report1 a :Report
```

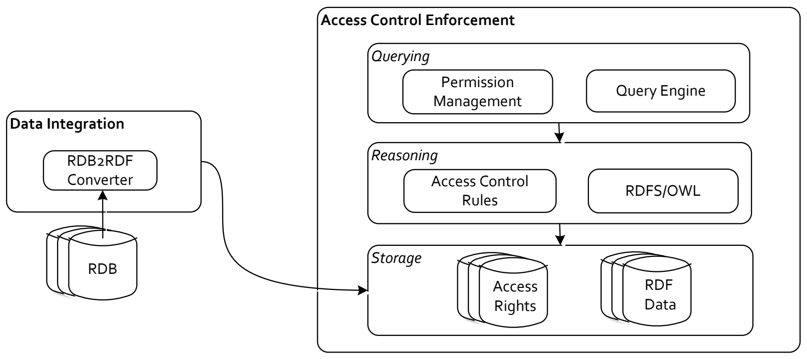


Figure 4.4: RDF data integration and access control enforcement framework

we can infer that:

```
:Report1 a :Report "([[:employee]])"
```

Rule 4.6 (Propagation based on resource typing)
Below the rule states that if $?O_2$ is a type of $?O_1$ then the access rights of triples with $?O_1$ i.e. λ_1 should be propagated to $?O_2$.

$$\frac{?S_1\ ?P_1\ ?O_1\ ?\lambda_1,\ ?S_2\ ?P_2\ ?O_2\ ?\lambda_2,\ ?O_2\ \texttt{a}\ ?O_1}{?S_2\ ?P_2\ ?O_2\ (?\lambda_2 \oplus_{ac} ?\lambda_1)}$$

Hierarchical Access Rights Subsumption. The access rights themselves can form a hierarchy whereby each permission level can include the access rights of the permission level below it. For example, we can assume update access from delete access and read access from update access. Given the following triples:

```
:Invoice3 a :Document "([[]], [[:john]], [[]])"
```

we can infer that:

```
:Invoice3 a :Document "([[:john]], [[:john]], [[]])"
```

Rule 4.7 (Propagation based on permission hierarchies)
Assuming that the ACL is a 3-tuple (R, U, D) and the permission hierarchy is stored as RDF. This rule states that if Update is part of Delete and Read is part of Update then the delete access rights should be propagated to the update annotation and the update access rights to the read annotation.

$$\frac{?S\ ?P\ ?O\ (?\lambda_1, ?\lambda_2, ?\lambda_3)}{?S\ ?P\ ?O\ ((?\lambda_1 \oplus_{ac} ?\lambda_2 \oplus_{ac} ?\lambda_3), (?\lambda_2 \oplus_{ac} ?\lambda_3), ?\lambda_3)}$$

4.4 Framework, Implementation and Evaluation

This section describes a set of components that are necessary for a data integration and access control enforcement framework, which caters for reasoning over both data and access control policies. It provides an overview of our implementation and presents an experimental evaluation of our prototype, which focuses on: (i) the *RDB2RDF converter*; (ii) the *reasoning engine*; and (iii) the *query engine*. The aim of this evaluation is simply to show the feasibility of our approach and, although we present different dataset sizes, at this point we are not looking to improve scalability and thus do not propose any kind of optimisations.

4.4.1 Access Control Enforcement Framework

An overview of the proposed framework is depicted in *Figure* 4.4, which is composed of two main modules: *Data Integration* and *Access Control Enforcement.* The Data Integration module is responsible for the conversion of existing RDB data and access control policies to RDF. Whereas the Access Control Enforcement module caters for the management of access rights and enables authenticated users to query their RDF data. We do not claim that this is an exhaustive list of the system components but rather the minimal components any access control framework requires. Noticeably, one missing component is the *authentication* component, which we do not focus on in this thesis. Authentication can be achieved by relying on WebID and self-signed certificates, working transparently over HTTPS.

Data Integration. The Data Integration module is responsible for the extraction of data and associated access rights from the underlying relational databases (RDBs). The information extracted is subsequently transformed into RDF and persisted in the *RDF Data* and *Access Rights* stores. Ideally, the data integration step would be carried out in conjunction with a domain expert, for example, to assist in defining an R2RML (Das et al., 2012) mapping that extracts and converts the relational data into RDF. In a pure Lined Data scenario, the data integration module is optional, in which case the data and access rights need to be populated manually or from specialised software systems.

Storage. The integrated data, retrieved from the original relational databases, is stored in the `RDF Data` store. Whereas, the access control policies over the RDF data are stored in the `Access Rights` store. A link between the two stores is necessary so that access policies can be related to existing graphs, triples, or resources. Any RDF store can be used as a back-end for storing this data. We do not restrict the representation format used for the Access Rights store which may be stored as quads, associated with reified triples or triple hash-codes.

Reasoning. For this component, we consider two distinct forms of inference: (i) data inference, where new triples are deduced from existing ones; and (ii) access rights inference, where new permissions are deduced from existing ones. For data inferencing we rely on well established forms of inference in the Semantic Web such as RDFS or OWL. However, if access control policies have been assigned to the data we also need a mechanism to propagate these policies when we infer new triples from existing ones. Another form of permission reasoning is required to support both permission management and data querying based on access control policies over RDF. Either backward or forward chaining could be used for both the data and the access rights inference.

Querying. SPARQL is the standard query language for RDF however the language itself does not provide any form of access control. It thus needs to be extended to ensure access is only given to the RDF triples that the user has been granted access to either explicitly or implicitly.

4.4.2 Implementation

In our prototype the data integration is performed using XSPARQL (Lopes et al., 2011), a transformation and query language for XML, RDB, and RDF, that allows data to be extracted from existing RDBs. Whereas the access control enforcement framework is a Prolog implementation of the Annotated RDF (Zimmermann et al., 2012) framework which enables reasoning over annotated triples.

RDB2RDF. XSPARQL was chosen over other alternative RDB2RDF approaches as both RDB and XML data formats are used extensively in organisations and also it has built-in support for the nquad format. In this chapter, we focus on relational databases but the ability to extract XML, from files or web-services, is desirable. The sample query presented in *Query* 4.1 is used to retrieve the `GroupTitle, ScopeUrl, RoleTitle` attributes from the `WebSiteView` relation and translate them into nquads. In this instance, the query is performed over a view that is used to combine data and permissions, however XSPARQL can also be used to extract data from multiple relations.

Query 4.1 (Sample XSPARQL query)

```
prefix ent: <http://urq.deri.org/enterprise#>

FOR GroupTitle, ScopeUrl, RoleTitle
FROM WebSiteView
CONSTRUCT {
entx:{$ScopeUrl} a entx:Web {$GroupTitle};
                entx:FullUrl {$ScopeUrl} {$GroupTitle};
                entx:Group {$GroupTitle} {$GroupTitle};
                entx:Role {$GroupTitle} {$GroupTitle};
}
```

For additional detail on XSPARQL, the reader is referred to Lopes et al. (2011).

Annotated RDF. Our Prolog Annotated RDF (Zimmermann et al., 2012) implementation, allows domain specific meta-information in the form of access rights to be attached to RDF triples. An overview of our annotated RDF access control domain model, presented in (Lopes et al., 2012), was provided in *Section* 4.2.2. The reasoning component is implemented by an extension of the RDFS inference rules. It allows the Annotated RDF reasoning engine to automatically determine the annotation values for any inferred triples based on the annotations of the premises. As such, we are actually performing data and annotation reasoning in the same step. We also support reasoning over the access rights alone, allowing information to be propagated according to the rules presented in *Section* 4.3.4. With the exception of *Rule* 4.7 our prototype provides for the presented rules, either in the form of inference rules or by explicit rules in the domain modelling. Our prototype currently does not cater for *Rule* 4.7 because the existing implementation only handles *read* permissions.

AnQL. Our query engine relies on an extension of SPARQL, AnQL (Lopes et al., 2010), which allows the annotated RDF data to be queried. AnQL extends SPARQL to consider also a fourth element in the query language. However, for the enforcement of access rights, the end user must not be allowed to write AnQL queries as this would pose a security risk, thus we rely on a translation step from SPARQL into AnQL. The Query Engine takes a SPARQL query and transparently expands it into an AnQL query, using the list of credentials provided by the authentication module. As authentication is outside the scope of this thesis, we simply extract end user credentials from the RDB applications, and maintain a mapping between the enterprise

Table 4.3: Dataset description

	DS_1	DS_2	DS_3	DS_4
database records	9990	17692	33098	63909
triples	62296	123920	247160	493648
file size (MB)	7.6	14.9	29.9	59.6

Table 4.4: Prototype evaluation

	DS_1	DS_2	DS_3	DS_4
RDB2RDF (sec)	39	52	92	179
Import (sec)	3	5	10	19
Inference engine (sec)	3218	11610	32947	58978
Inferred triples	59732	117810	237569	473292

employee and their usernames and roles. The AnQL query is subsequently executed against the Annotated RDF graph, which guarantees that the user can only access the triples annotated with their credentials.

4.4.3 Performance Evaluation

The benchmark system is a virtual machine, running a 64-bit edition of Windows Server 2008 R2 Enterprise, located on a shared server. The virtual machine has an Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, with 4 shared processing cores and 5GB of dedicated memory. For the evaluation we use XSPARQL to extract both the data and the access rights from two separate software application databases: a document management system (DMS) and a timesheet system (TS). We subsequently use the rules specified in *Section* 4.3.4 to propagate the permissions to relevant triples. In the case of the TS we only assigned existing access rights to one triple and let *Rule* 4.3 propagate the permissions to all triples with the same subject. As for the DMS we extracted the existing IRI hierarchy and the associated permissions and used *Rule* 4.5 to propagate the permissions to all data extracted depending on their location in the hierarchy. Existing type information was extracted from both systems and propagated using *Rule* 4.6. Finally we input the organisation structure as facts in the Prolog application and used *Rule* 4.4 to propagate permissions based on this hierarchy. As our prototype only supports read access we did not consider *Rule* 4.7 in our evaluation. The different datasets (DS_1, DS_2, DS_3, and DS_4) use the same databases, tables, and XSPARQL queries and differ only on the number of records that are retrieved from the databases. *Table* 4.3 provides a summary of each dataset, stating the number of database records queried, the number of triples generated, and the size in MegaBytes (MB) of the nquads representation of the triples.

RDB2RDF and Inference. *Table* 4.4, includes the time the data extraction process took in seconds (sec), the time it took to import the data into Prolog (sec), the evaluation time (sec) of the access control rules detailed in *Section* 4.3.4, and the number of triples inferred in this process. *Figure* 4.5a provides a high level overview of the times for each of the datasets. Based on this simple experiment we have hints that the extraction process and the loading of triples into Prolog behave linearly.

Table 4.5: Query execution time in seconds

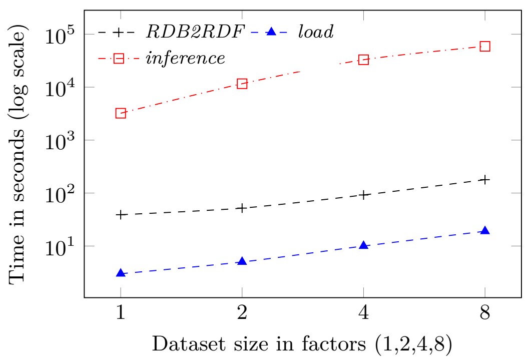
		DS_1	DS_2	DS_3	DS_4
	$\emptyset$	0.0000	0.0003	0.0013	0.0042
$1TP$	$\exists$	0.0065	0.0159	0.0300	0.0654
	$\nexists$	0.0081	0.0189	0.0316	0.0670
	$\emptyset$	0.0247	0.0544	0.1497	0.2679
$2TP$	$\exists$	0.0228	0.0638	0.0898	0.1845
	$\nexists$	0.0094	0.0198	0.0338	0.0690
	$\emptyset$	0.0169	0.0482	0.1322	0.2213
$3TP$	$\exists$	0.0241	0.0593	0.0943	0.1741
	$\nexists$	0.0101	0.0192	0.0316	0.0609

Query Engine. For the evaluation of the AnQL engine we created three separate query sets $1TP$, $2TP$ and $3TP$. The query sets were each composed of three queries with: one triple pattern $1TP$; two triple patterns $2TP$; or three triple patterns $3TP$. Each query was run without annotations ($\emptyset$), with a credential that appears in the dataset ($\exists$) and with a credential that does not appear in the dataset ($\nexists$). The results displayed in *Table* 4.5 were calculated as an average of 20 response times excluding the two slowest and fastest times. Based on this experiment we can see that there is an overhead for the evaluation of annotations when you query using a single triple pattern *Figure* 4.5b. However queries with a combination of annotations and either two or three triple patterns, *Figure* 4.5c and *Figure* 4.5d respectively, out perform their non annotated counterparts. Such behaviour is achieved in our implementation by pushing the filters into the triple patterns as opposed to querying the data and subsequently filtering the results. The results do not show a significant performance increase over the four datasets, between queries with annotations (*Figure* 4.5e) and those without annotations (*Figure* 4.5f). Furthermore we can see that there is no overhead associated with queries where the annotation is not present in the dataset. In fact such queries are actually more efficient when the query is made up of two or three triple patterns.

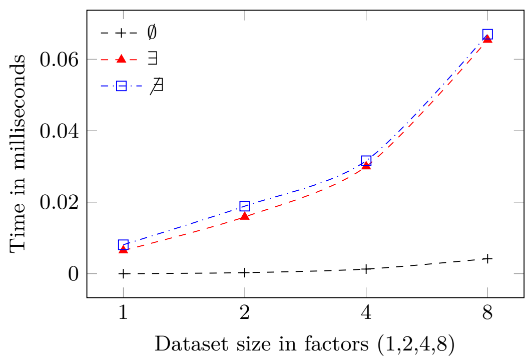
4.5 Related Work

A number of authors propose access control models for RDF graphs and focus on policy propagation and enforcement based on semantic relations. Concept-Level Access Control (CLAC) (Qin and Atluri, 2003), Semantic-Based Access Control (SBAC) (Javanmardi et al., 2006b) and the semantic network access control models proposed by Ryutov et al. (2009) and Amini and Jalili (2010), are well known works in this area. The policy language proposed by Javanmardi et al. (2006b) is not based on well defined semantics and no implementation details are provided. Ryutov et al. (2009) propose a path-based approach to policy composition. Amini and Jalili (2010) state that they use an analytical tableaux system, however they do not provide a mechanism for merging or for inference of permissions based on RDF structure. Qin and Atluri (2003) propose policy propagation of access control based on the semantic relationships among concepts. Javanmardi et al. (2006b), Ryutov et al. (2009) and Amini and Jalili (2010) enhance the semantic relations by allowing policy propagation based on the semantic relationships between the subjects, objects, and permissions.

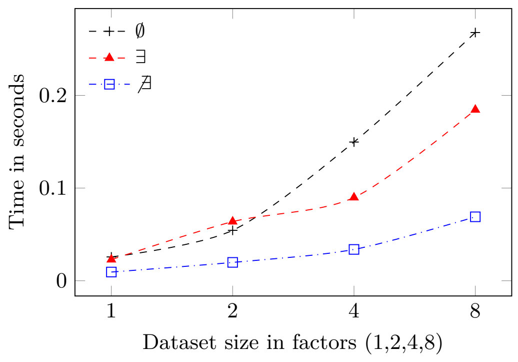
Dietzold and Auer (2006) describe the requirements needed by an RDF store, from a Semantic Wiki perspective. Apart from efficiency and scalability, the authors refer to the need for access



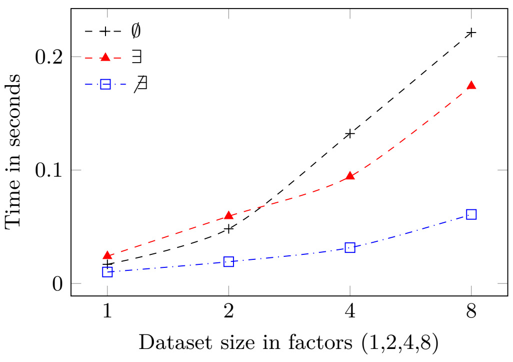
(a) Dataset load times



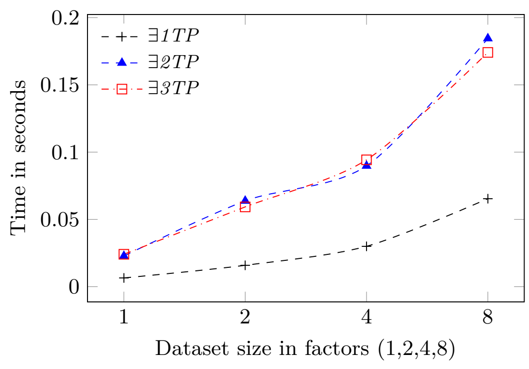
(b) Query execution times ($1TP$)



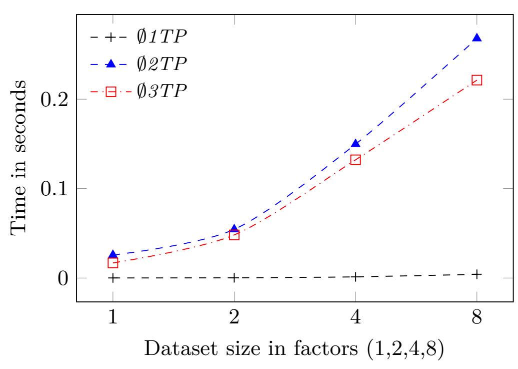
(c) Query execution times ($2TP$)



(d) Query execution times ($3TP$)



(e) Query execution times (∃)



(f) Query execution times (∅)

Figure 4.5: Load and query times across 4 datasets

control on a triple level and to integrate the structure of the organisation in the access control methods. The described system relies on a query engine (SPARQL is mentioned but no details are given) and a rule processor to decide the access control enforcement at query time. The system we propose in this chapter caters for both of these requirements and also integrates the access control into the annotation query language.

Hollenbach et al. (2009) present the possibility of maintaining metadata for RDF to enforce access control and touch upon of the work presented here, such as using rules for specifying access control, as possible extensions of their model. Providing access control on a resource level is also left as an open question, one we are tackling by the specification of rules.

Some work on extending query languages was presented by Abel et al. (2007), however this work pre-dates the SPARQL query language. In a similar fashion to the work presented in this chapter, their policy enforcement is also done by a query rewriting step however, their query rewriting does not involve including the user credentials but rather replicating the access policies within the query. They also take into account policies which allow or deny access to data.

Knechtel and Stuckenschmidt (2010) and Baader et al. (2009) also attach access control annotations to axioms in ontology, in order to allow access to subsets of the ontology to specific users. However the primary focus is on determining the minimal set of axioms that are necessary to support a certain conclusion. Although the setting is different to the one presented in this chapter, some of the algorithms for efficient annotation calculation may be ported to our modelling.

Costabello et al. (2012a) propose a lightweight vocabulary which defines fine-grained access control policies for RDF data. They focus specifically on modelling and enforcing access control based on contextual data. Sacco et al. (2011) present a privacy preference ontology which allows for the application of fine-grained access control policies to an RDF file. Both Costabello et al. (2012a) and Sacco et al. (2011) propose frameworks that rely on SPARQL `ASK` queries to determine access to resources based on the defined vocabularies.

In contrast to existing proposals the solution we present is tightly coupled with the cornerstone Semantic Web technologies (RDF, RDFS and SPARQL). We propose an integrated solution which extracts data and access rights from the enterprise software systems into RDF and an enforcement framework which handles reasoning over both the data and the access control statements. Approaches to date can be summarised as top-down approaches where the authors model access control based on RDF data structures and the access control requirements of open systems. We adopt a bottom up approach showing how existing access control models can be applied to RDF data.

4.6 Summary and Future Directions

RDF is a flexible data representation format that can be used for large scale integration of information from existing LOB applications. RDF can greatly benefit an enterprise, not only as a self-describing data model for their existing public data, but also as a means of extending internal data with RDF data freely available on the Web. The introduction of access control policies and enforcement over RDF, also allows organisations to selectively and securely share information with third parties (business partners, supplier or clients), using the existing web infrastructure.

In this chapter, we demonstrated how the Annotated RDF framework can be used to attach access control information to RDF data. Our model caters for access control at the triple level i.e. each RDF triple can contain different annotation values. Domain operators allow the annotations associated with identical triples to be combined and new annotations based on RDFS rules to be inferred. As access control policies for LOB applications are often stored in relational database tables, we discussed how RDB2RDF technology can be used to extract both the data and associated access rights from relational databases and how they can be represented as triples and annotations respectively. However, it is also feasible to specify annotations for information, which has not be extracted from existing systems and to update existing annotations directly. Although, on very large datasets, access control policy specification and administration at the triple level, would not be sustainable. To tackle this issue we proposed the management of permissions using domain specific inference rules. Based on our analysis of the predominant access control models found in the literature, we presented a set of rules, that can be used to simplify access control specification and administration. We also proposed a data integration and access control enforcement framework and described a possible implementation. In our implementation access control is enforced by transforming SPARQL queries into Annotated

SPARQL (AnQL) queries, using user credentials, which we assume have been verified by an external authentication service. A preliminary performance evaluation of the prototype, over enterprise data supplied by our project partner, was presented. As expected there is an overhead associated with access control for queries containing a single triple pattern. However, as a side effect of the query rewriting, from the user perspective queries with two or three patterns appear to run faster.

Nonetheless, the proposed solution is not without its drawbacks. From a performance perspective, given annotations are stored as lists of lists, for systems with many users and credentials, the proposed query rewriting and annotation matching will suffer from scalability issues. Whereas, from a modelling perspective, one of the major drawbacks of the proposed solution is that the existing modelling does not cater for access control at the graph level. As such, it cannot support graph wide permissions in general and the create permission in particular. A more flexible solution would be to cater for access control at multiple levels of granularity. Also although we identified the need for a permission management module and indicated that DAC is particularly suitable for managing access control over distributed data, further analysis is required in order to determine it suitability for the LDW.

Chapter 5

Applying Discretionary Access Control to Distributed RDF Data

In the early days, the web of documents was primarily used as a medium for sharing and linking static information. It wasn't until challenges with respect to data confidentiality, authenticity and integrity were addressed that electronic business became common place. It is not surprising that the Linked Data Web (LDW) is following a similar evolution. With the advent of SPARQL 1.1, it is possible for the LDW to evolve from a medium for publishing and linking data to a dynamic read/write distributed data source, that can support not only large scale data analytics but also the next generation of electronic business applications.

An important requirement for any data management system is the ability to protect data from unauthorised access. Traditionally data models, such as the relational and the XML data models, influenced the implementation of the corresponding access control mechanisms. A *data model* is an abstraction used to represent real world entities, the relationship between these entities and the operations that can be performed on them. Data models can be broadly categorised as relational, tree-based and graph-based. As Discretionary Access Control (DAC) allows users to delegate their permissions to others it is particularly suitable for managing access control over distributed data. Therefore, in this chapter, we examine how DAC can be used to restrict access to RDF data. We base our work on the DAC model as it has been successfully adopted by several relational DBMS vendors; because of its inherent flexibility; and its potential for handling context based authorisations in the future.

In *Chapter* 4, we demonstrated how together RDB2RDF technology and Annotated RDF can be used to securely integrated data from multiple LOB applications. The solution we presented also caters for the derivation of new annotations based on RDFS inference rules. We discussed how access control specification and maintenance can be simplified by using propagation rules based on resource types and hierarchies of subjects, access rights and resources. Given there is also a need for access control over RDF data in a broader web context, in this chapter we examine the specification, derivation and delegation of access control over distributed RDF data. Our research is guided by DAC principles and experiences applying these principles to the relational and tree-based data models. We discuss the need for additional rules to support DAC over the RDF data model, and demonstrate how authorisations based on graph patterns can be used to specify access control policies at different levels of granularity. In addition we demonstrate how the hierarchical Flexible Authorisation Framework (Jajodia et al., 2001) can be adapted, to cater for the specification, administration and enforcements of DAC policies over Linked Data.

In relation to access control for distributed RDF data, we make the following contributions.

We: (i) discuss how DAC principles can be used to restrict access to RDF data; (ii) propose a set of rules that are necessary for derivation of authorisations based on RDFSchema; (iii) describe how together pattern matching and propagation rules can be used to specify access control policies at multiple levels of granularity; (iv) demonstrate how the hierarchical Flexible Authorisation Framework (Jajodia et al., 2001) can be adapted to work with graph data; and (v) show how conflict resolution policies and integrity constraints can be used to ease maintenance and ensure the integrity of authorisations and propagation rules. Together these contributions provide a solid building block for DAC policy enforcement over distributed RDF data.

The remainder of the chapter is structured as follows: *Section* 5.1 describes how DAC is used to restrict access to the relational and tree-based data models. *Section* 5.2 discusses potential issues when we apply DAC to graph data and proposes possible handling mechanisms. *Section* 5.3 describes how the hierarchical Flexible Authorisation Framework (Jajodia et al., 2001) can be extended to cater for the RDF graph data model. *Section* 5.4 details how graph patterns, propagation rules, integrity constraints and conflict resolution policies can be used to specify and enforce access control over data represented using the RDF data model. *Section* 5.5 demonstrates how the proposed framework can be used to enforce access control over Linked Data exposed via SPARQL endpoints, and details the results of our performance evaluation of our authorisation enforcement and administration algorithms. *Section* 5.6 examines alternative approaches to the enforcement and administration of access control over RDF data. Finally *Section* 5.7 summarises the contributions and outlines directions for future work.

5.1 DAC Background

DAC policies limit access to data resources based on access rules that state the actions that can be performed by a *subject*. The term subject is an umbrella term used to collectively refer to users, roles, groups and attribute-value pairs. In DAC, access to resources is constrained by a central access control policy, however users are allowed to override the central policy and can pass their access rights on to other users (Sandhu and Samarati, 1994), known formally as delegation. Over the years the DAC model has been extended to consider: constraint based authorisations (e.g. time, location); access to groups of users, resources and permissions; support for both positive and negative authorisations; and conflict resolution mechanisms (Samarati and de Vimercati, 2001). In this section we describe how DAC is used to protect relational and tree-based data models.

5.1.1 Applying DAC to the Relational Model

In the relational model (*Figure* 5.1), data items are grouped into *n-ary relations*. A relation header is composed of a set of named data types known as *attributes*. The relation body is in turn made up of zero of more *tuples* (i.e. sets of attribute values). A *primary key*, composed of one or more attributes that uniquely identify each tuple, is defined for each relation. Relations are connected when a *foreign key* (i.e. one or more attributes) in one relation are linked to a *primary key* (i.e. one or more attributes) in another relation. Relations can be categorised as base relations or views. Base relations are actually stored in the database whereas views are virtual relations derived from other relations. Views are commonly used to: (i) provide access to information from multiple relations; (ii) restrict access to particular attributes or tuples; and (iii) derive data (for example sum, average, min and max).

In relational databases access is restricted both at a schema level (database, relations and attributes) and a data level (tuples and values). The access rights themselves are tightly coupled

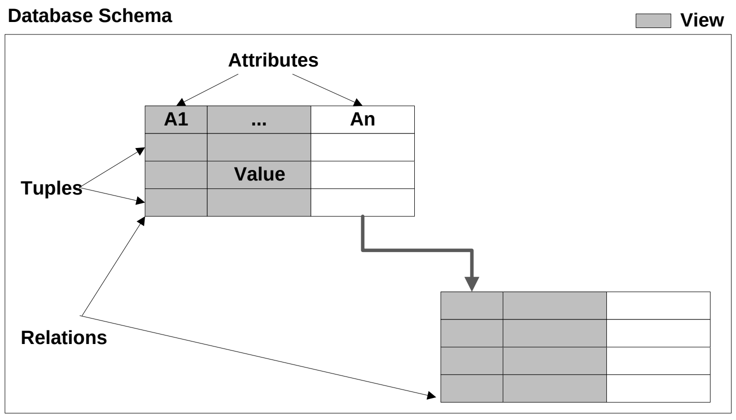


Figure 5.1: Relational data model

with database operations such as `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `DROP`. In addition the `GRANT` privilege allows users to grant access to others based on their own privileges. Griffiths and Wade (1976) describe how DAC is implemented in System R (Astrahan et al., 1976), an experimental DBMS developed to carry out research on the relational data model. Two of the underpinning principles of DAC are: (i) the derivation of implicit authorisations from explicit authorisations; and (ii) the delegation of access rights.

Authorisations explicitly defined at the schema level are implicitly inherited by other database entities. For example, (i) database authorisations are inherited by all database resources; (ii) relation authorisations are inherited by all tuples; and (iii) attribute authorisations are inherited by all attribute values. Aside from schema level derivations, Griffiths and Wade (1976) describe how views can be used to implicitly grant access to one or more tables, attributes or tuples spanning multiple relations. Under DAC, database users are granted sole ownership of the tables and views that they create. They can subsequently grant access rights to other database users. Griffiths and Wade (1976) and Bertino et al. (1997) discuss how the revocations process is complicated due to recursive delegation of permissions and propose algorithms which are used to revoke access rights.

5.1.2 Applying DAC to the Tree Model

In the tree data model information is organised into a hierarchical structure with a single root *node*. Each data item, represented as a node, is composed of one or more *attributes*. Relations are connected via parent-child links, whereby each parent can have many children, however each child can have only one parent. Both object-oriented databases and the Extensible Markup Language (XML) are examples of the tree data model. In the remainder of this section we focus on XML, however it is worth noting that the core derivation and delegation principles can also be applied to other instances of the tree data model.

In an XML data model (*Figure* 5.2) relations are represented as *elements* that can contain textual information and zero or more *attribute-value* pairs. Simple elements contain data values whereas complex data types are constructed from other elements and/or attributes, giving XML its hierarchical structure. A Document Type Definition (DTD) or an XMLSchema can be used to describe the structure of an XML document. In contrast to the relational model, XML data is not necessarily an instance of some schema.

Bertino et al. (2001) describe how DAC is implemented in Author-X, a prototype developed to demonstrate how access control policies can be applied to XML documents, that may or may

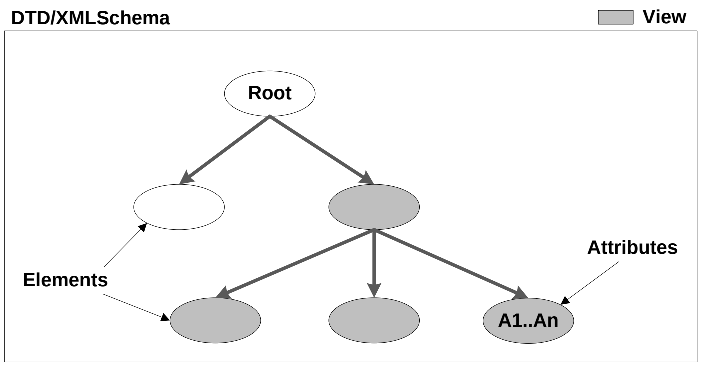


Figure 5.2: XML tree model

not conform to a DTD/XMLSchema. Similar to the relational model, tree-based access control can also be specified at both schema and data levels. From a schema perspective, access can be restricted based on the structure of the document/data item, a DTD or an XMLSchema. Whereas, data level restrictions can be applied to specific elements and attributes. Similar to the relational model, the access rights reflect the operations commonly performed on an XML document, for example `READ`, `APPEND`, `WRITE`, `DELETE` and `INSERT`.

Propagation of authorisations based on the is-part-of relationship between documents, elements, sub-elements and attributes, is one of the key features of DAC for XML (Bertino and Sandhu, 2005). Although implicit authorisations simplify access control administration, a knock on effect is that exceptions need to be catered for. In XML inheritance chains can be broken by explicitly specifying authorisations for leaf nodes. In addition, a combination of positive and negative authorisations are used to grant access in the general case, and to deny access for specific instances. The introduction of negative authorisations brings with it the need for conflict resolution mechanisms (for example, denial takes precedence). Gabillon (2004) describes how delegation of privileges can be adapted to work for XML databases. The author defines a security policy language for XML which incorporates SQL `GRANT` and `REVOKE` commands.

5.2 Applying DAC to the RDF Model

In this section, we describe how the graph data model differs from the tree data model. We discuss how DAC can be applied to graph-based data and detail the implication such structural differences have on access control in general. Although in this chapter we focus on RDF specifically, a number of observations could potentially be applied to other graph data models.

5.2.1 Graph and Schema Based Authorisations

The first step in the identification of access control requirements for RDF data is to identify the resources that need to be protected, and the access rights required. The graph data model alone is quite limiting when it comes to the management of access rights. Therefore, vocabularies such as RDFS and OWL are required in order to cater for more expressive authorisations.

RDF Resources. From a data perspective, access can be restricted to a node (subject or object), an arc (property), two connected nodes (triple), a collection of nodes and edges (multiple triples that share a common subject) or arbitrary views of the data (named graphs). Whereas, from a schema perspective authorisations can be applied to classes and properties. Given the

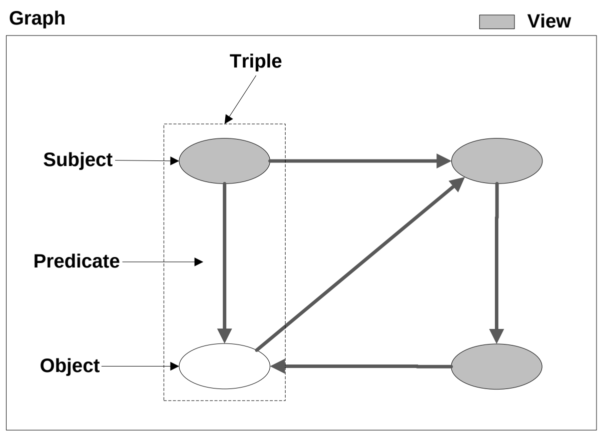


Figure 5.3: RDF graph model

```
foaf:Person rdf:type rdfs:Class.
foaf:givenName rdf:type rdf:Property.
foaf:givenName rdfs:domain foaf:Person.
foaf:lastName rdf:type rdf:Property.
foaf:lastName rdfs:domain foaf:Person.
```

Figure 5.4: Subset of FOAF vocabulary

tight coupling between schema and data items, authorisations based on classes (for instance, `foaf:person`) and properties (for instance, `foaf:givenName`) would need to be derived using schematic vocabularies such as RDFS or OWL[1]. In *Section* 5.2.2, we examine how permissions can be derived based on both the graph structure and RDFS.

Access Rights. The access rights of both the relational and XML data models are very similar and differ primarily by vocabulary. SPARQL proposes several operations similar to those that exist for relational and XML data (`SELECT`, `INSERT`, `DELETE/INSERT`, `DELETE` and `DROP`). However SPARQL also defines a number of additional query operations (`CONSTRUCT`, `ASK` and `DESCRIBE`) and a number of operations specifically for graph management (`CREATE, COPY, MOVE` and `ADD`). Notable omissions from the list of SPARQL operations are the `GRANT` and `REVOKE` commands which allows users to *grant access to* or *revoke access from* others based on their own privileges. In *Section* 5.2.3, we discuss how the grant and revoke operations could be accommodated in SPARQL.

Access Control Policy. An *Access Control Policy* details the actual authorisations and access restrictions to be enforced. Each authorisation is represented as a quad $\langle Sub, Acc, Sign, Res\rangle$ where *Sub* denotes the *subject* (not to be confused with an RDF triple subject), *Acc* the *access rights*, *Sign* indicates if the user is *granted* or *denied* access and *Res* represents the *resource* to be protected (i.e. rdf quad with optional variables, represented using a ? prefix, in any position). A matrix outlining the access rights that are appropriate for each operation (represented by an `X`) is provided in *Table* 5.1. For conciseness *Con* denotes two connected nodes, *Col* denotes a collection of nodes and edges and *Prop* denotes RDF properties. A sample access control policy is presented in *Table* 5.2. Each authorisation is labelled (A_n) to ease referenceability. (A_1), (A_2) and (A_3) grant access rights `SELECT`, `INSERT` and `DELETE` to the graph `entx:G1`. (A_4)

[1]OWL, `http://www.w3.org/TR/owl2-overview/`

```
entx:G1 {
entx:salary rdf:type rdf:Property.
entx:salary rdfs:domain foaf:Person.
entx:JoeBloggs rdf:type foaf:Person.
entx:JoeBloggs foaf:givenName "Joe".
entx:JoeBloggs foaf:lastName "Bloggs".
entx:JoeBloggs entx:salary "40000".
entx:MayRyan rdf:type  foaf:Person.
entx:MayRyan foaf:givenName "May".
entx:MayRyan foaf:lastName "Ryan".
entx:MayRyan entx:salary "80000".
}
```

Figure 5.5: Snapshot of enterprise employee data

Table 5.1: Relationship between access rights and resources

Rights	Node	Arc	Con.	Col.	View	Prop.	Class
`SELECT`	X	X	X	X	X	X	X
`CONSTRUCT`	X	X	X	X	X	X	X
`ASK`	X	X	X	X	X	X	X
`DESCRIBE`	X	X	X	X	X	X	X
`INSERT`			X				
`DELETE`			X				
`DROP`					X		
`CREATE`					X		
`COPY`					X		
`MOVE`					X		
`ADD`					X		
`GRANT`			X		X	X	X
`REVOKE`			X		X	X	X
			Data Model			Schema	

grants access to a particular class and (A_5) denies access to the salary property. If no explicit or implicit policy exists, it is possible to adopt either a safe conflict resolution policy (deny access by default) or an brave conflict resolution policy (grant access by default).

5.2.2 Derivation of Authorisations

In both the relational model and the tree model, authorisations can be derived based on the data schema. When it comes to the RDF data model, similar derivations are highly desirable as they simplify authorisation administration. Existing RDF database vendors adopt a view based approach to derivation, organising triples into named graphs based on the access control requirements and granting access to the entire graph. Although similar to views in relational databases, in this instance the graph is materialised. An alternative approach would be to derive permissions based on the graph structure. However, as it isn't possible to distinguish between schema and instance data such an approach alone is quiet limited. Therefore we propose an additional layer of derivations based on a vocabulary, such as RDFS, to define rules that leverage the semantic relations between nodes and edges. In the following rules, both the premises (above the line) and the conclusion (below the line) are represented as a 5 tuple $\langle S, P, O, \gamma, \lambda\rangle$. Where: (i) S represents a *subject*, P a *predicate* and O an *object* (together they represent a triple); (ii) γ is used to denote a named graph (which may or may not be the same for each triple); and (iii) λ

Table 5.2: Sample access control policy

	Sub	Rights	Sign	Res
(A_1)	`:Mgr`	`SELECT`	+	?S ?p ?o `entx:G1`
(A_2)	`:Mgr`	`INSERT`	+	?S ?p ?o `entx:G1`
(A_3)	`:Mgr`	`DELETE`	+	?S ?p ?o `entx:G1`
(A_4)	`:Emp`	`SELECT`	+	?S `rdf:type rdf:Class entx:G1`
(A_5)	`:Emp`	`SELECT`	-	`entx:salary rdf:type rdf:Property entx:G1`

is used to represent permissions (i.e. *authorisation subject*, *access rights* and *sign* attributes, represented as a tuple $\langle Sub, Acc, Sign\rangle$). By including the named graph in the derivation rules, it is possible to constrain the derivation to a particular graph, or alternatively to span multiple graphs. Such graphs in turn can be distributed across multiple data sources.

Similar to the tree model, we could assign permissions to a node and recursively derive authorisations for all nodes connected to it by arcs. Another approach would be to derive authorisations for all nodes along a particular path. Existing graph search algorithms, such as those proposed by Tarjan (1972), could potentially be used to recursively traverse the graph and assign permissions to the nodes. However, one limitation of the RDF data model is that it isn't possible to distinguish between schema and instances from the graph structure alone. For example, to restrict access to attributes we would need a means to derive permissions for all instances of a particular property type. Likewise, to restrict access to a relation we would need to derive permissions for all properties that are instances of a particular class. To accommodate schema based derivation, a combined data approach to derivation is warranted. The following rules can be used to derive access rights based on the RDFSchema vocabulary.

Rule 5.1 (Propagation from classes to instances)
Using this rule we can derive λ, which has been assigned to a class, for all instances of that class.

$$\frac{?X\ \texttt{rdf:type}\ \texttt{rdf:Class}\ \gamma\ \lambda,\ ?Z\ \texttt{rdf:type}\ ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma}{?Z\ ?Y\ ?A\ \gamma\ \lambda}$$

Rule 5.2 (Propagation from properties to instances)
In this rule, λ which has been assigned to a property of a class, is derived for all instances of that property.

$$\frac{\begin{array}{r}?X\ \texttt{rdf:type}\ \texttt{rdf:Class}\ \gamma,\ ?Y\ \texttt{rdf:type}\ \texttt{rdf:Property}\ \gamma\ \lambda,\\ ?Y\ \texttt{rdfs:domain}\ ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma\end{array}}{?Z\ ?Y\ ?A\ \gamma\ \lambda}$$

Rule 5.3 (Propagation from instances to properties)
The following rule propagates λ, assigned to an instance of a class, to property values associated with that instance.

$$\frac{\begin{array}{r}?X\ \texttt{rdf:type}\ \texttt{rdf:Class}\ \gamma,\ ?Z\ \texttt{rdf:type}\ ?X\ \gamma\ \lambda,\\ ?Y\ \texttt{rdfs:domain}\ ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma\end{array}}{?Z\ ?Y\ ?A\ \gamma\ \lambda}$$

Given a snapshot of the FOAF ontology, presented in *Figure* 5.4, a subset of an enterprise RDF dataset, presented in *Figure* 5.5, and a sample access control policy depicted in *Table* 5.2, we

Table 5.3: Snapshot of derived authorisations

	Sub	Rights	Sign	Obj
(DA_1)	:Emp	SELECT	+	entx:JoeBloggs rdf:type foaf:Person entx:G1
(DA_2)	:Emp	SELECT	+	entx:JoeBloggs foaf:givenName "Joe" entx:G1
(DA_3)	:Emp	SELECT	+	entx:JoeBloggs foaf:lastName "Bloggs" entx:G1
(DA_4)	:Emp	SELECT	+	entx:JoeBloggs entx:salary "40000" entx:G1
(DA_5)	:Emp	SELECT	+	entx:MayRyan rdf:type foaf:Person entx:G1
(DA_6)	:Emp	SELECT	+	entx:MayRyan foaf:givenName "May" entx:G1
(DA_7)	:Emp	SELECT	+	entx:MayRyan foaf:lastName "Ryan" entx:G1
(DA_8)	:Emp	SELECT	+	entx:MayRyan entx:salary "80000" entx:G1
(DA_9)	:Emp	SELECT	-	entx:JoeBloggs entx:salary "40000" entx:G1
(DA_{10})	:Emp	SELECT	-	entx:MayRyan entx:salary "80000" entx:G1

can derive additional authorisations such as those summarised in *Table* 5.3. (DA_1) to (DA_8) are derived by applying *Rule* 5.1 to (A_4). Whereas, (DA_9) and (DA_{10}) are inferred from *Rule* 5.2 and (A_5).

Two additional rules, which use `rdfs:subclass` (*Rule* 5.4) and `rdfs:subproperty` (*Rule* 5.5), are proposed to demonstrate the flexibility that can be gained from RDFS. More expressive rules based on richer vocabularies such as OWL could also be used. The database should be flexible enough to allow derivations to be turned on and off on a case by case basis.

Rule 5.4 (Propagation from class to subclass)
In this rule we use the RSFS subclass inheritance mechanism to derive the permissions λ assigned to a class for all subclasses.

$$\frac{?X\ \texttt{rdf:type rdf:Class}\ \gamma\ \lambda,\ ?Y\ \texttt{rdf:type rdf:Class}\ \gamma,\ ?Y\ \texttt{rdfs:subClassOf}\ ?X\ \gamma}{?Y\ \texttt{rdf:type rdf:Class}\ \gamma\ \lambda}$$

Rule 5.5 (Propagation from property to subproperty)
In this instance we use the subproperty inheritance to derive the permissions λ assigned to a property for all subproperties.

$$\frac{?X\ \texttt{rdf:type rdf:Property}\ \gamma\ \lambda,\ ?Y\ \texttt{rdf:type rdf:Property}\ \gamma,\ ?Y\ \texttt{rdfs:subPropertyOf}\ ?X\ \gamma}{?Y\ \texttt{rdf:type rdf:Property}\ \gamma\ \lambda}$$

5.2.3 Delegation of Access Rights

In both relational and XML databases, `GRANT` and `REVOKE` commands are used to manage delegation of access rights. The SPARQL 1.1 update language does not currently support the `GRANT` and `REVOKE` commands. It thus needs to be extended to cater for authorisation administration and delegation of access rights. We propose an adapted version of the SQL `GRANT` (*Definition* 5.1) and `REVOKE` (*Definition* 5.2) commands that cater for named graphs. We adopt the `USING NAMED` clause from other SPARQL 1.1 operations.

```
( USING ( NAMED )? IRIref )*
```

In addition in keeping with standard SPARQL we adapt the syntax of the `GRANT OPTION` replacing surrounding [] with () and a ? which indicates cardinality.

```
( WITH GRANT OPTION )?
```

`Privilege_name` denotes the privileges identified in *Section* 5.2.1 (`SELECT`, `CONSTRUCT`, `ASK`, `DESCRIBE`, `INSERT`, `DELETE/INSERT`, `DELETE`, `DROP`, `COPY`, `MOVE`, `ADD`). `Resource_name` represents one or more instances of the following RDF resources (`NAMED GRAPH`, `CLASS`, `PROPERTY`, `TRIPLE`). `User_name, role_name, attribute_value` are used to identify users, roles and attributes respectively and a reserved word `PUBLIC` is used to assign access to all users. Finally the `WITH GRANT OPTION` is used to provide users with the ability to delegate the access right(s) to others.

Definition 5.1 (GRANT command)
```
GRANT privilege_name
( USING ( NAMED )? IRIref )*
ON resource_name
TO {user_name |PUBLIC |role_name |attribute_name}
(WITH GRANT OPTION)?;
```

Definition 5.2 (REVOKE command)
```
REVOKE privilege_name
( USING ( NAMED )? IRIref )*
ON resource_name
FROM {user_name |PUBLIC |role_name |attribute_value_pair}
```

As revocation is not dependent on the data model existing approaches, such as cascading (Fagin, 1978; Griffiths and Wade, 1976) and non-cascading (Bertino, Elisa and Samarati, Pierangela and Jajodia, Sushil, 1993), devised for relational databases would also work for RDF databases (datastores).

5.2.4 Conflict Resolution

Conflicts can occur as a result of inconsistent explicit, derived and delegated policies. Samarati and de Vimercati (2001) discuss the need for different conflict resolution strategies depending on the situation. Earlier in this section, we proposed a number of derivation rules to ease RDF access control administration and stated that implicit authorisations should be overridden by explicit authorisations. It is important that the conflict resolution strategy proposed is in keeping with both the derivation rules and overriding mandate. In this chapter, we propose three complementary approaches to conflict resolution that fit well with DAC:

(i) *Explicit policies override implicit policies* (ensures that positive explicit authorisations will always prevail over negative implicit authorisations);

(ii) *Most specific along a path takes precedence* (allows users to grant access in the general case and deny access for specific instances); and

(iii) *Denial takes precedence* (caters for scenarios where we have a conflict between two explicit or two implicit authorisations).

In *Section* 5.2.2, we saw how derivation rules can result in conflicting authorisations, for example *Table* 5.3 (DA_4) and (DA_9) or (DA_8) and (DA_{10}). As both policies are implicit the *explicit policies override implicit policies* strategy is not applicable. In this instance the negative authorisation would prevail based on the *most specific along a path takes precedence* conflict resolution rule, as a policy assigned to a property is more specific than one applied to a class.

5.3 From a Hierarchical to a Graph Data System

The hierarchical Flexible Authorisation Framework, proposed by (Jajodia et al., 2001), was designed to address the need for a single authorization framework that could be used to restrict access to different classes of data objects. In this section, we introduce the hierarchical *Flexible Authorisation Framework* (Jajodia et al., 2001), henceforth referred to as H-FAF, and demonstrate how it can be extended to cater for DAC over the RDF graph data model, which we intuitively name G-FAF. We start by providing an overview of the original H-FAF data system and authorisation framework. Following on from this, we describe the individual G-FAF data system components in the context of RDF and extend the original formal definition of a `data system` to cater for graph data structures. Throughout the chapter, we use examples from the Berlin SPARQL Benchmark (BSBM) Dataset[2]. Prefixes are used as a shorthand notation for each vocabulary (for example, rdf, rdfs, bsbm) and variables are represented using a ? prefix. The following default prefix is used to increase readability:
(`http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1`).

5.3.1 H-FAF Data System and Authorisation Architecture

H-FAF was designed to address the need for a single authorization framework that could be used to restrict access to different classes of data objects (e.g. files, relations, objects, images), with different access control requirements. To this end, the authors provided a general definition for a data system and devised a modular architecture, which together with declarative rules, can be used to ease administration of access control policies, by exploiting the hierarchical structure of the data system components.

Data System Components. An authorisation framework describes the items to be protected (`data items`), to whom access is granted (users, groups, roles collectively known as `authorisation subjects`) and the operations that need to be protected (`access rights`). Together these components are known as a `data system`, which is formally defined by Jajodia et al. (2001) as follows.

Definition 5.3 (Data system)
A *Data System (DS)* is a 5-tuple
$\langle OTH, UGH, RH, A, Rel\rangle$ where: OTH is an object-type hierarchy (i.e. data items); UGH is a user-group hierarchy; RH is a role-hierarchy; A is a set whose elements are called *Access Rights* and *Rel* is a set of n-ary *Relationships* over the different elements of *DS*.

Authorisation Architecture. In addition to the formal `data system` definition Jajodia et al. (2001) propose a number of distinct components (authorisations, propagation policies, conflict resolution rules and integrity constraints) that together provide support for different access control policies.

- *Authorisations* are rules that dictate the access rights that `authorisation subjects` are allowed/prohibited to perform on `data items`.
- *Propagation policies* are used to derive implicit authorisations from explicit authorisations and the hierarchical data structure.
- *Conflict resolution rules* are used to support multiple conflict resolution strategies.

[2]Berlin SPARQL Benchmark (BSBM) - Dataset Specification, http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/Dataset/.

- *Integrity constraints* are used to specify restrictions on authorisation specification, thus decreasing the potential for runtime errors.

All components are specified using declarative rules, made up from explicit authorisations, historical authorisations, the hierarchical structure of, or the relationship between the different `data system` components. The conclusion differs depending on the rule type. As both the hierarchical structure of and the relationship between the `data system` components can be recorded as RDF, in this chapter we adapt and extend the original `data system` and rule definitions to work with the RDF Graph data model.

5.3.2 Individual Data System Components

In our graph based authorisation framework, which we call G-FAF, `authorisation subjects` are granted `access rights` to `data items` represented as one or more graphs that may or may not be disjoint.

Authorisation Subjects. Subject is an umbrella term used to collectively refer to user credentials. As we propose the verification of access based on credential matching we make no distinction between a user *playing* a role as opposed to *belonging* to a group. Therefore we merge both the user-group and role hierarchies, and refer to them simply as `authorisation subjects`. Such a merge does not impact the specification or enforcement of authorisations, and in fact affords a greater degree of flexibility with respect to the inclusion of additional types of user credentials. Also, as RDF is a web based distributed data model, we extend the subject definition to include user attributes. Users, groups, roles and attributes are often organised hierarchically, which means that access can be granted to a number of subjects simultaneously, simplifying permission management. Combined users, groups, roles and attributes are represented as one or more RDF graphs possibly disjoint.

Access Rights. Like databases and file systems access can be restricted based on the operations that a user attempts to execute on the data items. In the case of RDF, these operations take the form of: graph query operations (`SELECT`, `CONSTRUCT`, `ASK` and `DESCRIBE`); graph update operations (`INSERT, DELETE, DELETE/INSERT`); and a number of operations specifically for graph management (`DROP`, `COPY`, `MOVE` and `ADD`). Three additional access rights are required to facilitate the administration of access control, namely `GRANT`, `REVOKE` and `FULL ACCESS`. The `GRANT` privilege allows users to grant access to others based on their own privileges. The `REVOKE` privilege allows users to revoke the access rights they have granted to others. Whereas, `FULL ACCESS` is a super access right that subsumes all other access rights. We model the operations as one or more RDF graphs, and although we use vocabularies such as RDFS to define a partial order over the operations that can be used to infer implicit authorisations, we do not define how these are specified.

Data Items. In the Linked Data Web, information is represented as *RDF triples* used to make statements about resources in the form of subject-predicate-object expressions. An *RDF graph* is a finite set of RDF triples. *Named graphs* are used to collectively refer to a number of RDF statements. In this chapter, `data items` are represented as a set of *nquads*.

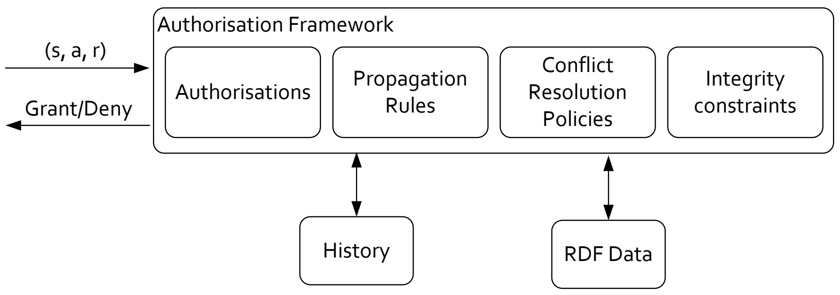


Figure 5.6: Authorisation framework

Definition 5.4 (RDF quad)
An *RDF Quad* is formally defined as a 4-tuple $\langle S, P, O, G \rangle \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L}) \times \mathbf{U}$, where S is called the *subject*, P the *predicate*, O the *object* and G the *named graph*. $\mathbf{U}$, $\mathbf{B}$ and $\mathbf{L}$, are in turn used to represent *URIs*, *blank nodes* and *literals* respectively.

Example 5.1 (RDF quad)
The following quad states that there exists a triple in the `Graph2013` *dataset stating that* `Vendor1` *is a type of* `Vendor`*.*
`:Vendor1 rdf:type bsbm:Vendor :Graph2013`

5.3.3 Extending the Original Data System Definition

We formally extend the original general definition of a data system to consider components that are represented as graphs as opposed to hierarchies. As the relationship between `data items`, `access rights` and `authorisation subjects` can also be represented as RDF, it is is not necessary to define a set of relations over the different elements of the data system. Although in this chapter we focus specifically on RDF, the extended data system definition is more general than the original and therefore it can be applied to both hierarchical and graph data models.

Definition 5.5 (Graph data system)
A *Data System (DS)* is defined as a 3-tuple $\langle DIG, ASG, ARG \rangle$ where: *DIG* represents one or more data graphs, that may be disjoint; *ASG* denotes one or more subject graph (nodes are use to represent users, groups, roles and attributes and edges represent the corresponding relations); and *ARG* stands for the graph of operations used to query and manipulate the data items in *DIG*.

5.4 G-FAF Authorisation Enforcement and Administration

Given an arbitrary but fixed `Graph Data System`, we describe the individual G-FAF components (*Figure* 5.6) and demonstrate how together these components can be used to deliver dynamic query results based on user credentials and to cater for the secure manipulation of RDF graph data (*Section* 5.5). We extend the original framework to include the *RDF Data Table*, which is necessary to infer new access control policies based on a combination of RDF data items and rules. Although we do not examine the role of the *History Table* in this thesis, it is worth noting that historical information is important for accountability, and also to cater for contextual access control policies that rely on historical data.

5.4.1 Specifying Authorisations using Quad Patterns

Authorisations are rules that dictate the `access rights` that `authorisation subjects` are allowed/prohibited to perform on `data items`. An *RDF Quad Pattern* is an RDF quad with optionally a variable V in the subject, predicate, object and/or graph position. A *Quad Pattern* is a flexible mechanism which can be used to grant/restrict access to an RDF quad, a collection of RDF quads (multiple quads that share a common subject), a named graph (arbitrary views of the data) or specific classes or properties.

Example 5.2 (RDF quad pattern)
A single quad pattern containing variables in both the subject and graph positions is used to match all quads with RDF type product.
$?s$ `rdf:type bsbm:Product` $?g$.

More expressive authorisations can be achieved using *RDF Graph Patterns*, which is composed of multiple quad patterns.

Example 5.3 (RDF graph pattern)
The following pattern, which contains variables in both the subject and graph positions is used to match all quads with RDF type `Product` *where the* `producer` *is* `Producer1`.
$?s$ `rdf:type bsbm:Product` $?g_1$.
$?s$ `bsbm:producer bsbm-inst:Producer1` $?g_2$.

In order to cater for certain conflict resolution strategies, and for the delegation of permissions, we extend the original authorisation definition to include `TYPE` and `BY` attributes. The `TYPE` attribute is necessary to differentiate between explicit and inferred authorisations, whereas the `BY` attribute is used to record information pertaining to the delegation of permissions. By default, the `TYPE` attribute is set to E for explicit and the `BY` attribute defaults to a reserved literal `OWNER`.

Definition 5.6 (Authorisation)
An authorisation is represented as a 6-tuple $\langle Sub, Acc, Sign, Res, Type, By\rangle$. *Sub* represents the `authorisation subject`. *Acc* is used to denote `access rights`. *Sign* indicates if the user is *granted* or *denied* access. *Res* symbolises the `data items` (i.e. Resources) to be protected, denoted by one or more *RDF Quad Patterns* known as an *RDF Graph Pattern*. *Type* is used to indicate if the authorisation is explicit (E) or implicit (I). Whereas, *By* represents the person who delegated the permission.

Example 5.4 (Authorisation)
Using the following authorisation an `Admin` *can grant all* `Partners UPDATE` *access to all triples in the* `Graph2008` *graph.*
$\langle$ `bsbm:Partner`, `UPDATE`, $+$, $\langle\ ?s, ?p, ?o,$ `:Graph2008`$\rangle$, *E*, `bsbm:Admin` $\rangle$

5.4.2 Propagation Policies

Propagation policies can be used to simplify authorisation administration, by allowing for the derivation of implicit authorisations from explicit ones. For example, we can derive new authorisations based on the logical organisation of `authorisation subjects`, `access rights` and `data items` (*Section* 4.3) or the RDFS vocabulary (*Section* 5.2.2). We provide a formal definition for a propagation rule which can be used as a blueprint for the specification of both general and specific derivation rules (*Definition* 5.7). In addition, we present a propagation algorithm

(*Algorithm* 5.1) which can be used to either evaluate the propagation policy at query time or alternatively to materialise implicit authorisations when authorisations are added or removed.

Definition 5.7 (Propagation rule)
A Propagation Rule is a a rule of the following format:
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle ?S_y, ?P_y, ?O_y, ?G_y \rangle, ?Type_y, ?By_y \rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x, ?P_x, ?O_x, ?G_x \rangle, ?Type_x, ?By_x \rangle$, $[\langle S_1, P_1, O_1, G_1 \rangle$
$\wedge ... \wedge \langle ?S_x, ?P_x, ?O_x, ?G_x \rangle \wedge ... \wedge \langle ?S_y, ?P_y, ?O_y, ?G_y \rangle \wedge ... \wedge \langle ?S_n, ?P_n, ?O_n, ?G_n \rangle]$
The premise is composed of an *Authorisation* and an ***`RDF Graph Pattern`***. If the Authorisation exists in the Authorisations and the RDF Quad Pattern exists in the RDF Data then we can infer the conclusion.

Example 5.5 (Subject hierarchy inheritance)
Using the following rule we can derive access rights assigned to employees for all managers.
$\langle$ `bsbm:Mgr`, $?Acc, ?Sign, \langle ?S, ?P, ?O, ?G \rangle, I, ?By \rangle \leftarrow$
$\langle$ `bsbm:Emp`, $?Acc, ?Sign, \langle ?S, ?P, ?O, ?G \rangle, ?Type, ?By \rangle$,
$[\langle$ `bsbm:Mgr`, `rdf:type`, `bsbm:Emp`, $?G \rangle \wedge \langle ?S, ?P, ?O, ?G \rangle]$

Example 5.6 (Class to instance propagation)
The following rules propagates the access rights assigned to a bsbm:Product *class to all instances of the class.*
$\langle ?Sub, ?Acc, ?Sign, \langle ?Z, ?Y, ?A, ?G_x \rangle, I, ?By \rangle \leftarrow$
$\langle ?Sub, ?Acc, ?Sign, \langle$ `bsbm:Product`, `rdf:type`, `rdf:Class`, $?G_y \rangle, ?Type, ?By \rangle$,
$[\langle ?Z,$ `rdf:type`, `bsbm:Product`, $?G_z \rangle \wedge \langle ?Z, ?Y, ?A, ?G_x \rangle]$

$\langle ?Sub, ?Acc, ?Sign, \langle ?Z, ?Y, ?A, ?G_x \rangle, I, ?By \rangle \leftarrow$
$\langle ?Sub, ?Acc, ?Sign, \langle$ `bsbm:Product`, `rdf:type`, `rdf:Class`, $?G_y \rangle, ?Type, ?By \rangle$,
$[\langle ?Z,$ `rdf:type`, `bsbm:Product`, $?G_z \rangle \wedge \langle ?A, ?Y, ?Z, ?G_x \rangle]$

5.4.3 Conflict Resolution Rules

Rather than propose a particular conflict resolution strategy, we provide a formal definition for a conflict resolution rule (*Definition* 5.8), that can be used to determine access given several different conflict resolution strategies. For example, conflict resolution policies based on the structure of the different `graph data system` components; the sensitivity of the data requested; or contextual conditions pertaining to the requester. As multiple conflict resolution rules may be applicable, each rule should be assigned a priority, and rules should be evaluated in order until the conflict has been resolved. The default rule, which matches everything, is assigned the lowest priority thus ensuring that a conclusion of grant or deny can always be drawn.

```
Data: Authorisations, PropPolicy, RDFData
Result: Authorisations
forall the Pol in PropPolicy do
    if Authorisations CONTAINS Pol.Premise.Auth then
        if RDFData CONTAINS Pol.Premise.GraphPattern then
            Authorisations += Pol.Conclusion.Auth
        end
    end
end
return Authorisations
```

Algorithm 5.1: Applying the propagation rules

Definition 5.8 (Conflict resolution rule)
A *Conflict Resolution Rule* is a rule of the following format:
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x|\texttt{VAR}|\texttt{CON}, ?P_x|\texttt{VAR}|\texttt{CON}, ?O_x|\texttt{VAR}|\texttt{CON}, ?G_x|\texttt{VAR}|\texttt{CON}\rangle, ?Type_x, ?By_x\rangle$
$\leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x|\texttt{VAR}|\texttt{CON}, ?P_x|\texttt{VAR}|\texttt{CON}, ?O_x|\texttt{VAR}|\texttt{CON}, ?G_x|\texttt{VAR}|\texttt{CON}\rangle, ?Type_x, ?By_x\rangle$
$>$
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle ?S_y|\texttt{VAR}|\texttt{CON}, ?P_y|\texttt{VAR}|\texttt{CON}, ?O_y|\texttt{VAR}|\texttt{CON}, ?G_y|\texttt{VAR}|\texttt{CON}\rangle, ?Type_y, ?By_y\rangle$
where $>$ indicates the authorisation to the left of the $>$ symbol takes precedence over the authorisation to the right; $?Sub$, $?Acc$, $?Type$ and $?By$ match authorisation *subjects*, *access rights*, *type* and *by* attributes. $?S$, $?P$, $?O$ and $?G$ denote RDF *subjects*, *predicates*, *objects* and *named graphs*; ? indicates the ability to specify either constants or variables; `CON` and `VAR` are reserved words used match constants and variables respectively; | symbolises logical disjunction;

Example 5.7 (Most specific takes precedence)
The following rule states that authorisations assigned to specific subject, predicate and objects in a graph override authorisations assigned to the whole graph.
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON}\rangle, ?Type_x, ?By_x\rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON}\rangle, ?Type_x, ?By_x\rangle >$
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle \texttt{VAR}, \texttt{VAR}, \texttt{VAR}, \texttt{CON}\rangle, ?Type_y, ?By_y\rangle$

Example 5.8 (Explicit overrules implicit)
Using the following rule it is possible to state that explicit authorisations override implicit authorisations.
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x, ?P_x, ?O_x, ?G_x\rangle, E, ?By_x\rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x, ?P_x, ?O_x, ?G_x\rangle, E, ?By_x\rangle >$
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle ?S_y, ?P_y, ?O_y, ?G_y\rangle, I, ?By_y\rangle$

5.4.4 Integrity Constraints

Integrity constraints are used to restrict the type of authorisations that can be specification based on existing relationships between SPARQL operations and RDF data items. For example, `INSERT` and `DELETE` can only be applied to an RDF quad, whereas `DROP`, `CREATE`, `COPY`, `MOVE` and `ADD` should only be associated with a named graphs. We provide a formal definition of an integrity constraint (*Definition* 5.9) and demonstrate how general rules and type checking in

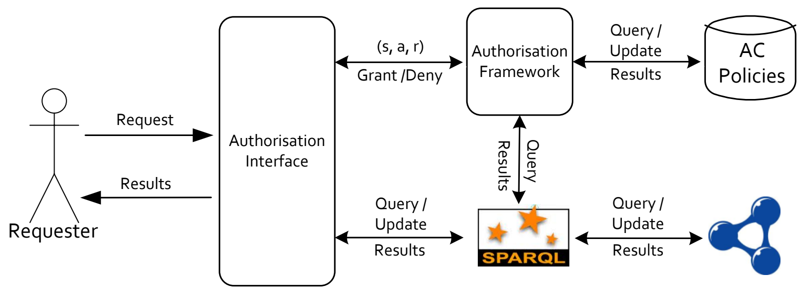


Figure 5.7: Authorisation architecture

particular can be used to constrain the specification of the `INSERT` (*Example* 5.9) and the `CREATE` (*Example* 5.10) access rights.

Definition 5.9 (Integrity constraint)
An Integrity Constraint is a rule of the following format:
error ←
$[\neg]\ \langle ?Sub, ?Acc, ?Sign, \langle ?S|\texttt{VAR}|\texttt{CON}, ?P|\texttt{VAR}|\texttt{CON}, ?O|\texttt{VAR}|\texttt{CON}, ?G|\texttt{VAR}|\texttt{CON}\rangle, ?Type, ?By\rangle$

where square brackets [] are used to denote the optional classical negation prefix (¬); $?Sub$, $?Acc$, $?Type$ and $?By$ match authorisation *subjects*, *access rights*, *type* and *by* attributes. $?S$, $?P$, $?O$ and $?G$ denote RDF *subjects*, *predicates*, *objects* and *named graphs*; ? indicates the ability to specify either constants or variables; `CON` and `VAR` are reserved words used match Constants and Variables respectively; | symbolises logical disjunction;

Example 5.9 (INSERT constraint)
Using an integrity constraint we can ensure that the INSERT operation can only be applied to RDF quads.
error ← $\neg\langle ?Sub, \texttt{INSERT}, ?Sign, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON}\rangle, ?Type, ?By\rangle$

Example 5.10 (CREATE constraint)
The following integrity constraint ensures that the CREATE graph management operation is only be applied to Named Graphs.
error ← $\neg\langle ?Sub, \texttt{CREATE}, ?Sign, \langle \texttt{VAR}, \texttt{VAR}, \texttt{VAR}, \texttt{CON}\rangle, ?Type, ?By\rangle$

5.5 Application and Evaluation

Although the architecture we propose will work with any query language, in this chapter we describe how it can be used in conjunction with SPARQL. First we discuss how the framework can be used to enforce and administer access control over Linked Data sources. Next we examine the performance of our Java implementation of the framework.

5.5.1 Applying the G-FAF Framework to Linked Data

RDF data is mostly exposed on the web via SPARQL endpoints. *Figure* 5.7 depicts how G-FAF can be used for the enforcement and administration of access control policies over Linked Data sources. We do not focus on authentication in this thesis, and thus we assume that the

```
Data: AuthRequest, AuthHashMap, ConflictPolicy
Result: grant/deny
domAuth = false
key = AuthRequest.subj + AuthRequest.acc
authHashSet = getKeyAuthHashSet(key)
quadHashMap = createQuadAuthHash(AuthHashSet)
if quadHashMap CONTAINS AuthReq.res then
    authMatches += quadHashMap.Auth
    domAuth = quadHashMap.Auth
end
if authMatches CONTAINS true and authMatches CONTAINS false then
    domAuth = applyConflictRes(authMatches, ConflictPolicy)
end
return domAuth.Sign
```

Algorithm 5.2: Authorisation enforcement algorithm

```
Data: AuthRequest, AuthHashMap, IntegrityPolicy
Result: true/false
newAuth = AuthRequest + Sign.Grant + Type.E + By.Owner
if AuthRequest.OPER = INSERT or AuthRequest.OPER = ADD or
AuthRequest.OPER = COPY then
    if IntegrityCheck(AuthRequest, IntegrityPolicy) = true then
        AuthHashMap += newAuth
        AuthHashMap += applyPropRules(Authorisations, PropRules)
        return true
    end
end
else if AuthRequest.OPER = DELETE or AuthRequest.OPER = DROP or
AuthRequest.OPER = MOVE then
    AuthHashMap -= newAuth return true
end
return false
```

Algorithm 5.3: Authorisation administration algorithm

credentials supplied by the requester have been successfully authenticated via alternative means, for example WebID and self-signed certificates, working transparently over HTTPS.

Enforcement of Authorisations. In addition to the usual SPARQL query, the requester must submit their credentials, which are verified by an external authentication system. The *Authorisation Interface* will map the SPARQL query to an *Authorisation Request* of the form $\langle Sub, Acc, Res \rangle$ (a subset of *Definition.* 5.6) and submit it to the *Authorisation Framework* (*Figure* 5.7). The authorisation algorithm (*Algorithm* 5.2) will check if the *Authorisation Request* can be derived using the *Authorisations* and the *Conflict Resolution Policies.* If the algorithm manages to successfully derive the authorisation, access to the requested data will be granted. Otherwise the request will be denied. If the authorisation framework grants access, the *Authorisation Interface* will pass the SPARQL query to the *Query Engine*, which in turn processes the query in the normal way. Finally, the query results are returned to the *Requester* via the *Authorisation Interface.* In the current implementation, the `subject` must be granted access to each triple in order to be permitted to execute the query. In *Chapter* 6, we investigate the data integrity implications of granting access to subsets of the graph pattern through query rewriting.

Table 5.4: Dataset and authorisations description

	DS_1	DS_2	DS_3	DS_4	DS_5
quads	250223	500258	1000109	2000164	4000936
scale factor	830	1689	3402	6830	13780
file size (MB)	24.5	49	98	195	391
	QS_1	QS_2	QS_3	QS_4	QS_5
authorisations	60000	120000	240000	480000	960000
file size (MB)	6.5	13	26	53	105

Table 5.5: Queries over increasing datasets

		DS_1	DS_2	DS_3	DS_4	DS_5
	$\emptyset$	345	657	1432	2604	5005
QS_S	$\exists$	429	700	1164	2549	5149
	$\emptyset$	8	8	9	8	9
QS_U	$\exists$	9	9	9	9	9

Administration of Authorisations. We propose an ownership model, whereby the data producer is granted `FULL ACCESS` to the data items they create. When a user issues a graph update or graph management query, access is verified using the authorisation algorithm (*Algorithm* 5.2). If authorisation succeeds, the SPARQL query is passed to the *Query Engine*. For `INSERT`, `ADD` or `COPY` operations, if the query succeeds the administration algorithm (*Algorithm* 5.3) ensures it adheres to the integrity constraints prior to creating a new authorisation. For `DELETE`, `DROP` or `MOVE` operations, if the query succeeds the administration algorithm (*Algorithm* 5.3) simply deletes relevant authorisations from the access control policy. In both instances, the update of both the RDF graph and the authorisations should be wrapped in a transaction to ensure that either both or neither succeed.

Delegation of Access Rights. In order to cater for delegation of access control, a number of administration modules are required, for example the ability to: grant/revoke access rights to others and list your own access rights and those assigned to others. As data producers are granted `FULL ACCESS` to the data items they create, they have the ability to `GRANT` and `REVOKE` access to/from others. As neither the grant nor the revoke algorithms are dependent on the data model, traditional revocation approaches such as cascading (Fagin, 1978; Griffiths and Wade, 1976) and non-cascading (Bertino, Elisa and Samarati, Pierangela and Jajodia, Sushil, 1993) could be used in conjunction with the proposed framework.

5.5.2 Performance Evaluation

For the evaluation of G-FAF we created three separate experiments to:

(i) examine the overhead associated with access control over different data sets;

(ii) deduce the impact given an increasing number of authorisations; and

(iii) determine the performance increase for several propagation rules (the most expensive administration operation).

Table 5.6: Queries over increasing authorisations

	AS_1	AS_2	AS_3	AS_4	AS_5
QS_S *query time (ms)*	5056	4801	4861	4892	4869
QS_U *query time (ms)*	9	8	9	8	9

Table 5.7: Propagation rules performance

	DS_1	DS_2	DS_3	DS_4	DS_5
AS_5 *query time (ms)*	98531	104894	107017	106823	106248
	AS_1	AS_2	AS_3	AS_4	AS_5
DS_5 *query time (ms)*	6248	12733	24257	51339	112887

Evaluation Setting. The benchmark system has an Intel(R) Xeon(R) CPU 8 core 2.13GHz processor, 64 GB of memory and runs Debian 6.0.3. The authorisation framework is written in Java and is evaluated over an in memory store using Jena ARQ. Both the datasets (*Table* 4.3) and the queries are generated from the Berlin SPARQL Benchmark (BSBM) dataset. Two separate query sets are created:

(i) QS_S which contained *10* `SELECT` queries; and

(ii) QS_U which contained *5* `INSERT` and *5* `DELETE` queries.

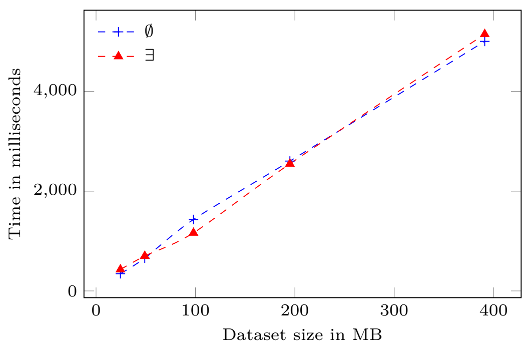
In both instances the queries are composed of a combination of one, two and three triple patterns. Access is granted or restricted to: all quads (?s ?p ?o ?g); a particular graph (?s ?p ?o G1); all quads of type offer (?s rdf:type bsbm:Offer ?g); all classes (?s rdf:type rdf:Class); and all properties (?s rdf:type rdf:Property). Users are either assigned (`SELECT`; `SELECT` & `INSERT`; `SELECT`, `INSERT` & `DELETE`) or denied (`DELETE`; `INSERT` & `DELETE`; `SELECT`,`INSERT` & `DELETE`) access to single quad patterns. The integrity constraints presented in *Examples* 5.9 and 5.10 are added, to ensure that `INSERT` and `DELETE` operations are only applied to RDF quads. The conflict resolution rules presented in *Examples* 5.7 and 5.8, along with an additional denial takes precedence rule, are executed in the event of a conflict. The datasets, queries and the conflict resolution, integrity and propagation rules used in the experiments can be found at `http://gfaf.sabrinakirrane.com/`. All calculations presented were based on an average of 20 response times, excluding the two slowest and fastest times.

Evaluation Results. In order to evaluate the enforcement algorithm we ran both the select (QS_S) and the update (QS_U) query sets, without access control ($\emptyset$) and with access control for users who were granted access ($\exists$), over an authorisation set containing *588,000* grant and *402,001* deny authorisations. As expected, the results indicate that select query execution times are not impacted when we increase the dataset (*Table* 5.5). However, the marginal increase in performance times over increasing authorisations (*Table* 5.6) was unexpected. Such behaviour can be attributed to the fact that all authorisations are indexed by a combined `subject` and `access right` key and subsequently by graph pattern (*Algorithm* 5.2). For the evaluation of the propagation rules, we examined the impact associated with three schema based derivations from:

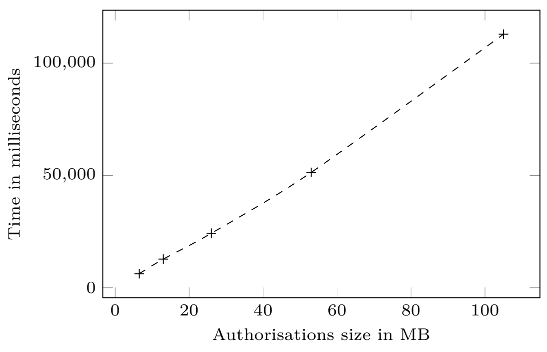
(i) classes to all instances of that class;

(ii) properties to all instances of that property;

(iii) an instance to property values associated with that instance.



(a) Select query performance



(b) Rules over increasing authorisations

Figure 5.8: Query and propagation times

Again we ran the experiment over increasing datasets and authorisations (*Table* 5.7). Based on the results we can see that reasoning behaves linearly when the number of authorisations are increased, while there is little or no impact when the number of quads are increased. Given that the five different authorisation types are very general (all quads, all classes, all properties, a particular graph, a particular type), and scaling involves adding authorisations for new subjects, the number of rules that are triggered increases linearly with the number of authorisations. In turn, the negligible impact on performance times over increasing data can be attributed to the fact that the number of propagation rules is not dependent on the size of the dataset.

5.6 Related Work

Initially Semantic Web researchers focused on the modelling and the enforcement of access control over RDF stores. A number of authors have proposed access control models for RDF that define access control policies based on RDF patterns that can be mapped to one or more RDF triples (Reddivari et al., 2005; Jain and Farkas, 2006; Dietzold and Auer, 2006; Abel et al., 2007). Reddivari et al. (2005) define a set of actions required to manage an RDF store, and demonstrate how query based access control can be used to permit or prohibit requested actions. The authors propose default and conflict preferences that can simply be set to either permit or deny. Jain and Farkas (2006) propose a data level security model which can be used to protect both explicit and inferred triples. They provide a formal definition for each of the RDF security objects and define an algorithm to generate security labels for both explicit and inferred triples based on a security policy and a conflict resolution strategy. Limited details of the implementation are supplied and no evaluation is performed. Abel et al. (2007) propose the evaluation of access control policy constraints at both the query layer and the data layer. Contextual conditions that are not depended on RDF data are evaluated by a policy engine. While the query is expanded to include the contextual conditions that are dependent on RDF data. Such an approach requires the substitution of variables to ensure uniqueness. However, in doing so they are able to leverage the highly optimized query evaluation features of the RDF store. The authors adopt a denial by default conflict resolution strategy.

Gabillon and Letouzey (2010) highlight the possible administration burden associated with the maintenance of access control policies that are based on triple patterns. They propose the logical distribution of RDF data into SPARQL views, and the subsequent specification of access control policies based on existing RDF Graphs or predefined views. They describe a query based enforcement framework whereby each user defines a security policy for the RDF graph/views that they own. The authors acknowledge the need for conflict resolution, however they do not propose a conflict resolution strategy.

More recently, the focus has shifted to the specification and enforcement of access control over web resources. Costabello et al. (2012a) and Sacco et al. (2011) both propose access control ontologies and enforcement frameworks that rely on SPARQL ASK queries to determine if the requester possesses the attributes necessary to access the resource. Costabello et al. (2012a) use context data supplied by the requester to limit the scope of the SPARQL query to authorised named graphs. The authors propose the disjunctive evaluation of policies, thus circumventing the need for a conflict resolution mechanism. Whereas, Sacco et al. (2011) provide a filtered view of a data providers FOAF profile based on a matching between data provider privacy preferences and requester attributes. Policies can be specified for an entire graph, one or more triples or individual subjects, predicates and objects. The authors do not propose any conflict resolution strategy. Neither Costabello et al. (2012a) nor Sacco et al. (2011) perform any reasoning over the access control policies they propose.

Other researchers adopt a rule based approach to access control. Dietzold and Auer (2006) define access control requirements from a Semantic Wiki perspective. The authors propose a filtered data model using a combination of SPARQL queries and SWRL rules. Li et al. (2005) also adopt a rule based approach providing users with a more intuitive way to specify access control policies. Both Dietzold and Auer (2006) and Li et al. (2005) use rules to give a more explicit meaning to the access control policies, as opposed to authorisation derivation in our case.

Several researchers propose access control models for RDF graphs and focus on policy propagation and enforcement based on semantic relations (Qin and Atluri, 2003; Javanmardi et al., 2006b; Ryutov et al., 2009; Amini and Jalili, 2010). None of the authors examine access control from either a data model or a database perspective. Like us, Jain and Farkas (2006) derive authorisations and propose conflict resolution mechanisms. They adopt a multilevel label-based approach to access control where policies are specified in terms of RDF patterns associated with an instance, a schema and a security classification. However, the derivations they propose are limited to RDFS entailment rules.

Only Jain and Farkas (2006) and Javanmardi et al. (2006b) actually mention DAC, and even then they just describe DAC and do not examine how their approach compares or contrasts. A number of authors who use Semantic Technology for access control, however do not apply their approach to the RDF data model, detail their support for DAC (Kodali et al., 2003; Damianou et al., 2001; Weitzner et al., 2006a). However to the best of our knowledge to date no one has investigated the application of DAC to the RDF data model. We fill this gap by examining how DAC has been used to protect the relational and tree data models, and by proposing strategies that allow us to apply DAC to the RDF graph model. We identify the resources to be protected and the access rights required, based on the RDF data model and SPARQL the predominant query language for RDF. In addition, we propose mechanisms to assist with access control administration through derivation of authorisation, delegation of permissions and conflict resolution

In *Chapter* 4, we demonstrated how annotated RDF could be used to assign permissions to triples and to infer annotations based on RDFS entailment rules. In this chapter, we provide a general mechanism for the administration and enforcement of access control policies using a combination of propagation rules, integrity constraints and conflict resolution policies. In addition, we allow for the specification of authorisations based on quad patterns, thus catering for authorisation at multiple levels of granularity, for example one or more graphs, triples, classes or properties.

5.7 Summary and Future Directions

With the introduction of RDF update languages, such as SPARQL 1.1, it is now feasible to both query and manage distributed and linked RDF data. However, like web applications and web services, SPARQL endpoints need to protect the security of the data source and the privacy and the integrity of the data therein. Although the RDF data model has been around for over a decade, little research has been conducted into the application of existing access control administration techniques to RDF data. In this chapter, we discussed how the DAC model can be applied to RDF, a distributed graph-based data model. We identified the resources to be protected and the access rights required based on the graph data model and SPARQL query operations respectively. We proposed a layered approach to authorisation derivation based on the graph structure and RDFS. We subsequently identified a number of rules that can be used to manage authorisations in an intuitive manner.

Given the dynamic nature of RDF, we identified the need for a framework that was capable of supporting a variety of authorisations, derivation, delegation and conflict resolution policies. The hierarchical Flexible Authorisation Framework, proposed by Jajodia et al. (2001), was designed to address the need for a single authorization framework that could be used to restrict access to different classes of data objects. Although originally designed to work with hierarchical data, we adapted the framework to cater for DAC over graph data. We provided a formal definition of authorisations, propagation rules, conflict resolution policies and integrity constraints, within the context of RDF. In addition, we described how together these components can simultaneously provide access control over interlinked RDF graphs. The results of our initial performance evaluation are very promising, as in general they show only a negligible increase in query processing time in light of increasing datasets and authorisations, and an linear increase in derivation over increasing authorisations.

However, the existing proposal has a number of limitations. If the requester is denied access to a single quad pattern they are denied access to the entire query. An alternative approach would be to rewrite the query to filter out the data which has been restricted, thus allowing partial results to be returned to the requester. Also, as only basic graph pattern matching is supported, in its current form the solution presented is not able to cater for complex SPARQL 1.1 queries. Therefore, additional analysis is required in order to determine a query rewriting strategy for complex SPARQL queries, that include aggregates, negation, subqueries and property path.

Chapter 6

Enforcing Access Control via Query Rewriting

The term Linked Data Web (LDW) is used to describe a World Wide Web where data is directly linked with other relevant data using machine-accessible formats. Many of the technologies required to realise the LDW vision are already in place, for example RDF, RDFS, OWL and SPARQL (O'Hara et al., 2013; Heath and Bizer, 2011). Although the LDW has seen considerable growth in recent years, the focus continues to be on linking public data. This can partially be attributed to the fact that no formal recommendations exist for the secure querying of the LDW.

Several researchers have proposed access control frameworks for RDF, which could be applied to Linked Data. Broadly speaking, access control frameworks for RDF data enforce access control either at the data layer (Dietzold and Auer, 2006; Muhleisen et al., 2010), the query layer (Abel et al., 2007; Franzoni et al., 2007; Chen and Stuckenschmidt, 2010; Oulmakhzoune et al., 2010; Costabello et al., 2012a) or a combination of both (Li and Cheung, 2008). Enforcement of access control at the data layer is concerned with removing unauthorised data from the dataset. Whereas, enforcement at the query layer relies on query rewriting techniques to ensure that the query results are restricted to those that are permitted by an access control policy. As we saw in *Chapter* 3, regardless of the proposed enforcement mechanisms, in the vast majority of cases researchers conduct performance evaluations of their solutions as opposed to verifying the correctness of the security algorithms they propose.

In *Chapter* 5, we demonstrated how access control based on graph patterns can be used to grant/deny access at multiple levels of granularity. However, using the proposed enforcement strategy, if access is denied to a single graph pattern the entire query is denied. In this chapter, we demonstrate how SPARQL 1.1 queries and updates can be rewritten so that they behave as if the unauthorised data is not present in the dataset. In order to verify the effectiveness of our query rewriting strategy, we define a set of criteria that can be used to examine the correctness of our query rewriting algorithms. This criteria can be used to compare the results obtained when the query is executed over a dataset where all unauthorised data has been removed, and the results obtained when a query is rewritten and executed over a dataset which contains both authorised and unauthorised data. With a view to examining the efficiency of our query rewriting strategy, we automatically generate a set of queries and authorisations, based on the BSBM dataset, and use this data to compare the performance of queries executed with and without access control. Finally, we discuss how the proposed access control framework fits with the LDW infrastructure.

The query results presented throughout this chapter are based on the sample data presented in *Figure* 6.1 and *Figure* 6.2. As per previous chapters, we assume the `foaf` prefix to denote the

FOAF Vocabulary `http://xmlns.com/foaf/0.1/` [1] and the `entx` prefix to refer to our sample enterprise vocabulary `http://urq.deri.org/enterprisex#`.

The contributions of the chapter can be summarised as follows: We (i) adapt a set of criteria from relational databases and demonstrate how they can be used to ensure the correctness of access control over RDF data; (ii) propose query rewriting algorithms for the SPARQL 1.1 query and update languages; (iii) verify the correctness of our access control algorithms using the adapted correctness criteria; (iv) present the results of our performance evaluation that can serve as a benchmark for other researchers; and (v) describe an authorisation architecture for the LDW, which we call LinDAA.

The remainder of the chapter is structured as follows: *Section* 6.1, presents a set of correctness criteria, which allows for access control via query rewriting to be compared to access control via data filtering. *Section* 6.2 and *Section* 6.3 describe query rewriting strategies for the SPARQL 1.1 query and update languages respectively. In *Section* 6.4 our query rewriting algorithms are evaluated against the adapted correctness criteria. In *Section* 6.5 we discuss how together the G-FAF framework, which was introduced in *Chapter* 5, and the proposed query rewriting strategy can be used to enforce access control for the LDW. Finally, we describe the related work in *Section* 6.6 and present conclusions and directions for future work in *Section* 6.7.

6.1 Access Control Correctness Criteria

According to Wang et al. (2007) a secure query processing algorithm should be `secure`, `sound` and `maximum`. An algorithm is `secure` if it does not return information which has not been authorised. An algorithm is `sound` if it it does not return invalid results. Furthermore, an algorithm is `maximum` if it returns as much information as possible without violating the `secure` and `sound` constraints. In this section, we present the access control correctness criteria, which is used by Wang et al. (2007) to evaluate access control over relational data. We subsequently discuss how this criteria can be adapted to allow for the verification of access control over RDF data via SPARQL query rewriting.

6.1.1 Correctness Criteria for Relational Databases

Wang et al. (2007) use the following criteria to evaluate their relational access control algorithm. An algorithm is deemed `secure`, if a given query executed over two different database states, that are deemed equivalent by some access control policy, returns equivalent results. An algorithm is `sound`, if a given query constrained by an access control policy returns equivalent results, or less results, than the same query executed without access control. Whereas, an algorithm is `maximum` if there is no other query, that is semantically equivalent to the given query, which will return additional results.

[1]FOAF vocabulary Specification, http://xmlns.com/foaf/spec/.

The correctness criteria is formally defined by Wang et al. (2007) as follows:

Definition 6.1 (Query correctness criteria)
An access control policy P specifies what information a database D may disclose in answer to a query Q. A secure query processing algorithm is defined as $R = A(D, P, Q)$, where R is the result set authorised by P when Q is executed on D.

Secure. *As P defines an equivalence relation among all database states, if $A(D, P, Q)$ does not depend on information not allowed by P, then one should not be able to tell the difference among database states that are equivalent under P. It follows that, a query processing algorithm is* `secure` *if and only if:*

$$\forall_P \forall_Q \forall_D \forall_{D'} [(D \equiv_P D') \implies A(D, P, Q) = A(D', P, Q)].$$

Sound. *Let S denote a query processing algorithm without access control and let $S(D, Q)$ represent the results returned when query Q is executed on database D without access control. Given P restricts access to D, then $A(D, P, Q)$ may return less information than $S(D, Q)$ however it should not return invalid results. A query processing algorithm is* `sound` *if and only if:*

$$\forall_P \forall_Q \forall_D A(D, P, Q) \sqsubseteq S(D, Q).$$

Maximum. *A trivial algorithm which does not return any results would be both secure and sound however it would not satisfy the maximum constraint. Therefore it follows, that $A(D, P, Q)$ should return at least as much information as any R which is an acceptable answer for Q given D and P. As such, a query processing algorithm is* `maximum` *if and only if:*

$$\forall_P \forall_Q \forall_D$$
and
$$\forall_{D'} [(D \equiv_P D') \implies (R \sqsubseteq S(D', Q))]$$
we have
$$\forall_P \forall_Q \forall_D [R \sqsubseteq A(D, P, Q)]$$

6.1.2 Correctness Criteria for the SPARQL Query Language

In its current form, the correctness criteria proposed by Wang et al. (2007), is unsuitable for the verification of access control over RDF data. Although it is possible to use the `secure` and `maximum` criteria to verify that a secure query processing algorithm holds over different database states, it cannot be used to verify that the algorithm is infact secure. Also, as SPARQL queries that include `MINUS` and `FILTER NOT EXISTS` may in fact be less restrictive than the corresponding query without access control, the `sound` criteria is not valid for RDF data.

Therefore in this section, we redefine each of the criteria to cater for a comparison between the result sets returned when: (i) a query is executed against a dataset which is generated by removing the unauthorised data (`filtering`); and (ii) when the same query is updated so that unauthorised data will not be returned and subsequently executed over the unmodified dataset (`rewriting`). As RDF data does not necessarily reside in a database, the term database is taken to mean an arbitrary but fixed *RDF dataset* composed of one or more *RDF graphs*.

Assuming that we have two datasets, one that contains all of the data and another that only contains authorised data, which we will refer to as the filtered dataset. A query rewriting algorithm is deemed `secure`, if it is not possible to differentiate between the results returned when access control is enforced using a query rewriting approach and the results returned when access control is enforced using data filtering. The algorithm is `sound`, if all of the results returned by the algorithm are also present in the result set which is generated when the same query is executed, without access control, over the filtered dataset. The algorithm is `maximum` if the data returned by the algorithm is equivalent to the data returned when the query is simply executed over the filtered dataset.

First we provide a definition for a *secure RDF subgraph*:

Definition 6.2 (Secure RDF subgraph)
Let D denote a database and P an access control policy. Given D and P, let DG denote the set of quads in D where access is granted by P and DD the set of quads in D where access is denied by P. Assuming that DG is disjoint from DD, then DG is the RDF subgraph of D, which is authorised by P.

Next, we formally redefine the correctness criteria as follows:

Definition 6.3 (RDF query correctness criteria)
Given our definition for a secure RDF subgraph*, if S denotes a query processing algorithm without access control, when query Q is executed on DG the result set returned by $S(DG, Q)$ only contains authorised data.*

Let $A(D, P, Q)$ represent a secure query processing algorithm, which returns the result set authorised by P when query Q is executed on D.

Secure. *For $A(D, P, Q)$ to be deemed secure, each resource $r \in A(D, P, Q)$ must be contained in the RDF subgraph, which is accessible under P. It follows that a query processing algorithm is* `secure` *if and only if:*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_r [r \in A(D, P, Q) \implies r \in DG].$$

Sound. *To be deemed sound,* A(D,P,Q) *may return less information than $S(DG, Q)$, however it should not return invalid results. Therefore, a query processing algorithm is* `sound` *if and only if:*

$$\forall_P \forall_Q \forall_D \forall_{DG} [A(D, P, Q) \sqsubseteq S(DG, Q)].$$

Maximum. *In order to be deemed maximum the results returned by* A(D, P, Q) *should be equivalent to those returned by $S(DG, Q)$. A query processing algorithm is* `maximum` *if and only if:*

$$\forall_P \forall_Q \forall_D \forall_{DG} [A(D, P, Q) \equiv S(DG, Q)].$$

6.1.3 Correctness Criteria for the SPARQL Update Language

As our access control framework can also be used in conjunction with the SPARQL 1.1 update language, we also define correctness criteria for both the `DELETE` and the `INSERT` operations, that can be used to verify the correctness of any SPARQL 1.1 update query. Assuming that we have three datasets, one that contains all of the data, another that only contains authorised data, which we will refer to as the *filtered dataset*, and another which contains unauthorised data. We use the term *merged filtered dataset* to refer to the merge of both the filtered dataset and the unauthorised data.

A delete query processing algorithm is deemed `secure`, if it does not delete any unauthorised data. Every resource that is in the merged filtered dataset, is in the rewritten dataset. The algorithm is `sound`, if it only deletes relevant data. The rewritten dataset should not contain less resources than the merged filtered dataset. The algorithm is `maximum`, if all of the relevant data is deleted. The rewritten dataset should be equivalent to the merged filtered dataset.

An insert query processing algorithm is deemed `secure`, if all of the data contained in the rewritten dataset is also in the filtered dataset. The algorithm is `sound`, if only relevant data is inserted. The rewritten dataset should not contain more resources than the merged filtered dataset. The algorithm is `maximum`, if all of the relevant data is inserted. The rewritten dataset should be equivalent to the merged filtered dataset.

Definition 6.4 (RDF update correctness criteria)
Given our definition for a secure RDF subgraph*, if U denotes an update query processing algorithm without access control, when query Q is executed on DG, the dataset generated by $U(DG, Q)$ only contains authorised data.*

Let UD denote a secure delete processing algorithm, where $UD(D, P, Q)$, produces a new database state D', when Q is executed on the subset of D which is authorised by P.

Secure. *For $UD(D, P, Q)$ to be deemed secure, D' must contain all of the data that is present in the merged filtered dataset. It follows that a query processing algorithm is* `secure` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_r [r \in (U(DG, Q) \cup DD) \implies r \in UD(D, P, Q)].$$

Sound. *For $UD(D, P, Q)$ to be deemed sound, D' must not contain less data than the filtered dataset. It follows that a query processing algorithm is* `sound` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_{DD} [UD(D, P, Q) \sqsupseteq (U(DG, Q) \cup DD)].$$

Maximum. *In order for $UD(D, P, Q)$ to be deemed maximum D' should be equivalent to the merged filtered dataset. A query processing algorithm is* `maximum` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_{DD} [UD(D, P, Q) \equiv (U(DG, Q) \cup DD)].$$

Let UI denote a secure insert processing algorithm, where $UI(D, P, Q)$, produces a new database state D'', when Q is executed on the subset of D, which is authorised by P.

Secure. *For $UI(D, P, Q)$ to be deemed secure, all of the data in D'' must also be present in the filtered dataset. It follows that a query processing algorithm is* `secure` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_{DD} \forall_r [r \in UI(D, P, Q) \implies r \in (U(DG, Q) \cup DD)].$$

Sound. *For $UI(D, P, Q)$ to be deemed sound, D'' must not contain more data than the filtered dataset. It follows that a query processing algorithm is* `sound` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_{DD} [UI(D, P, Q) \sqsubseteq (U(DG, Q) \cup DD)].$$

Maximum. *In order for $UI(D, P, Q)$ to be deemed maximum D'' should be equivalent to the merged filtered dataset. A query processing algorithm is* `maximum` *if and only if :*

$$\forall_P \forall_Q \forall_D \forall_{DG} \forall_{DD} [UI(D, P, Q) \equiv (U(DG, Q) \cup DD)].$$

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

Figure 6.1: Employee named graph

```
entx:OrgStructure{
entx:MRyan entx:worksFor entx:JBloggs .
entx:JSmyth entx:worksFor entx:MRyan .
}
```

Figure 6.2: Organisation structure named graph

6.2 Access Control for SPARQL 1.1 Queries

In *Chapter* 5 we discussed how *RDF quad patterns* can be used to specify authorisations at different levels of granularity. The *RDF quad pattern* is an extension of the *RDF triple pattern* with optionally a variable V in the *graph* position. In addition to a standard SPARQL query, the requester submits credentials that are used to determine if they are permitted to execute the query. After the credentials are verified by an external authentication system, the authorisation algorithm checks if the *authorisation request* can be derived using the *authorisations*. If the algorithm manages to successfully derive the authorisation, access to the requested data is granted, otherwise the request is denied. If access is granted the *enforcement framework* passes the SPARQL query to the *query engine*, which in turn processes the query in the normal way and returns the query results to the *requester*.

The access control strategy presented in *Chapter* 5 has a number of limitations. Firstly, the *requester* needs to be granted access to each triple in order to be permitted to execute the query. Secondly, the current framework does not support complex queries composed of filters or subqueries. As such it is necessary to devise a strategy where access can be granted to partial query results.

- One approach is to use a filtering algorithm to generate a dataset which does not contain any prohibited data, and to subsequently execute the query against the filtered dataset.
- An alternative is to develop a query rewriting algorithm that can be used to ensure that unauthorised data is not returned by the query.

Given the potential scalability issues associated with results filtering, we adopt a query rewriting approach to access control. As we are only concerned with rewriting the query to filter out the data which has been restricted, we assume that an authorisation framework has already determined the unauthorised quads.

```
⟨?sub, SELECT, −, ⟨entx:MRyan, entx:salary, ?o, ?g⟩, E, bsbm:Admin⟩
⟨?sub, SELECT, −, ⟨entx:MRyan, entx:worksFor, ?o, ?g⟩, E, bsbm:Admin⟩
```

Figure 6.3: SPARQL query authorisations

6.2.1 SPARQL 1.1 Queries

Prior to proposing a query rewriting algorithm, we examine the different query rewriting strategies for SPARQL graph patterns, based on the correctness criteria identified in the previous section. The query results presented in this section are based on the data depicted in *Figure* 6.1 and *Figure* 6.2. In addition, we use the quad patterns, presented in *Figure* 6.3 to restrict the query results. The quad pattern, `entx:MRyan entx:salary` *?o ?g*, denies access to `May Ryan's` salary. Whereas, `entx:MRyan entx:worksFor` *?o ?g*, restricts access to information pertaining to the people that `May Ryan` works for.

As we saw in *Section* 2.2.1, the SPARQL query language supports four distinct query types (`SELECT`, `ASK`, `CONSTRUCT` and `DESCRIBE`). In each case, SPARQL graph pattern matching is used in order to determine the output of the query. Although the examples presented in this section are limited to `SELECT` queries, the proposed query rewriting strategy is effective for all four query types.

Basic Graph Patterns. When we execute *Query* 6.1 without any access restrictions the identifiers, names and the salaries of all persons are returned.

Query 6.1 (Basic graph pattern)
Given the following query:

```
SELECT ?id ?name ?salary
WHERE { GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary } }
```

The output is as follows:

?id	*?name*	*?salary*
entx:JBloggs	"Joe Bloggs"	60000
entx:MRyan	"May Ryan"	33000
entx:JSmyth	"John Smyth"	33000

However, if the requester is denied access to the salary pertaining to `entx:MRyan`, by authorisation 1 in *Figure* 6.3, we need to filter out the restricted data. A naive implementation might:

(i) replace the variables in the authorisation with the corresponding variables in the query; and

(ii) construct a `FILTER NOT EXISTS` expression from the derived quad pattern.

A filter of this nature, which is presented in *Query* 6.2, will bind the variables based on the filter clause, which will remove all of the salaries that are the same as the salary for `entx:MRyan`, instead of simply removing the salary for `entx:MRyan`. Clearly such an approach is `secure` and `sound`, however it fails to meet the `maximum` correctness criteria.

Query 6.2 (Basic graph pattern restricted using pattern)
Given the following query:

```
SELECT ?id ?name ?salary
WHERE { GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary
FILTER NOT EXISTS {GRAPH entx:EmployeeDetails {
entx:MRyan entx:salary ?salary } } } }
```

The output is as follows:

?id	*?name*	*?salary*
entx:JBloggs	"Joe Bloggs"	60000

The correct approach would be rewrite the filter so that it eliminates the pattern specified in the authorisation from the result set. For each graph pattern group:

(i) A `FILTER NOT EXISTS` expression is generated for each quad in the graph pattern group that matches an authorisation. If the named graph in the query is a variable and the named graph in the authorisation is a constant, then a new graph pattern group is constructed using the named graph from the authorisation and the graph pattern from the query. Otherwise the unchanged graph pattern group is added to the filter.

(ii) The constants in the subject, predicate and object positions of the authorisation are bound to the variables in the query using a `FILTER =` expression. If multiple bindings exists the `FILTER` is generated using the conjunction of the bindings.

(iii) Finally, the `FILTER NOT EXISTS` is added to the relevant graph pattern group.

The expanded SPARQL query (*see Query* 6.3) limits the result set to the identities, names and salaries of authorised persons.

Query 6.3 (Basic graph pattern restricted using binding)
Given the following query:

```
SELECT ?id ?name ?salary
WHERE { GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary
FILTER NOT EXISTS {GRAPH entx:EmployeeDetails {
?id foaf:name ?name. ?id entx:salary ?salary
FILTER ( ?id = entx:MRyan ) } } } }
```

The output is as follows:

?id	*?name*	*?salary*
entx:JBloggs	"Joe Bloggs"	60000
entx:JSmyth	"John Smyth"	33000

Aggregates. Aggregates are functions that are applied to groups of solutions, for example `COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT` and `SAMPLE`. Although in the following example we use `COUNT` and `AVG`, the proposed query rewriting strategy can also be used with `SUM, MIN, MAX, GROUP_CONCAT` and `SAMPLE`. A query which returns the average salary of all `foaf:persons` is presented in *Query* 6.4. When this query is run over our sample data the average salary returned is `42000`.

Query 6.4 (Aggregates)
Given the following query:

```
SELECT ( COUNT(?id) AS ?numEmployees ) ( AVG(?salary) AS ?avgSalary )
WHERE { GRAPH ?g {
?id rdf:type foaf:Person . ?id entx:salary ?salary} }
```

The output is as follows:

?numEmployees	***?avgSalary***
3	42000

As before when access is restricted using authorisation 1 in *Figure* 6.3, we construct the `FILTER NOT EXISTS` expression and add it to each graph pattern group containing data which should be restricted (*see Query* 6.5). As the salary for `entx:MRyan` is not accessible, the query returns `46500` which is the average of the two remaining salaries.

Query 6.5 (Aggregates restricted with binding)
Given the following query:

```
SELECT ( COUNT(?id) AS ?numEmployees ) ( AVG(?salary) AS ?avgSalary )
WHERE { GRAPH ?g {
?id rdf:type foaf:Person . ?id entx:salary ?salary
FILTER NOT EXISTS { GRAPH ?g {
?id rdf:type foaf:Person . ?id entx:salary ?salary
FILTER ( ?id = entx:MRyan ) } } } }
```

The output is as follows:

?numEmployees	***?avgSalary***
2	46500

Subqueries and Filters. In SPARQL 1.1, negation can be achieved by filtering query results using `FILTER EXISTS`, `FILTER NOT EXISTS` or `MINUS` expressions. Although subqueries are not classified under negation, such queries are used to limit the result set, based on the results of an embedded query. The following queries are constructed using an inner `SELECT` query, however the query rewriting strategy is the same for queries that contain `FILTER EXISTS`, `FILTER NOT EXISTS` and `MINUS` expressions. *Query* 6.6 returns the names of all people and the names of the people that they work for.

Query 6.6 (Subqueries)
Given the following query:

```
SELECT DISTINCT  ?employee ?manager
WHERE { GRAPH ?g {
?x foaf:name ?employee . ?y foaf:name ?manager
{ SELECT ?x ?y WHERE { GRAPH ?g { ?x entx:worksFor ?y } } } } }
```

The output is as follows:

?employee	***?manager***
"John Smyth"	"May Ryan"
"May Ryan"	"Joe Bloggs"

If authorisation 2 in *Figure* 6.3 denies access to a quad in the outer query, we construct the `FILTER NOT EXISTS` expression and add it to each graph pattern group containing data which should be restricted. As the authorisation also matches a quad in the inner query we also add a `FILTER NOT EXISTS` to the relevant graph pattern group in the subquery or filter (*see Query* 6.7). Such an approach results in the filtering of `May Ryan` and her manager from the result set.

Query 6.7 (Subqueries restricted with binding)
Given the following query:

```
SELECT DISTINCT  ?employee ?manager
WHERE { GRAPH ?g {
?x foaf:name ?employee . ?y foaf:name ?manager
{ SELECT ?x ?y WHERE { GRAPH ?g { ?x entx:worksFor ?y
FILTER NOT EXISTS {
GRAPH ?g { ?x entx:worksFor ?y
FILTER ( ?x = entx:MRyan ) } } } } }
 } }
```

The output is as follows:

?employee	*?manager*
"John Smyth"	"May Ryan"

Property Paths. A property path is used to match a path of arbitrary length between two graph nodes. Using property paths it is possible to devise a query (*see Query* 6.8) which returns details of all employees, that are either directly or indirectly connected using the `entx:worksFor` property. Using this query we can see that `entx:JSmyth` works for `entx:MRyan` who in turn works for `entx:JBloggs`.

Query 6.8 (Property paths)
Given the following query:

```
SELECT DISTINCT ?employee ?manager
WHERE { GRAPH ?g1  { ?employee  entx:worksFor+ ?manager } }
```

The output is as follows:

?employee	*?manager*
entx:JSmyth	entx:MRyan
entx:MRyan	entx:JBloggs
entx:JSmyth	entx:JBloggs

Given authorisation 2 in *Figure* 6.3, which denies access to information relating to the persons that `entx:MRyan` works for, it is possible to employ a query rewriting strategy to filter out the unauthorised data. As per basic graph patterns, we generate a `FILTER NOT EXISTS` expression and add it to the query (*see Query* 6.9). As a result it is not possible to see who `entx:MRyan` works for. However, such a rewriting strategy is neither secure nor sound, as it is still possible to see that `entx:JSmyth` works for `entx:JBloggs`.

Query 6.9 (Property paths with binding)
Given the following query:

```
SELECT DISTINCT ?employee  ?manager
WHERE { GRAPH ?g1 { ?employee entx:worksFor+ ?manager
FILTER NOT EXISTS {
GRAPH ?g1 { ?employee entx:worksFor+ ?manager
FILTER ( ?employee  = entx:MRyan ) } } } }
```

The output is as follows:

?employee	***?manager***
entx:JSmyth	entx:MRyan
entx:JSmyth	entx:JBloggs

An alternative strategy is to removed the `FILTER =` binding, however this would result in no bindings being returned (*see Query* 6.10), thus failing to meet the maximum correctness criteria.

Query 6.10 (Property paths without binding)
Given the following query:

```
SELECT DISTINCT ?employee  ?manager
WHERE { GRAPH ?g1 { ?employee entx:worksFor+ ?manager
FILTER NOT EXISTS {
GRAPH ?g1 { ?employee entx:worksFor+ ?manager } } } }
```

The output is as follows:

?employee	***?manager***

In the case of property paths, when a binding exists for the *subject* a `FILTER NOT EXISTS` expression does not restrict access to the path data. Further analysis is thus required in order to determine an appropriate rewriting strategy for these corner cases.

6.2.2 Query Rewriting Algorithm

Based on the query rewriting strategies presented in the previous section, we propose a query rewriting algorithm, which ensures that only authorised data is returned by SPARQL 1.1 queries (*see Algorithm* 6.1). The algorithm takes as input a query, and a set of quads that need to be filtered out of the query results, and checks each of the SPARQL graph patterns recursively.

(i) If any of the graph patterns in the outer query matches any of the unauthorised quads, a `FILTER NOT EXISTS` element is generated. If the named graph in the query is a variable and the named graph in the authorisation is a constant, then a new graph pattern group is constructed using the named graph from the authorisation and the graph pattern from the query. Otherwise the unchanged graph pattern group is added to the filter. In addition, the constants in the subject, predicate and object positions of the authorisation are bound to the variables in the query using `FILTER =` expression. If multiple bindings exists the `FILTER` is generated using the conjunction of the bindings.

(ii) If any of the graph patterns in an inner `SELECT`, `EXISTS FILTER`, `NOT EXISTS FILTER` or a `MINUS` matches any of the unauthorised quads, a `FILTER NOT EXISTS` element is generated as described above and subsequently added to the relevant graph pattern group in the `SELECT`, `EXISTS FILTER`, `NOT EXISTS FILTER` or the `MINUS` expression.

```
Algorithm rewriteQuery
    Data: query, authPatterns
    Result: query
    for GraphPattern gp in query do
        newgp = checkGP(gp, authPatterns)
        if gp != newgp then
            replace(gp, newgp)
        end
    end
return query

Procedure checkGP
    Data: gp, authPatterns
    Result: gp
    for Element subElement in gp do
        if (subElement instanceOf Subquery) || (subElement instanceOf ExistsFilter) ||
        (subElement instanceOf NotExistsFilter) || (subElement instanceOf
        ElementMinus) then
            for GraphPattern qpSub in subElement do
                checkGP(qpSub, authPatterns)
            end
        end
    end
    gp = checkAuth(gp, authPatterns)
return gp

Procedure checkAuth
    Data: gp, authPatterns, negFilter
    Result: notExistsFilter
    for QuadPattern quad in gp do
        for Quad auth in authPatterns do
            if match(auth, quad) then
                NotExistsFilter filter = new NotExistsFilter(auth, quad)
                gp.add(filter)
            end
        end
    end
return gp
```

Algorithm 6.1: Query rewriting algorithm

6.3 Access Control for SPARQL 1.1 Updates

SPARQL 1.1 update caters for a number of update operations (`CLEAR`, `LOAD`, `INSERT DATA`, `DELETE DATA` and `DELETE/INSERT`) and a number of graph management operations (`CREATE`, `DROP`, `MOVE`, `COPY` and `ADD`). In this section, we examine several of scenarios where access to graph data is partially restricted. In the examples that follow, we use the authorisations presented in *Figure* 6.4, to restrict access to SPARQL update operations, over the sample data presented in *Figure* 6.1 and *Figure* 6.2. The quad pattern, `entx:JBloggs` *?p ?o ?g* denies access to all information pertaining to `entx:JBloggs`. Whereas, *?s* `entx:salary` *?o* `entx:EmployeeDetails` restricts access to `entx:salary` information in the `entx:EmployeeDetails` graph.

```
⟨?sub, ?oper, −, ⟨ entx:JBloggs, ?p, ?o, ?g⟩, E, bsbm:Admin⟩
⟨?sub, ?oper, −, ⟨ ?s, entx:salary, ?o, entx:EmployeeDetails⟩, E, bsbm:Admin⟩
```

Figure 6.4: SPARQL update authorisations

6.3.1 SPARQL 1.1 Updates

For SPARQL update operations, three distinct query rewriting strategies are required. The `DELETE/INSERT` operation uses graph patterns in order to determine the data which should be inserted, deleted or updated. Therefore, the query rewriting strategy proposed in the previous section can be used to filter out unauthorised data. Given, that the `INSERT DATA` and `DELETE DATA` operations are used to insert and delete specific quads, any unauthorised quads need to be removed from the query. Finally, given the `CLEAR`, `DROP`, `ADD`, `LOAD`, `COPY`, `MOVE` and `CREATE` operations work at the graph level, when the requester does not have access to the entire graph these queries need to be rewritten so that they operate at the triple level. In the case of update queries there are two possible options:

(1) The system should inform the requester that the query cannot be completed and provide a list of the triples that cannot be deleted, inserted etc.

(2) The system should behave as if the unauthorised data is not present.

If we adopt the first option, the requester will be aware that data exists which they do not have access to, and could potentially infer unauthorised information by issuing one or more additional queries. As a result we adopt the second option. For a discussion on explanations and the potential issues see *Section* 7.4.

DELETE, INSERT, DELETE/INSERT. Any quads in the query `WHERE` clause that match an authorisation, which denies access for that requester and the `INSERT` operation, the `DELETE` operation or both should be filtered out of the query results. As per the query rewriting strategy proposed for the SPARQL query language (*Section* 6.2.1): When no access control is present, as long as the query is well formed (even if the data specified in the query does not exist) the query simply returns success.

Query 6.11 is used to delete all information pertaining to `entx:JBloggs` from the employee graph. If authorisation 2 in *Figure* 6.4 is used to prohibit the requester from deleting salary information from the `entx:EmployeeDetails` graph, we need to filter the restricted data from the query.

Query 6.11 (DELETE/INSERT authorised data)
Given the following query:

```
WITH  entx:EmployeeDetails
DELETE { ?s ?p ?o }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER (?s = entx:JBloggs ) } }
```

continued overleaf ->

If authorisation 2 in Figure *6.4 denies access to all* `salary` *information the query is rewritten as follows:*

```
WITH entx:EmployeeDetails
DELETE { ?s ?p ?o }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER (?s = entx:JBloggs )
FILTER NOT EXISTS {
GRAPH ?g {?s ?p ?o
FILTER (?p = entx:salary ) } } } }
```

After the rewritten query is executed over the dataset presented in Figure *6.1, the new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs entx:salary 60000 .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

DELETE DATA. Any quads in the query that match an authorisation quad pattern, which denies access for that requester and the `DELETE DATA` operation, should be removed from the query. The query presented in *Query* 6.12 is used to delete all data relating to `Joe Bloggs` and `May Ryan` from the `entx:EmployeeDetails` graph.

Query 6.12 (DELETE authorised data)
Given the following query:

```
DELETE
WHERE { GRAPH entx:EmployeeDetails {
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
} }
```

continued overleaf ->

If authorisation 1 in Figure *6.4 is used to prohibit the requester from deleting information pertaining to* `entx:JBloggs`*, the query is rewritten as follows:*

```
DELETE
WHERE { GRAPH entx:EmployeeDetails {
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
} }
```

After the rewritten query is executed over the dataset presented in Figure *6.1, the new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

INSERT DATA. When it comes to the `INSERT DATA` operation, any quads in the query that match an authorisation quad pattern, which denies access for that requester and the `INSERT DATA` operation, should be removed from the query. The query presented in *Query* 6.13 is used to insert data relating to `Mike Murphy` into the `entx:EmployeeDetails` graph.

Query 6.13 (INSERT authorised data)
Given the following query:

```
INSERT DATA
{ GRAPH entx:EmployeeDetails {
entx:MMurphy rdf:type foaf:Person .
entx:MMurphy foaf:name "Mike Murphy" .
entx:MMurphy entx:salary 45000 } }
```

If authorisation 2 in Figure *6.4 is used to prohibit the requester from inserting salary information into the* `entx:EmployeeDetails` *graph, the query is rewritten as follows:*

```
INSERT DATA
{ GRAPH entx:EmployeeDetails {
entx:MMurphy rdf:type foaf:Person .
entx:MMurphy foaf:name "Mike Murphy" } }
```

continued overleaf ->

After the rewritten query is executed over the dataset presented in Figure *6.1. The new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
entx:MMurphy rdf:type foaf:Person .
entx:MMurphy foaf:name "Mike Murphy" .
}
```

CLEAR. The `CLEAR` operation removes all of the data from the target graph. When the requester does not have access to the entire graph, the `DELETE` operation can be used to clear the authorised data from the target graph. *Query* 6.14 is used to demonstrate how the `CLEAR` operation can be represented using a `DELETE` operation.

Query 6.14 (CLEAR authorised data)
Given the following query:

```
CLEAR GRAPH entx:EmployeeDetails
```

If authorisation 1 in Figure *6.4 is used to prohibit the requester from clearing information pertaining to* `entx:JBloggs`*, the query is rewritten as follows:*

```
DELETE { GRAPH entx:EmployeeDetails { ?s ?p ?o } }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER NOT EXISTS {
GRAPH entx:EmployeeDetails {?s ?p ?o
FILTER (?s = entx:JBloggs ) } } } }
```

After the rewritten query is executed over the dataset presented in Figure *6.1, the new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
}
```

DROP. The `DROP` operation removes all of the data from the target graph, and afterwards deletes the target graph. When access control is present the `DELETE` operation can be used to delete authorised data from the target graph. When access is restricted to some of the graph data, the `DROP` operation would be equivalent to the `CLEAR` operation (i.e. the graph cannot be deleted as it contains data), and thus the same rewriting strategy applies.

ADD. The `ADD` operation, copies data from a source graph to a destination graph. This operation can be supported by using the `INSERT` operation to insert this data into the destination graph. However, this operation should only be permitted if the requester has `INSERT` permissions for all of the data that is returned by the select query. The query presented in *Query* 6.15 is used to demonstrate how the `ADD` operation can be catered for using the `INSERT` operation.

Query 6.15 (ADD authorised data)
Given the following query:

```
ADD GRAPH entx:EmployeeDetails TO GRAPH entx:Management
```

If authorisation 1 in Figure *6.4 is used to prohibit the requester from adding information pertaining to* `entx:JBloggs`*, the query is rewritten as follows:*

```
INSERT { GRAPH entx:Management { ?s ?p ?o } }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER NOT EXISTS {
GRAPH entx:EmployeeDetails {?s ?p ?o
FILTER (?s = entx:JBloggs ) } } } }
```

continued overleaf ->

After the rewritten query is executed over the dataset presented in Figure *6.1. The new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}

entx:Management{
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

LOAD. The `LOAD` operation copies data from a source URI to a destination graph. The `INSERT` operation can be used to insert this data into the destination graph. However, like the `ADD` this operation should only be permitted if the requester has `INSERT` permissions for all of the data that is returned by the select query. Given the `LOAD` operation would be equivalent to the `ADD` operation, the same rewriting strategy applies.

COPY. The `COPY` operation removes all data from the destination graph, and copies data from a source graph to a destination graph. This can be done by using `DELETE` to delete all of the authorised data from the destination graph and using `INSERT` to insert this data into the destination graph. However, given there is a dependency between the queries, they should be wrapped in a transaction in order to ensure that either both, or neither, succeed. The query presented in *Query* 6.16 is used to demonstrate how the `COPY` operation can be catered for using a combination of `DELETE` and `INSERT`.

Query 6.16 (COPY authorised data)
Given the following query:

```
COPY GRAPH entx:EmployeeDetails TO GRAPH entx:Management
```

continued overleaf ->

Given authorisation 1 in Figure *6.4 is used to prohibit the requester from copying information pertaining to* `entx:JBloggs`*, the query is rewritten as follows:*

```
DELETE { GRAPH entx:Management { ?s ?p ?o } }
WHERE { GRAPH entx:Management { ?s ?p ?o } }

INSERT { GRAPH entx:Management { ?s ?p ?o } }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER NOT EXISTS {
GRAPH entx:EmployeeDetails {?s ?p ?o
FILTER (?s = entx:JBloggs ) } } } }
```

After the rewritten query is executed over the dataset presented in Figure *6.1, the new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}

entx:Management{
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

MOVE. The `MOVE` operation moves all of the data from the source graph into the destination graph, and afterwards deletes the source graph. When access control is present the `INSERT` operation can be used to insert authorised data into the destination graph, and the `DELETE` operation can be used to delete authorised data from the source graph. However, given there is a dependency between the queries, they should be wrapped in a transaction in order to ensure that either both, or neither, succeed. The query presented in *Query* 6.17 is used to demonstrate how the `MOVE` operation can be catered for using a combination of `DELETE` and `INSERT`.

Query 6.17 (MOVE authorised data)
Given the following query:

```
MOVE GRAPH entx:EmployeeDetails TO GRAPH entx:Management
```

If authorisation 1 in Figure *6.4 is used to prohibit the requester from moving information pertaining to* entx:JBloggs*, the query is rewritten as follows:*

```
DELETE { GRAPH entx:Management { ?s ?p ?o } }
WHERE { GRAPH entx:Management { ?s ?p ?o } }

INSERT { GRAPH entx:Management { ?s ?p ?o } }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER NOT EXISTS {
GRAPH entx:EmployeeDetails {?s ?p ?o
FILTER (?s = entx:JBloggs ) } } } }

DELETE { GRAPH entx:EmployeeDetails { ?s ?p ?o } }
WHERE { GRAPH entx:EmployeeDetails {
?s ?p ?o
FILTER NOT EXISTS {
GRAPH entx:EmployeeDetails {?s ?p ?o
FILTER (?s = entx:JBloggs ) } } } }
```

After the rewritten query is executed over the dataset presented in Figure *6.1, the new state of the dataset is as follows:*

```
entx:EmployeeDetails{
entx:JBloggs rdf:type foaf:Person.
entx:JBloggs foaf:name "Joe Bloggs" .
entx:JBloggs entx:salary 60000 .
entx:JBloggs foaf:phone "111-1111" .
}

entx:Management{
entx:MRyan rdf:type foaf:Person .
entx:MRyan foaf:name "May Ryan" .
entx:MRyan entx:salary 33000 .
entx:MRyan foaf:phone "222-2222" .
entx:JSmyth rdf:type foaf:Person .
entx:JSmyth foaf:name "John Smyth" .
entx:JSmyth entx:salary 33000 .
entx:JSmyth foaf:phone "333-3333" .
}
```

6.3.2 Update Query Rewriting Algorithm

Based on the query rewriting strategies presented in the previous section, we propose a secure update query rewriting algorithm, which ensures that only authorised data is inserted and deleted (*Algorithm* 6.2). The algorithm takes as input a query and a set of negative authorisations that relate to that query. In order to cater for the different query types, a number of rewriting strategies are required:

```
Algorithm rewriteUpdate
    Data: query, authPatterns
    Result: newQueries
    if query.type == DELETE/INSERT then
        query = rewriteQuery(query, authPatterns)
        newQueries.add(query)
    end
    else if ( query.type == DELETE DATA ) || ( query.type == INSERT DATA )
    then
        for QuadPattern q in query do
            if q in authPatterns then
                query = query.remove(q)
            end
        end
        newQueries.add(query)
    end
    else if ( query.type == CLEAR ) || ( query.type == DROP ) then
        deleteQuery = deleteFromGraph(query.target, authPatterns)
        newQueries.add(deleteQuery)
    end
    else if ( query.type == ADD ) || ( query.type == LOAD ) ||
    ( query.type ==COPY ) || ( query.type == MOVE ) then
        if query.type ==COPY || query.type == MOVE then
            deleteQuery = deleteFromGraph(query.destination, authPatterns)
            newQueries.add(deleteQuery)
        end
        insertQuery = new InsertQuery(query.destination)
        for QuadPattern q in authPatterns do
            insertQuery = addNotExistsFilter(insertQuery, q)
        end
        newQueries.add(insertQuery)
        if query.type == MOVE then
            deleteQuery = deleteFromGraph(query.source, authPatterns)
            newQueries.add(deleteQuery)
        end
    end
return newQueries

Procedure deleteFromGraph
    Data: graph, authPatterns
    Result: query
    query = new DeleteQuery(graph)
    for QuadPattern q in authPatterns do
        query = addNotExistsFilter(query, q)
    end
return query
```

Algorithm 6.2: Update rewriting algorithm

(i) `DELETE/INSERT`. The query rewriting strategy presented in *Section* 6.2.2 is used to filter out unauthorised quad patterns using a `FILTER NOT EXISTS` expression.

(ii) **`DELETE DATA` and `INSERT DATA`.** If any of the quads in the query match the unauthorised quad patterns, these quads are removed from the query.

(iii) **`CLEAR` and `DROP`.** Negative authorisations pertaining to the specified graph are added as filters to a `DELETE` query, which is used to ensure that only authorised data is removed from the graph.

(iv) **`ADD` and `LOAD`.** Negative authorisations relating to the source and the destination graphs are added as filters to an `INSERT` query, which is used to add/load only authorised data to the destination graph.

(v) **`COPY`.** A `DELETE` query is used to remove all data from the destination graph. While negative authorisations matching the source and the destination graphs are added as filters to an `INSERT` query, which is used to copy only authorised data into the destination graph.

(vi) **`MOVE`.** A `DELETE` query is used to remove all data from the destination graph. While negative authorisations associated with the source and the destination graphs, are added as filters to an `INSERT` query, which is used to move only authorised data into the destination graph. Finally a `DELETE` query is used to delete data from the source graph.

6.4 Evaluation of the Proposed Query Rewriting Strategies

In *Section* 6.1 we redefined the `secure`, `sound` and `maximum` correctness criteria proposed by Wang et al. (2007), so that access control via query rewriting can be compared against access control via data filtering. In order to evaluate our query rewriting strategy, we use this criteria, along with an unconventional approach to model checking, to perform a comparison between the two approaches.

Model checking is used to verify the correctness properties of finite-state systems. The objective being, given a model of a system, exhaustively and automatically check whether this model meets a given specification. Model checking has been successfully used to verify access control in other domains, however to date it has not been used to evaluate access control over RDF. Although we use model checking in an unconventional way.

The aim of our model checking approach is to verify the query rewriting algorithms hold, irrespective of the data, for both the SPARQL 1.1 query and update languages. Therefore, we developed an authorisation and query data generator, which automatically generates a set of authorisations from all 2^4 possible combinations (of constants and variables), for each quad in a given dataset, and which systematically generates queries for each RDF quad pattern. Given that SPARQL query results are dependent on pattern matching and filtering, in order to prove the correctness of our query rewriting strategy we only have to show it works for all possible combinations of quad patterns injected into a bounded set of queries. Both the size of the dataset and the data itself are irrelevant, therefore our model checking approach is unconventional.

6.4.1 Evaluation Setting

The benchmark system has an Intel(R) Xeon(R) CPU 8 core 2.13GHz processor, 64 GB of memory and runs Debian 6.0.3. The entire system (test data generator, query rewriting algorithms and model checking algorithms) is implemented in Java, and the query evaluation is performed over an in memory store using Jena. The Berlin SPARQL Benchmark (BSBM) dataset generator was used to generate a dataset containing 1194 quads. Authorisations and queries are auto generated from a Berlin SPARQL Benchmark (BSBM) dataset, using our test data generator. For each RDF quad in the BSBM dataset, 2^4 authorisations are generated, resulting in a total of 19104 authorisations. Although we examined twenty four query types, the `SAMPLE`, `LOAD` and the `CREATE` operations where not included in the evaluation. As `SAMPLE` returns different data each time it is executed it is not possible to compare query rewriting to results filtering. The query rewriting strategy for `LOAD` is identical to that for the `ADD` operation and no rewriting strategy is required for `CREATE` as access is either granted or denied. Therefore, for each of these authorisations twenty one queries are generated as follows:

- **Basic graph patterns.** A query is generated, which contains either one, two or three RDF quad patterns, that are randomly generated from data selected from the entire dataset.
- **Aggregates.** For `COUNT` and `GROUP_CONCAT` operations, queries are also generated from up to three RDF quad patterns, which like basic graph patterns, are randomly generated from the entire dataset. Given `SUM`, `MIN`, `MAX` and `AVG` operations are dependent on numeric data, these queries are generated from a quad pattern which matches all offers (*?s* `rdf:type bsbm:offer` *?g*) and a pattern that matches the associated delivery days (*?s* `bsbm:deliveryDays` *?o ?g*).
- **Subqueries and filters.** In the case of subqueries and filters a pattern with all variables is added to the outer query, and the inner `SELECT`, `MINUS`, `FILTER EXISTS` and `FILTER NOT EXISTS` are generated from either one, two or three RDF quad patterns, which are randomly selected from the entire dataset.
- **Delete data and insert data.** For `DELETE DATA` and `INSERT DATA`, queries are generated from one, two or three RDF quads, that are randomly selected from the entire dataset.
- **Delete, insert and delete/insert.** As per basic graph patterns, `DELETE`, `INSERT` and `DELETE/INSERT` queries are generated from graph patterns that are randomly selected from the entire dataset.
- **Graph update operations.** `CLEAR`, `DROP`, `ADD`, `COPY` and `MOVE` queries are generated for each graph appearing in the dataset.

6.4.2 Verification of SPARQL 1.1 Queries

Our verification of our query rewriting strategy for the SPARQL 1.1 query language was performed as per *Algorithm* 6.3. For each query, the relevant authorisation is retrieved and the following steps are performed:

(i) Firstly, the quad pattern is used to create a new dataset which only contains authorised data, and the query is executed against this filtered dataset.

(ii) Secondly, the quad pattern is used to rewrite the query based on the query rewriting algorithm presented in *Algorithm* 6.1, and this rewritten query is executed over the original dataset.

(iii) Finally, the results of both approaches are compared. If each resource in the rewritten result set is also present in the filtered dataset, the query rewriting strategy is `secure`. If each resource in the rewritten result set is also present in the filtered result set, the query rewriting strategy is `sound`. Whereas, if both the rewritten result set and the filtered result set are equivalent the query is deemed `maximum`.

In the case of basic graph patterns, aggregates and negation, each of the queries generated from the BSBM dataset, were deemed `secure`, `sound` and `maximum`. However, given property given a `FILTER NOT EQUALS` does not restrict access to the path data, further analysis is required in order to determine an appropriate rewriting strategy for property paths.

6.4.3 Verification of SPARQL 1.1 Updates

Our verification of our query rewriting strategy for the SPARQL 1.1 update language was performed as per *Algorithm* 6.4. As before, for each query, the relevant authorisation is retrieved and the following steps are performed:

```
Algorithm queryModelChecking
  Data: dataset, auths, queries
  for Query query in queries do
    Auth auth = getAuth(query, auths)

    Dataset filteredDS = filterDataset(dataset, auth)
    List<String> filteredResults = execute(query, auth, filteredDS)

    Query rewrittenQuery = rewriteQuery(query, auth)
    List<String> rewrittenResults = execute(rewrittenQuery, auth, dataset)

    secure = " secure "
    sound = " sound "
    maximum = " maximum "

    for String s in rewrittenResults do
      if !filteredDS.contains(s) then
        secure= " not secure "
      end
      if !filteredResults.contains(s) then
        sound= " not sound "
      end
    end

    if rewrittenResults != filteredResults then
      maximum= " not maximum "
    end
    print(query.filename + secure + sound + maximum)
  end
return
```

Algorithm 6.3: Query model checking algorithm

(i) Firstly, the quad pattern is used to create a dataset which only contains authorised data, and a dataset which only contains unauthorised data. Like the SPARQL query language, the query is executed against the dataset which contains authorised data. However, in this instance the updated dataset is returned rather than a list of projections. Both the unauthorised dataset and the updated filtered dataset are merged to form a new merged filtered dataset.

(ii) Secondly, the quad pattern is used to rewrite the query, based on the update query rewriting algorithm presented in *Algorithm* 6.2, and this rewritten query is executed over the original dataset. Again, the the updated dataset is returned rather than a list of projections.

(iii) Finally, the results of both approaches are compared. As `DELETE DATA`, `DELETE`, `CLEAR`, `DROP` and `MOVE` operations delete data from the dataset, if each of the quads in the filtered dataset are also present in the rewritten dataset the delete is deemed `secure`. Whereas, if all of the data in the merged filtered dataset is present in the rewritten dataset, the delete is deemed `sound`. Given, `INSERT DATA`, `INSERT`, `ADD`, `LOAD`, `COPY` and `MOVE` operations insert data into the dataset, if all of the data in the rewritten dataset is also in the merged filtered dataset, the insert is `secure`. Whereas, if each of the quads in the rewritten dataset are also present in the merged filtered dataset, the insert is deemed `sound`. If both the rewritten result set and the filtered result set are equivalent, both the delete and/or the insert are deemed `maximum`.

```
Algorithm updateModelChecking
    Data: dataset, auths, queries
    for Query query in queries do
        Auth auth = getAuth(query, auths)

        Dataset filteredDenyDS = filterDenyDataset(dataset, auth)
        Dataset filteredGrantDS = filterGrantDataset(dataset, auth)
        filteredGrantDS = execute(query, filteredGrantDS)
        Dataset filteredMerged = mergeDataset(filteredGrantDS, filteredDenyDS)

        Query rewrittenQ = rewriteQuery(query, quad)
        Dataset rewrittenDS = execute(rewrittenQ, auth, dataset)

        secure = " secure "
        sound = " sound "
        maximum = " maximum "

        if ((query.type == DELETE DATA) || (query.type == DELETE) ||
        (query.type == CLEAR) || (query.type == DROP) || (query.type == MOVE))
        then
            for Quad qf in filteredMerged do
                if !rewrittenDS.contains(qf) then
                    result+= " delete not secure " result+= " delete not sound "
                end
            end
        end

        if ((query.type == INSERT DATA) || (query.type == INSERT) ||
        (query.type == ADD) || (query.type == LOAD) ||
        (query.type == COPY) || (query.type == MOVE)) then
            for Quad rw in rewrittenDS do
                if !filteredMerged.contains(rw) then
                    result+= " insert not secure " result+= " insert not sound "
                end
            end
        end

        if filteredMerged != filteredResults then
            result+= " delete not maximum " result+= " insert not maximum "
        end
        print(result)
    end
return
```

Algorithm 6.4: Update model checking algorithm

As each of the update queries, that were generated from the BSBM dataset, were deemed `secure`, `sound` and `maximum`, we can conclude that the update rewriting algorithm is `secure`, `sound` and `maximum`.

6.4.4 Comparison of Query Rewriting and Filtering

In order to evaluate our query rewriting algorithms we examine the performance for each of the different query types that were generated using the BSBM dataset as described in *Section* 6.4.1.

Table 6.1: Average query processing times (ms)

	Query Execution	Query Rewriting & Query Execution	Dataset Filtering & Query Execution
`AVG`	1.26	1.49	18.06
`BGP`	1.06	1.11	17.82
`GROUP CONCAT`	1.27	1.28	18.00
`COUNT`	1.25	1.31	17.85
`MAX`	1.17	1.45	17.85
`MIN`	1.17	1.42	17.93
`SUM`	1.18	1.43	17.93
`EXISTS`	9.76	49.37	26.35
`MINUS`	4.06	74.77	20.81
`NOT EXISTS`	10.68	96.1	27.41
`SUBQUERY`	11.39	38.63	27.14
`DELETE`	1.21	1.29	17.16
`DELETE/INSERT`	1.20	1.28	17.19
`DELETE DATA`	1.01	0.35	17.10
`INSERT`	1.19	1.28	17.37
`INSERT DATA`	1.02	0.35	17.01
`CLEAR`	0.47	15.58	16.56
`DROP`	0.46	15.53	16.60
`MOVE`	0.63	33.41	16.62
`ADD`	0.68	15.38	16.72
`COPY`	0.69	16.72	16.67

For each query set we perform the following:

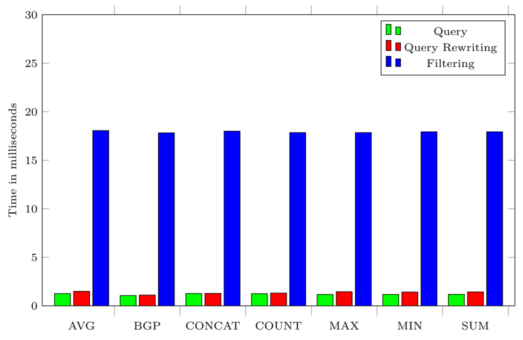
(i) **Query Execution.** We calculate the average time taken to execute the original query over the filtered dataset. This calculation does not take into consideration the time required to filter the data.

(ii) **Query Rewriting & Execution.** We calculate the average time taken to rewrite the query and subsequently execute the rewritten query over the original dataset.

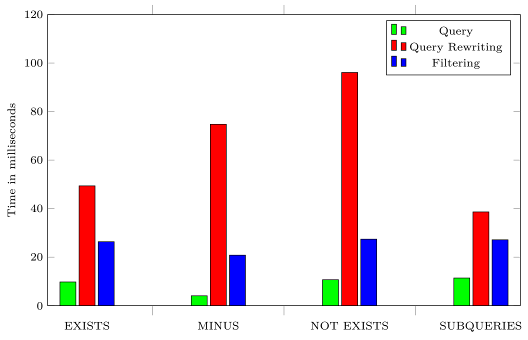
(iii) **Dataset Filtering & Query Execution.** We calculate the average time taken to remove all unauthorised data from the dataset and to execute the original query over the filtered dataset.

In *Table* 6.1 and *Figure* 6.5 we present the average query processing time for each query category. In each instance we calculate the total processing time for all 19104 queries, based on 10 separate runs, remove the fastest and the slowest total test runs, and calculate the average query processing time for an individual query. The results indicate that in the case of:

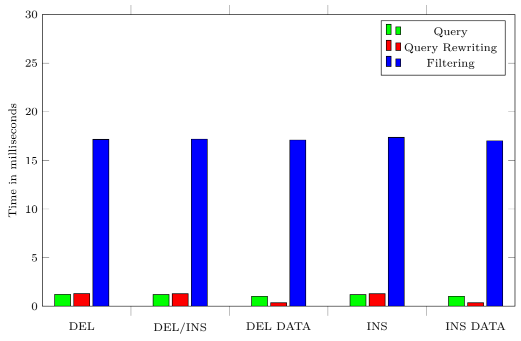
- Basic Graph Patterns (BGPs) and aggregates - there is only a small increase in processing time when the query is rewritten. However, given that the data filtering approach needs to iterate through each quad in the dataset, the processing times are substantially greater than the query rewriting approach.
- Quad update operations - Given that unauthorised data is remove from the `INSERT DATA` and `DELETE DATA` queries, such operations take less time than the respective queries without access control. Whereas, the results for the `DELETE/INSERT` operations are consistent with that of the BGPs and aggregates.



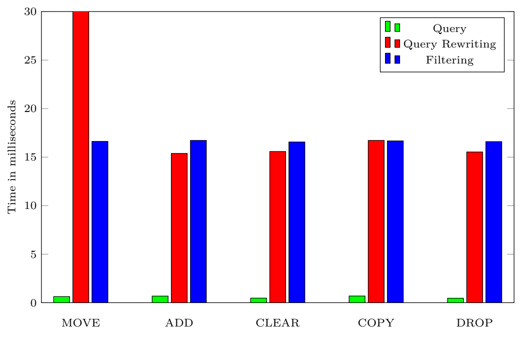
(a) BGPs and aggregates



(b) Filters and subqueries



(c) Quad update operations



(d) Graph update operations

Figure 6.5: Query processing times

- Graph update operations - we see a considerable increase in processing times. This can be attributed to the fact that when graph based operations are translated into triple based operations we construct INSERT/DELETE queries with a graph pattern matching all quads and a filter which removes all of the unauthorised data. The move operation takes twice as long as the other operations as we need two separate operations, one to `INSERT` into the destination and one to `DELETE` from the destination.
- Filters and subqueries - given that our query rewriting strategy involves adding a filter inside a filter, in the case of negation we see a severe impact in query processing times with `MINUS` and `NOT EXISTS` operations taking longer than `EXISTS` and subqueries.

Based on our evaluation, we can conclude that although it is possible to rewrite SPARQL 1.1 queries and updates so that they behave as if the unauthorised data is not present in the dataset, it is clear that there is a need for query planning in order to optimise the performance times for different query categories. As such, it is envisaged that the results presented in this chapter, will serve as a baseline that can be used to benchmark alternative access control strategies for Linked Data.

6.5 Access Control for the LDW

In *Section* 5.5.1 we examined how the G-FAF framework can be used to enforce and manage access control policies over Linked Data that is exposed via SPARQL endpoints. In this section, we extend the original architecture to allow for access to be granted to partial RDF data, and to cater for RDF exposed as web documents, embedded in HTML documents or exposed via RDB2RDF interfaces. The discussion that follows demonstrates how the extended Linked Data Authorisation Architecture (LinDAA), which is presented in *Figure* 6.6, can be used to seamlessly enforce and manage RDF data exposed as Linked Data.

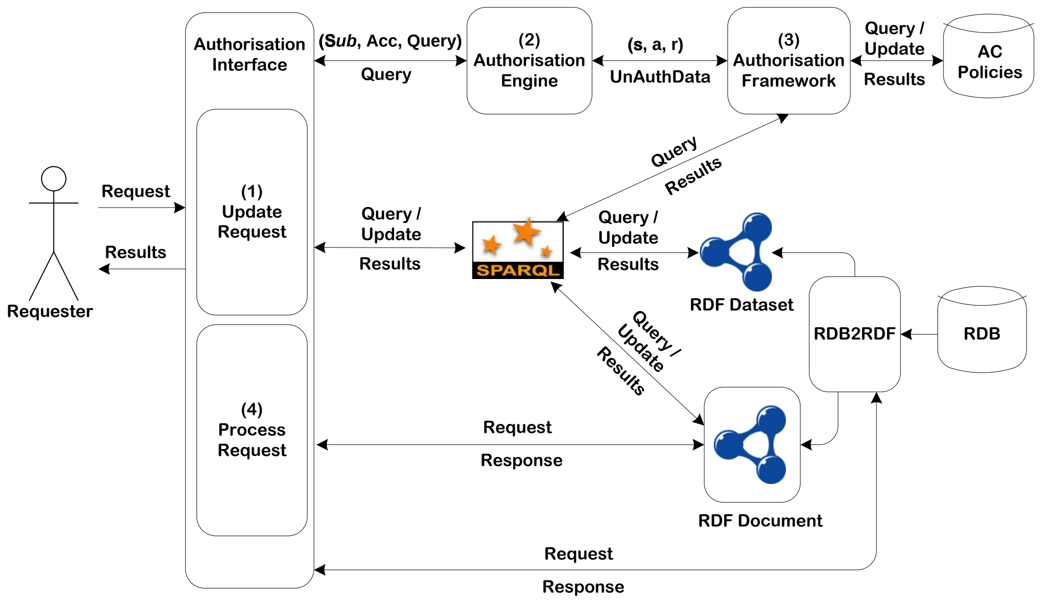


Figure 6.6: Linked Data Authorisation Architecture (LinDAA)

6.5.1 Securely Publishing Linked Data

Linked Data publishers either serve RDF data or enable RDF data to be queried using the existing web infrastructure. In *Section* 2.4.2, we presented the different strategies used to expose RDF data as Linked Data. In summary:

- RDF data that is represented in RDF stores can be exposed as Linked Data via SPARQL endpoints.
- RDF represented in static RDF documents can be served by regular web servers.
- RDF can be embedded in HTML documents using RDFa and served as per normal HTML pages.
- RDB2RDF tools can be used to translate relational data into RDF, which can be represented in an RDF store or an RDF document. Alternatively, custom interfaces and RDB2RDF mappings can be used to expose relational data as RDF.

Given that LinDAA is designed to work seamlessly with the LDW, regardless of whether the RDF data is exposed via a SPARQL endpoint, represented in an RDF document, embedded in HTML or accessed via an RDB2RDF interface, the web server needs to be configured to redirect all requests for access to the LinDAA *Authorisation Interface* which ensures that access is only permitted to authorised data.

6.5.2 Securely Consuming Linked Data

In a similar fashion to the web of documents, Linked Data can be accessed using Linked Data browsers, search engines and crawlers. In *Section* 2.4.3, we examine the different patterns, that according to Heath and Bizer (2011), are commonly used to consume Linked Data. The *Crawling Pattern* uses URI dereferencing to explore the Linked Data web and to create a local index. The *On-The-Fly-Dereferencing Pattern* also uses URIs to dereference, however in this case data is consumed at query time. Whereas, the *Query Federation Pattern* involves querying a fixed set of data sources via SPARQL endpoints. In order for LinDAA to work seamlessly with the LDW, regardless of the mechanism used to consume Linked Data, the requester needs to be able to both dereference and query the RDF data. As LinDAA also supports SPARQL 1.1 queries, a fourth pattern, which we call the *Update Pattern*, can be used to maintain RDF data.

6.5.3 The Linked Data Authorisation Architecture

In this section we discuss how the LinDAA architecture depicted in *Figure* 6.6 can be used in conjunction with SPARQL endpoints, RDF documents, RDFa and RDB2RDF interfaces. In addition to either the document request or the query, the requester must submit their credentials, which are verified by an external authentication system.

(1) Authorisation Interface (Update Request). For each request the *Authorisation Interface* generates an *Authorisation Request* of the form $\langle Sub, Query \rangle$. Where *Sub* refers to the requesters credentials and *Query* refers to the query, which is either submitted by the requester or constructed by the *Authorisation Interface.* In the case of:

- **SPARQL queries and SPARQL updates.** The SPARQL query or SPARQL update query is simply mapped to an *Authorisation Request.*
- **RDF documents.** All of the data residing in the RDF Document is loaded into a temporary RDF dataset and a new SPARQL query, which is used to return all data from the temporary dataset is constructed and mapped to an *Authorisation Request.*
- **RDFa documents.** A distiller or parser is used to extract the RDF data and to store it in a temporary RDF dataset and a new SPARQL query, which is used to return all data from the temporary dataset is constructed and mapped to an *Authorisation Request.*
- **RDB2RDF.** An RDB2RDF interface is used to retrieve the RDF data and to load it in a temporary RDF dataset and a new SPARQL query, which is used to return all data from the temporary dataset is constructed and mapped to an *Authorisation Request.*

In each instance the *Authorisation Request* is subsequently passed to the *Authorisation Engine.*

(2) Authorisation Engine. For each graph pattern present in the query the *Authorisation Engine*, constructs a tuple $\langle Sub, Acc, Res \rangle$, where *Sub* refers to the authorisation subject, *Acc* refers to the access right based on the type of query and *Res* refers to the graph pattern. For each tuple the *Authorisation Engine* calls the *Authorisation Framework* in order to determine what data needs to be filtered out of the query.

(3) Authorisation Framework. The authorisation algorithm checks if the *Authorisation Request* can be derived using the *Authorisations* and the *Conflict Resolution Policies.* If the authorisation framework denies access, the algorithm returns a list of the quad patterns that are unauthorised. This possibly empty list of unauthorised quad patterns is returned to the *Authorisation Engine.* For additional details on the *Authorisation Framework* see *Section* 5.4.

(2) Authorisation Engine. When the *Authorisation Framework* has checked all graph patterns in the query, the *Authorisation Engine* proceeds as follows:

- **Access granted.** If the list of unauthorised quad patterns is empty, then access is granted to the entire graph pattern and no action is required.
- **Access denied.** If the list contains all of the quad patterns, then access is denied to the entire graph pattern and the graph pattern is removed.

- **Access partially granted.** If access to some graph patterns is granted and access to other graph patterns is denied, then the query is rewritten according to either the SPARQL query rewriting algorithm (*Algorithm* 6.1) or the SPARQL update query rewriting algorithm (*Algorithm* 6.2).

Once all of the quad patterns have been checked then the rewritten query is returned to the *Authorisation Interface*:

- **Access granted.** If access is granted to the entire query, then the original query is returned.
- **Access denied.** If access is denied to the entire query, then an empty query is returned.
- **Access partially granted.** If access was partially granted to one or more graph patterns, then the rewritten query is returned.

(4) Authorisation Interface (Process Request). Once the query is returned by the *Authorisation Engine*. The *Authorisation Interface* proceeds as follows:

- **Access granted.** If access is granted, then depending on the original request the *Authorisation Interface*, proceeds as follows:
  - **SPARQL queries and updates.** The query is executed against the SPARQL endpoint and the results are returned to the requester;
  - **RDF documents and RDFa documents.** The RDF or HTML document is returned to the requester; or
  - **RDB2RDF.** The data returned from the RDB2RDF interface is returned to the requester.
- **Access denied.** If access is denied, then depending on the original request the *Authorisation Interface*, proceeds as follows:
  - **SPARQL queries and updates.** The interface returns the same results as you would get if the data was not present in the RDF dataset;
  - **RDF documents.** The interface returns an empty document;
  - **RDFa document.** The RDFa is removed from the HTML document and the resulting HTML document is returned to the requester;
  - **RDB2RDF.** The interface returns the same results as you would get if the data was not present in the relational database.
- **Access partially granted.** If the query was rewritten, then depending on the original request the *Authorisation Interface*, proceeds as follows:
  - **SPARQL queries and updates.** The rewritten query is executed against the SPARQL endpoint and the results are returned to the requester;
  - **RDF documents.** The rewritten query is executed against the temporary RDF dataset, and a new document is generated from the results and returned to the requester;
  - **RDFa documents.** The rewritten query is executed against the temporary RDF dataset, and any RDF data that is not present in the result set is removed from the HTML. Finally the updated HTML document is returned to the requester;
  - **RDB2RDF.** The rewritten query is executed against the temporary RDF dataset, and a new document is generated from the results and returned to the requester.

In order to evaluate LinDAA, we plan to examine both the performance and the correctness of the access control policies, that are specified on top of Linked Data (exposed as RDF documents, embedded in HTML documents, exposed via SPARQL endpoints and accessible using RDB2RDF technology). Additionally, in order to evaluate the expressiveness of our framework we plan to choose a representative set of case studies, based on use cases that are commonly adopted by Semantic Web researchers, and examine the type of authorisation and rules that are required in each scenario.

6.6 Related Work

Broadly speaking access control frameworks for RDF data enforce access control either at the data layer (Dietzold and Auer, 2006; Muhleisen et al., 2010), the query layer (Abel et al., 2007; Franzoni et al., 2007; Chen and Stuckenschmidt, 2010; Oulmakhzoune et al., 2010; Costabello et al., 2012a) or a combination of both Li and Cheung (2008). When access control is enforced at the data layer, a filtering mechanism is used to generate a view of the data based on a given access control policy (which only contains authorised data). Whereas, when access control is enforced at the query layer, query rewriting techniques are used to limit access to data, based on a given access control policy.

Both Dietzold and Auer (2006) and Muhleisen et al. (2010), adopt a filtering approach to access control. Dietzold and Auer (2006) propose access control policy specification at multiple levels of granularity (*triples*, *classes* and *properties*). Authorisations are used to assign users access to RDF data using filters (SPARQL `CONSTRUCT` queries). When a requester submits a query, a virtual model is created based on the matched authorisations. The query is subsequently executed against the virtual model, which only contains data the requester is authorised to access. Muhleisen et al. (2010) allow access control policies to be specified for *triple patterns*, *resources* or *instances*. When a requester submits a query, the system uses their WebID to determine the data instances the user is permitted to access and generates a temporary named graph containing authorised data. The requesters query is subsequently executed against the temporary named graph and the results are returned to the user.

Abel et al. (2007) and Franzoni et al. (2007) demonstrate how contextual conditions associated with the requester, the resource or the environment can be injected into the query. Abel et al. (2007) specify authorisations in terms of sets of *contextual predicates*, *path expressions* and *boolean expressions*. The authors propose a query rewriting strategy which constructs bindings for authorisation path expressions and contextual predicates. For positive authorisations, the bindings are appended to the query `WHERE` clause. For negative authorisations, the bindings are added to a `MINUS` clause, which in turn is appended to the query. Whereas, Franzoni et al. (2007) propose a query rewriting strategy which is used to grant/deny access to ontology instances, based on contextual information pertaining to the user or the environment. Authorisations are used to associate properties in the form of path expressions, attributes and filter conditions with resources. Like Abel et al. (2007) bindings are generated for the path expressions and both the path expressions and the bindings are added to the query. Costabello et al. (2012a) also use contextual data to restrict access to RDF. However, the proposed query rewriting strategy restricts access to named graphs as opposed to specific classes, properties and instances.

Like us, Chen and Stuckenschmidt (2010) and Oulmakhzoune et al. (2010) use filters to bind/unbind query solutions based on access control policies that are associated with classes, properties and individuals. Chen and Stuckenschmidt (2010) present a query rewriting strategy which can be used to restrict access to data represented using ontologies. Access control policies are used to deny access to specific *individuals*, or to grant/deny access to instances associated with a given *class* or *property*. The authors propose a query rewriting strategy, which uses `FILTER` expressions to bind or unbind variables specified in the query to instances, properties and classes specified in the authorisations. When access is prohibited to *properties* or *classes*, the matching triple patterns are made `OPTIONAL`, which ensures that all authorised information is returned. Oulmakhzoune et al. (2010) also cater for both positive and negative authorisations. In the presented modelling, authorisations are composed of sets of filters that are associated with simple conditions or involved conditions. The authors use the term simple condition to refer to an authorisation, which either permits/denies access to one or more triple patterns, and the term involved condition to refer to authorisations that permit/deny access for a given *predicate*. In the case of simple conditions, when access is permitted to a single triple pattern, no action is required. Whereas, when access is denied to a single triple pattern, the triple pattern is deleted. Like Chen and Stuckenschmidt (2010) when access is permitted/denied to a basic graph pattern, the pattern is converted to an `OPTIONAL` pattern, and the authorisation `FILTER` expression is added to the query. In the case of involved conditions, when access is permitted the `FILTER` expression, which is associated with the predicate is added to the query. If the triple pattern is not already part of the query, it is also added. When access is prohibited, the `FILTER` expression associated with the predicate is added, and if the object associated with the given predicate is a variable, a `!BOUND` expression is added for the object variable. Although both Chen and Stuckenschmidt (2010) and Oulmakhzoune et al. (2010) propose query rewriting strategies for SPARQL queries, the authors focus specifically on `SELECT` queries and no special consideration is given to complex SPARQL queries that include subqueries or negation. In addition, no specific consideration is given to named graphs.

Li and Cheung (2008) propose a query rewriting strategy for views generated from ontological relations. The proposed query rewriting strategy involves expanding the view concepts to include implicit concepts, retrieving both explicit and implicit access control policies and adding range and instance restrictions, associated with the matched authorisation, to the view query. Like us, the authors take into account both explicit and implicit access control policies. However, they propose a combined query rewriting and filtering access control strategy, as opposed to a query rewriting strategy for the SPARQL 1.1 query and update languages in our case.

When it comes to the formal evaluation there is very little agreement on a formal notion of correctness for access control for RDF. Oulmakhzoune et al. (2010) use the three correctness properties (`secure`, `sound` and `maximum`) proposed by Wang et al. (2007), to evaluate their query rewriting strategy for RDF data. However, given their query rewriting algorithm does not support negation, an investigation on how the correctness criteria can be used to validate SPARQL queries that include `FILTERS` is left for future work.

Although, a number of authors have proposed query rewriting strategies for basic SPARQL queries, we build on this work by providing a query rewriting strategy for both complex SPARQL queries and SPARQL update queries. In addition, we demonstrate how a set of correctness criteria, that was originally used to verify that a given access control policy holds over different database states, can be used to verify that our query rewriting algorithm is `secure`, `sound` and `maximum`. As the correctness criteria is not specific to RDF, it can be used in general to compare access control at the query layer to access control at the data layer.

6.7 Summary and Future Directions

Although the technology to link web data with other relevant data using machine-accessible formats has been in existence for a number of years, one could argue that without appropriate security and privacy mechanisms the LDW will struggle to reach its full potential as a global data space. A number of access control enforcement frameworks have been proposed for RDF, the data model which underpins the LDW. However, limited research has been conducted into providing partial data access to SPARQL queries or SPARQL update queries. Additionally, to date researchers have focused on performance evaluations, as opposed to verifying the correctness of the proposed access control mechanisms.

In this chapter, we proposed a query rewriting strategy for both the SPARQL 1.1 query and update languages. We adapted a set of criteria, which was originally used to verify the correctness of relational access control algorithms, to allow for access control via query rewriting to be compared against access control via results filtering. We subsequently used the adapted correctness criteria to evaluate the proposed query rewriting algorithms. Through evaluation we verified the correctness of our query rewriting strategy. However it is clear from the performance evaluation, that in the case of subqueries and filters over large datasets the performance impact would be prohibitive. Finally, we discussed how our Linked Data Authorisation Architecture (LinDAA) can be applied to the LDW.

In the current framework, both authorisations and propagation rules are specified declaratively. Given access control maintenance over a large number of declarative policies would be cumbersome for administrators, in future work we propose to investigate how existing policy languages can be used to represent our access control policies and propagation rules. In addition, we plan to develop an analytical tool, which can be used not only to graphically analyse explicit and implicit access control policies, but also to examine the potential impact of new access control policies.

Chapter 7

Conclusions

The Internet is growing exponentially, fuelling research into the next generation of Internet technologies. Over the past two decades much research, has been conducted into the use of semantic technologies and Linked Data for data integration and search over public data sources. This exponential expansion of data, and many of the challenges that come with it are mirrored within the enterprise. With the introduction of SPARQL 1.1, the infrastructure underpinning the Linked Data Web (LDW) could potentially be used as a global dataspace, which can support large scale data integration, search and analysis of both public and private data. However, given the potential sensitivity of private data, appropriate access control mechanisms need to be put in place.

In this section, we provide a summary of the overall access control strategy presented in this thesis. We subsequently detail the contributions that enable us to validate our hypothesis. We compare our access control strategy for the LDW to alternative approaches, based on the access control requirements identified during our literature survey. Finally, we conclude by identifying a number of avenues for future work.

7.1 Summary

Several researchers have proposed access control strategies for RDF data that could potentially be used to enforce access control over Linked Data. In *Chapter* 3, we provided a summary of the relevant access control models, standards, policy languages and enforcement frameworks. Rather than limit ourselves to existing proposals for access control over Linked Data, we adopt a broader view examining existing strategies for RDF irrespective of whether the authors apply their proposals to Linked Data or not. We discussed how a number of well known and emerging access control models (MAC, DAC, RBAC, ABAC, VBAC and CBAC) and relevant standardisation efforts (XACML, WebID, WAC and P3P), have been used in conjunction with, or enhanced by, semantic technologies. We subsequently examined access control policy specification using ontologies (KAoS), rules (Rei, Protune) and a combination of both (Proteus). Following on from this, we turned our focus to access control over RDF data. Here, we examined the various approaches used to specify access control policies (triple patterns, views, named graphs and ontology concepts) and reviewed the different reasoning strategies (propagation of authorisations, RDFS inference and reasoning over restricted data). In addition, we examined how data filtering and query rewriting can be used to return partial query results, when access to some of the data required by the query is permitted, and access to other data is prohibited. Finally, given that any of the policy languages or enforcement frameworks proposed to date, could potentially be used/adapted to enforce access control over Linked Data, we provided a summary of RDF access control requirements found in the literature, and categorised existing access control frameworks accordingly.

Our early work, which was presented in *Chapter* 4, focused on lifting both data and access control policies from existing LOB applications using an RDB2RDF tool, known as XSPARQL. Both the triples and the associated access permissions are represented using an extension of RDF, known as Annotated RDF, which allows domain specific meta-information (access rights in our case) to be attached to RDF triples. Domain operations are used to merge the permissions that relate to the same triple, and to infer new permissions for triples derived using RDFS inference rules. Although access control specification at the triple level works well when both the data and the access control policies are lifted from existing data sources, over large datasets it would be difficult to maintain access control at this level of granularity. Therefore, in addition to the RDFS inference rules, we proposed a number of rules that can be used to propagate permissions based on hierarchies of access control entities (subjects, access rights and resources). We subsequently demonstrated how both the proposed modelling and the inference rules can be used to support commonly used access control models, such as RBAC, ABAC and VBAC. We provided an overview of the components necessary for data integration and access control enforcement, and presented the results of our performance evaluation over increasing datasets. The results of the evaluation demonstrated that, when access control is present, there is an overhead associated with queries containing a single triple pattern. However, queries with two or more triple patterns took less time than the same query without access control. This can be attributed to the fact that queries with access control return less data, than those without.

Having shown that it is feasible to extract and enforce existing policies over relational data that is translated into RDF, and to represent existing access control models using rules, in *Chapter* 5 our focus turned to the enforcement and the administration of access control over distributed RDF data. Given DAC allows users to delegate their permissions to others, it is particularly suitable for managing access control over distributed data. In order to devise a suitable strategy for DAC over RDF data we examined how DAC is used in conjunction with the relational and the XML data models. Based on this analysis we identified the need for access rights at both the data and the schema levels, and a tight coupling between query operations and access rights. In order to cater for these additional requirements, we extended the original triple level access control by proposing a policy layer on top of the RDF query layer. We demonstrated how quad patterns (where the fourth element is used to match the named graph) can be used to specify access rights at multiple levels of granularity (including the triple level permissions from our earlier work). We also extended our inference rules, in order to cater for various RDFS relations (permission propagation based on class, property and instance relations). Following on from this, we demonstrated how the Flexible Authorisation Framework, originally proposed by Jajodia et al. (2001) for hierarchical data, can be adapted in order to cater for the enforcement and the administration of dynamic authorisations and inference rules over RDF graph data. The results of our initial performance evaluation showed a negligible increase in query processing time in light of both increasing datasets and increasing authorisations, and a linear increase in inference times over increasing authorisations. We concluded this chapter by identifying the need for further analysis in order to cater for complex SPARQL queries (graph patterns, filters, aggregates and subqueries) and SPARQL update operations.

To meet this need, in *Chapter* 6 we proposed a query rewriting strategy which translates graph based operations to triple operations, and filters out inaccessible data. In addition we proposed a strategy, which can be used to verify access control for the SPARQL query and update languages. To this end, we adapted a set of correctness criteria from relational databases, to allow for a comparison between our query rewriting strategy and access control via data filtering. We subsequently used the adapted correctness criteria to evaluate our query rewriting algorithms. We presented a benchmark, based on the BSBM dataset, that can be used to compare different access control strategies for Linked Data. Finally, we proposed the Linked Data Authorisation Architecture (LinDAA) and discussed how it can be used to cater for access control over Linked Data, irrespective of whether the data is queried via SPARQL endpoints, exposed in RDF documents, embedded in HTML documents or accessed via RDB2RDF interfaces.

7.2 Contributions

In this section, we present a summary of the contributions, that enable us to validate the hypothesis presented in this thesis.

The main hypothesis is as follows:

> **Access control for the Linked Data Web can be achieved by (i) a representation format which can be used to express access control policies that are lifted from relational databases or associated directly with RDF data; (ii) a set of rules that simplify access control specification and maintenance; and (iii) an enforcement strategy which allows for the retrieval of partial query results.**

Based on this hypothesis, we devised a number of research questions:

(i) **When relational data is exposed as RDF, how can we ensure the original access control policies are applied to the RDF data?**
Access control policies from existing LOB applications are often stored in relational databases. In *Chapter* 4, we discussed how RDB2RDF tools, such as XSPARQL, can be used to extract both data and access control policies from existing LOB applications. Both the triples and the associated access permissions are represented using Annotated RDF, which allows domain specific meta-information (access rights in our case) to be attached to RDF triples. Access is enforced by translating SPARQL queries into AnQL queries by using credentials supplied by the requester. This query rewriting step ensures that only triples with a matching non negative annotation are returned.

(ii) **Beyond triple level access control, what rules are necessary to (a) support existing access control models and (b) simplify access control specification and maintenance?**
In *Chapter* 4, we demonstrated how ρdf, a subset of RDFS, can be used not only to infer new triples, but also to infer the relevant access rights. We presented two annotation domain operations $\oplus$ and $\otimes$. The former is used to combine annotations when the same triple appears in multiple quads. Whereas, the latter is used to derive new annotations for triples derived using ρdf entailment rules. For the $\oplus$ access control domain operator we use disjunction, which maintains the access restrictions of both triples. Whereas for the $\otimes$ access control domain operator we use conjunction, which further restricts access to the triple.

We also proposed the following rules that can be used to provide support for a number of existing access control models, namely MAC, DAC, RBAC, VBAC and ABAC.

Resource based access is used to propagate access rights to all triples with the same subject. Such a rule is necessary to support VBAC, where access is granted to a number of entities simultaneously.

Hierarchical subject inheritance is used to propagate access rights based on the subject hierarchy. Access rights that are assigned to groups are inherited by all subgroups or individuals belonging to the specified group. This rule is necessary to support RBAC, where access rights are often propagated downwards based on nested groups.

Hierarchical subject subsumption is also used to propagate access rights based on the subject hierarchy. For example access rights assigned to employees are subsumed by their managers. However, in this instance lower level subjects inherit the access rights of higher level subjects. Although the vocabulary is different, this rule is essentially the same as the hierarchical subject inheritance rule.

Hierarchical resource inheritance is used to propagate access rights based on the resource hierarchy, with lower level resources inheriting the access rights of higher level resources. As this rule relates to the structure of resources, it is necessary for each of the aforementioned access control models.

Resource categorisation is used to propagate access rights to all resources that are of a given type. As before, this rule is required to support MAC, DAC, RBAC, VBAC and ABAC.

Hierarchical access rights subsumption is used to propagate access rights based on a partial order defined between access rights. Using this rule, lower level access rights will inherit the permissions/prohibitions assigned to higher level access rights. As this rule relates to the structure of access rights, like the previous two rules, it is relevant for each of the aforementioned access control models.

In *Chapter* 5, we proposed five additional rules that are necessary for schema based access control, similar to the type of rules proposed for DAC over relational data.

Classes to instances is used to propagate access rights assigned to a class, to all instances of that class.

Properties to instances is used to propagate access rights assigned to a property, to all instances of that property.

Instances to properties is used to propagate access rights assigned to an instance of a class, to all properties associated with that instance.

Classes to subclasses is used to propagate access rights assigned to a class, to all subclasses.

Properties to subproperties is used to propagate access rights assigned to a property, to all subproperties.

In addition to the propagation policies, we identified the need for a flexible framework that can support reasoning, not only over propagation rules, but also conflict resolution policies and integrity constraints.

Conflict resolution rules are used to support multiple conflict resolution strategies. For example, conflict resolution policies based on the structure of the different components required for access control over graph data; the sensitivity of the data requested; or contextual conditions pertaining to the requester.

Integrity constraints are used to specify restrictions on authorisation specification, thus decreasing the potential for runtime errors. For example, `INSERT` and `DELETE` can only be applied to an RDF quad, while `DROP`, `CREATE`, `COPY`, `MOVE` and `ADD` should only be associated with a named graph.

(iii) **What adjustments need to be made to SPARQL queries, to ensure that only authorised data is returned?**
In *Chapter* 6, we proposed a query rewriting strategy for the SPARQL 1.1 query and update languages.

For SPARQL queries we generate a `FILTER NOT EXISTS` expression, by binding the variables in the graph pattern group to the constants in the authorisation. Depending on the query, the following query rewriting strategies are applied:

- **Basic graph patterns and aggregates.** When an unauthorised quad pattern matches a quad pattern contained in a basic graph pattern the `FILTER NOT EXISTS` is added to the relevant graph pattern group.
- **Subqueries and negation.** If the matched graph pattern is part of a subquery or a filter the `FILTER NOT EXISTS` expression is added to the relevant graph pattern group in the inner `SELECT`, `FILTER NOT EXISTS`, `FILTER EXISTS` or `MINUS` expression.

For SPARQL updates in order to cater for the different query types, a number of rewriting strategies are proposed:

- **`DELETE/INSERT`.** The query rewriting strategy is the same as the SPARQL query rewriting strategy presented above.
- **`DELETE DATA` and `INSERT DATA`.** If any of the quads in the query match the unauthorised quad patterns, these quads are removed from the query.
- **`CLEAR` and `DROP`.** Negative authorisations pertaining to the specified graph are added as filters to a `DELETE` query, which is used to ensure only authorised data is removed from the graph.
- **`ADD` and `LOAD`.** Negative authorisations relating to the source and the destination graphs are added as filters to an `INSERT` query, which is used to add/load only authorised data to the destination graph.
- **`COPY`.** A `DELETE` query is used to remove all data from the destination graph. Negative authorisations that match the source graph are added as filters to an `INSERT` query, which is used to ensure that only authorised data is copied to the destination graph.
- **`MOVE`.** A `DELETE` query is used to remove all data from the destination graph. Negative authorisations associated with the source graph are added as filters to an `INSERT` query, which is used to ensure that only authorised data is moved to the destination graph. Finally, a `DELETE` query is used to delete data from the source graph.

In addition, we define a set of criteria that can be used to verify the correctness of access control via query rewriting. Using this criteria, it is possible to compare the results obtained when access control is enforced via query rewriting to those obtained when access control is enforced via filtering.

(iv) **What components are required to support the specification, enforcement and administration of access control for the LDW?**
In *Chapter* 5, we proposed G-FAF, a graph-based flexible authorisation framework, which

can be used to reason over authorisations, propagation rules, conflict resolution policies and integrity constraints.

Authorisations are rules that indicate the `access rights` that `authorisation subjects` are allowed/prohibited to perform on `data items`. In our case, data items are represented as *quad patterns*, a flexible mechanism which can be used to grant/restrict access to an RDF quad, a collection of RDF quads (multiple quads that share a common subject), a named graph (arbitrary views of the data) or specific classes or properties.

Propagation policies are used to simplify authorisation administration, by allowing for the derivation of implicit authorisations from explicit ones. For example, we can derive new authorisations based on the logical organisation of `authorisation subjects`, `access rights` and `data items` or the RDF Schema vocabulary.

Conflict Resolution Policies are rules that are used to determine precedence if one policy permit access and another policy prohibit access. Rather than propose a particular conflict resolution strategy, we provide a formal definition for a conflict resolution rule, that can be used to determine access given several different conflict resolution strategies. For example, conflict resolution policies based on the structure of the different components required for access control over graph data; the sensitivity of the data requested; or contextual conditions pertaining to the requester.

Integrity Constraints are used to restrict the specification of authorisations based on the existing relationships between SPARQL operations and RDF data items. For example, `INSERT` and `DELETE` can only be applied to an RDF quad, whereas `DROP`, `CREATE`, `COPY`, `MOVE` and `ADD` should only be associated with a named graph. As per conflict resolution, rather than propose specific integrity constraints, we provide a formal definition for an integrity constraint that can be used to determine if the requested action should be permitted.

In *Chapter* 6, we proposed LinDAA, our Linked Data authorisation architecture, which can be used to provide access control for Linked Data irrespective of how the data is represented or consumed.

- **RDF Data** can reside in an RDF Store, be represented as an RDF document, be embedded in a HTML document or generated using an RDB2RDF tool.
- **Access control policies** which are composed of authorisations and rules, are represented in a data store. Although we use a rule based approach, both an ontology based approach or a combined approach could also be adopted.
- The **Authorisation Framework**, G-FAF in our case, is used to reason over authorisation, propogation policies, conflict resolution rules and integrity constraints. This component also determines if access should be granted or denied to a particular graph pattern.
- The **Authorisation Engine** is responsible for determining the query rewriting strategy based on the query type, and for rewriting the query accordingly.
- The **Authorisation interface** is responsible for intercepting both queries and requests for documents. In the case of RDF documents, data embedded in HTML documents or relational data translated into RDF using RDB2RDF tools, this component also generates the appropriate SPARQL query. Either the submitted or the generated SPARQL query, together with the requesters credentials, are mapped to an *Authorisation Request* and submitted to the *Authorisation Engine*. The *Authorisation Engine* rewrites the

query to filter out unauthorised data, and returns the updated query to the *Authorisation Interface*. Finally the *Authorisation Interface* uses SPARQL to generate a filtered view of the data, and returns the results in the expected format to the requester.

7.3 Critical Assessment

In this section, we discuss how our proposal for access control for Linked Data, can be used in conjunction with existing data publishing and consumption strategies. In addition, we use the access control requirements identified in *Section* 3.4, to compare the work presented in this thesis against alternative approaches.

7.3.1 High Level Requirements

Firstly, we examine the extent to which our access control architecture can be used cater for access control over Linked Data.

Publishing Linked Data. In *Chapter* 6 we introduced LinDAA, our authorisation architecture for Linked Data. LinDAA is dependent on SPARQL, which is used to ensure that only authorised data is returned using our query rewriting strategy, irrespective of whether the data is exposed via an RDF Document, an RDB2RDF interface or a SPARQL endpoint. Through evaluation we demonstrated that in the case of basic graph patterns and aggregates access control via query rewriting takes marginal more time than the same queries executed without access control. Queries that are translated from graph based operations to triple based operations can take approximately twice as long as the graph based operations. `INSERT DATA` and `DELETE DATA` operations take one third of the time of the respective queries without access control. While queries that include subqueries or filters can take on average twenty times longer using our query rewriting strategy. Given there is no access control benchmark for RDF, it remains to be seen how our proposal compares to others.

Consuming Linked Data. When it comes to consuming Linked Data, there are three typical patterns (the *Crawling Pattern*, the *On-The-Fly-Dereferencing Pattern* and the *Query Federation Pattern*). Regardless of the mechanism used to consume Linked Data, the requester needs to be able to both dereference and query the RDF data. Although in this thesis we demonstrate how we can cater for both, we do not focus on the specific access control requirements arising from the different applications that consume Linked Data (browsers, search engines and domain specific applications). It would be interesting to investigate what are the different access control requirements, and to what degree G-FAF and LinDAA satisfy these different requirements.

7.3.2 Detailed Requirements

Next, we examine our proposal for access control for Lined Data with respect to the specification, enforcement, administration and implementation requirements we identified in *Section* 3.4.

7.3.2.1 Specification

When it comes to access control policy specification, we demonstrate how together declarative access control policies and rules can be used to provide support for a number of existing access control models. A summary of our support for the specification requirements, identified in *Section* 3.4.1, is depicted in *Table* 7.1.

Granularity. Existing proposals specify access control policies based on ontology concepts, classes, properties and individuals; graph patterns with/without filters; named graphs or views and triple patterns (*see Table* 7.1). In our early work, we specified permissions and prohibitions at the triple level and propagated access rights, for sets of subjects, access rights and resources, using rules. However, using such a modelling it was not possible to cater for graph wide permissions in general, or `CREATE` permissions in particular. Our follow-up work demonstrated how RDF quad patterns can be used to specify access control at multiple levels of granularity (a triple, a quad, a collection of RDF quads, a named graph, or specific classes or properties). Rather than simply granting or denying access, we demonstrated how query rewriting can be used to allow for SPARQL queries to be executed over data which is partially restricted.

Underlying Formalism. Generally speaking, the underlying formalism for existing proposals for access control over RDF data fall into one of two categories. With OWL based approaches adopting a description logic formalism (Amini and Jalili, 2010; Bao et al., 2007; Chen and Stuckenschmidt, 2010; Javanmardi et al., 2006a; Kolovski et al., 2007; Toninelli et al., 2009) and rule based approaches adopting a logic programming formalism (Bonatti and Olmedilla, 2007; Kagal and Finin, 2003; Toninelli et al., 2009). Given our reliance on rules we fall into the second category, with both our early work on aRDF and our later work on G-FAF both adopting a non recursive DATALOG formalism. As such the proposed access control policies have unambiguous semantics and can be evaluated in PTIME.

Reasoning. To date, a number of different reasoning strategies for access control over RDF have been proposed, with policy languages tending to focus on abductive and deductive reasoning, over both ontology concepts and individuals. Several researchers propose reasoning strategies that use RDFS entailment rules to either propagate permissions to existing triples, or to infer both new data and permission based on an extension of the existing RDF model. Others focus on subsumption reasoning based on hierarchies of subjects, predicates and objects. For a summary of existing reasoning strategies see *Table* 7.1. In this thesis, we present a number of specific rules that are necessary to provide support for common access control models. In addition, we propose a Graph-based Flexible Authorisation Framework (G-FAF), which can be used to support reasoning over propagation policies, integrity constraints and conflict resolution rules. One of the limitations of the existing framework is the lack of support for abductive reasoning. Therefore, it is currently not possible to determine the potential impact of new access control policies, or to determine the access rights required in order to satisfy a given policy. Although many general policy languages provide support for abductive reasoning (Amini and Jalili, 2010; Bonatti and Olmedilla, 2007; Chen and Stuckenschmidt, 2010; Kagal and Finin, 2003; Kolovski et al., 2007; Muhleisen et al., 2010; Toninelli et al., 2009; Uszok et al., 2003b), there is still an need for an investigation into the type of abductive rules that would be beneficial for the LDW, and a general syntax for specifying these rules.

Condition Expressiveness. A number of researchers who propose general policy languages (Amini and Jalili, 2010; Kagal and Finin, 2003; Toninelli et al., 2009; Uszok et al., 2003b) argue that in addition to permissions and prohibitions, it should be possible to specify both obligations and dispensations. This requirement stems from the desire to support a wider range of policies. Given we focus specifically on access control, both obligations and dispensations are deemed outside the scope of this work.

Attributes, Context & Evidences. Many solutions proposed to date, including ours, demonstrate how access can be granted/denied based on user attributes that are communicated using WebID (Muhleisen et al., 2010; Costabello et al., 2012b; Sacco and Passant, 2011b). A number of researchers also use contextual information pertaining to the requester, the system or the environment, in order to determine if access should be permitted or prohibited. Much of the work in this area has been on the provision of vocabularies that can be used to specify context aware access control policies (Corradi et al., 2004a; Costabello et al., 2012b; Montanari et al., 2005; Shen and Cheng, 2011) or demonstrating how contextual information can be represented in an access control policy (Abel et al., 2007; Amini and Jalili, 2010; Bonatti and Olmedilla, 2007; Franzoni et al., 2007; Gabillon and Letouzey, 2010; Kagal and Finin, 2003; Papakonstantinou et al., 2012; Sacco and Passant, 2011b; Toninelli et al., 2009; Uszok et al., 2003b). Given our access control framework uses credential matching to determine access, it would need to be extended in order to cater for reasoning over contextual information pertaining to the system and the environment.

Heterogeneity & Interoperability. Given the open nature of the web, access control for the LDW needs to be able to support a variety of access control policies, resources and users. The access control strategies, we examined in *Chapter* 3 provide support of RDF or OWL, with some also providing support for RDFS and SPARQL. Being aware of the interoperability benefits that can be obtained though adoption of standards, we have proposed an access control architecture which is tightly coupled with existing RDF, RDFS and SPARQL standards. Rather than propose a new access control model or policy language, we discuss how existing access control models can be enforced over RDF and propose a flexible authorisation framework which can be mapped to existing policy languages.

Table 7.1: Specification requirements

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Interoperability
LinDAA	quad patterns	SPARQL 1.1 queries & updates	DATALOG	flexible framework	permissions±	attributes & WebID	RDF & RDFS & SPARQL & RDB2RDF
Abel et al.	graph patterns & filters	bindings & filters	-	-	permissions±	context	RDF & SPARQL & SeRQL
Amini and Jalili & Ehsan et al.	ontology concepts & individuals	-	DL & deontic logic	deduction & abduction	permissions± obligation±	attributes & context	OWL
Bao et al.	ontology concepts	-	DL $\mathcal{SHIQ}$	privacy preserving	-	-	OWL
Bonatti and Olmedilla	ontology concepts	-	LP	deduction & abduction	permissions±	attributes & context & OpenID	RDF
Chen and Stuck-enschmidt	graph patterns & filters	bindings & filters	DL	deduction & abduction	permissions±	-	OWL & SPARQL
Costabello et al.	named graphs	named graphs	-	-	permissions±	attributes & context & WebID	RDF & SPARQL
Dietzold and Auer	triples, classes & properties	data filtering	-	-	permissions	-	RDF
Flouris et al.	graph patterns	bindings & filters	-	-	permissions±	-	RDF & SPARQL

Continued on next page

Table 7.1 – continued from previous page

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Heterogeneity & Interoperability
Franzoni et al.	ontology individuals	bindings & filters	-	RDFS entailment	permissions±	attributes & context	RDFS & SPARQL
Gabillon and Letouzey	named graph & views	-	-	-	permissions±	attributes & context	RDF & SPARQL
Jain and Farkas	triple patterns	-	-	RDFS entailment	permissions±	-	RDFS
Javanmardi et al.	ontology concepts & individuals	-	DL *SHOIN*	subjects, predicates & objects subsumption	permissions±	-	OWL
Kagal and Finin	ontology concepts	-	LP	deduction & abduction	permissions± obligation±	attributes & context	OWL
Kim et al.	triple pattern	-	-	RDFS entailment	permissions±	-	RDFS
Kolovski et al.	ontology concepts	-	DL *SHOIN* & defeasible logic	deduction & abduction	permissions±	attributes	OWL
Li and Cheung	views	propagation	-	-	permissions±	-	RDF
Muhleisen et al.	triple patterns	data filtering	-	deduction & abduction	permissions+	WebID & OpenID	OWL
Oulmakhzoune et al.	ontology concepts	bindings & filters	-	-	permissions±	-	RDF & SPARQL
Papakonstantinou et al.	triples	-	-	RDFS entailment	permissions±	context	RDFS & SPARQL
Qin and Atluri	concept	-	-	ontology concept relations	permissions±	-	RDF

Continued on next page

Table 7.1 – continued from previous page

	Granularity	Partial Results	Underlying Formalism	Reasoning	Condition Expressiveness	Attributes, Context & Evidences	Heterogeneity & Interoperability
Reddivari et al.	triple patterns	-	-	RDFS entailment	permissions±	attributes	RDFS
Ryutov et al.	nodes & edges	-	many sorted first order logic	subject & object subsumption	permissions±	attributes	RDF
Sacco and Passant	resource, triple & graph	-	-	-	permissions±	attributes & context & WebID & OpenID	RDF & SPARQL
Toninelli et al.	ontology concepts	-	DL & LP	deduction & abduction	permissions± obligation±	attributes & context	OWL
Uszok et al.	ontology concepts	-	DL	deduction & abduction	permissions± obligation±	context	OWL

7.3.2.2 Enforcement

Access control enforcement requirements refer to the various components that are used in order to determine if access should be granted or denied. Here our contribution is two fold: (i) The G-FAF enforcement algorithm demonstrates how together authorisations, propagation rules and conflict resolution policies can be used to enforce access control over RDF data; (ii) We propose a query rewriting strategy for the SPARQL 1.1 query and update languages. A summary of our support for the enforcement requirements, identified in *Section* 3.4.2, is depicted in *Table* 7.2.

Negotiation. The process whereby both the requester and the data provider exchange credentials, until a decision on whether to grant or deny access can be reached, is commonly known as negotiation. Negotiation commonly goes hand in hand with trust, both of which are often linked with privacy research. Although we do not specifically focus on privacy or trust in this thesis, the proposed framework, authorisation architecture and algorithms could potentially be used to enforce privacy policies. Given that individuals and organisations are more cautious of the type of information they provide to service providers, a negotiation component may be required in the future. However, like authentication, such a component could be implemented as an add-on to our current authorisation framework.

Explanations. Rather than simply granting or denying access to resources, a few researchers believe that the system should also provide details of how this decision was reached. Existing research on policy languages and explanations focus on how to associate explanations with access control policies or policy decisions (Bonatti and Olmedilla, 2007; Ryutov et al., 2009; Toninelli et al., 2009). In the case of our query rewriting strategy, we take the opposite approach, rather than informing the requester that they do not have the necessary access rights, we behave as if the data does not exist. Given the system may indirectly reveal unauthorised data, an interesting direction for further research would be to examine the trade-off between usability and security.

Conflict Resolution. A conflict arises when one policy grants access and another policy denies access. When it comes to conflict resolution there is *no one size that fits all*. Many researchers adopt a default approach to conflict resolution, choosing to either grant or deny access. Others either propose conflict resolution algorithms, or meta policies based on one or more conflict resolution strategies (*see Table* 7.2). For example, the structure of the graph, the sensitivity of the data, the nature of the request or contextual information pertaining to the requester. As conflict resolution requirements vary depending on the use case, rather than propose a specific conflict resolution strategy, we identify the need for a combined approach to conflict resolution. To meet this need, we provide a conflict resolution rule syntax and discuss how our enforcement framework can be used to determine access given several different conflict resolution strategies.

7.3.2.3 Administration

When it comes to access control administration, we demonstrate how discretionary access control can be used to guide the administration over RDF in general. In addition, the G-FAF administration algorithm ensures that any new information is accessible, and any redundant authorisations are removed. A summary of our support for the requirements, identified in *Section* 3.4.3, is depicted in *Table* 7.3.

Delegation. Like relational databases several researchers have proposed access control models whereby users are granted full access to the data they create, and they can subsequently

Table 7.2: Enforcement requirements

	Negotiation support	Explanation	Conflict Resolution
LinDAA	-	-	flexible framework
Abel et al.	-	-	default
Amini and Jalili &Ehsan et al.	bidirectional policies	-	algorithm
Bao et al.	-	-	algorithm
Bonatti and Olmedilla	bidirectional policies	queries	-
Chen and Stuckenschmidt	-	-	-
Costabello et al.	-	-	default
Dietzold and Auer	-	-	-
Flouris et al.	-	-	default
Franzoni et al.	-	-	-
Gabillon and Letouzey	-	-	default
Jain and Farkas	RDFS	-	algorithm
Javanmardi et al.	-	-	algorithm
Kagal and Finin	-	-	meta policies
Kim et al.	-	-	algorithm
Kolovski et al.	-	-	priorities
Li and Cheung	-	-	harmonisation
Muhleisen et al.	-	-	-
Oulmakhzoune et al.	-	-	-
Papakonstantinou et al.	-	-	default
Qin and Atluri	-	-	default
Reddivari et al.	-	-	meta policies
Ryutov et al.	-	user interface	algorithm
Sacco and Breslin	-	-	-
Toninelli et al.	bidirectional policies	descriptions	harmonisation
Uszok et al.	-	-	harmonisation

delegate some or all of their access rights to others. We go beyond existing proposals by demonstrating how the discretionary access control model, which allows users to delegate their permissions to others, can be used to enforce access control over graph data represented as RDF.

Consistency & Safety. Here consistency and safety refers to the access control policies as opposed to the data. For example, all data is accessible by someone or no user can elevate their own privileges. To date researchers ensure consistency and safety either using algorithms (Amini and Jalili, 2010; Bao et al., 2007; Jain and Farkas, 2006) or meta policies (Chen and Stuckenschmidt, 2010; Ryutov et al., 2009). Similar to the conflict resolution strategy, rather than propose specific consistency and safety policies we provide a syntax for integrity constraints, that can be used specify conditions that should generate an error.

Usability. From a usability perspective access control specification and maintenance should be made as simple as possible. A number of researchers have proposed user interfaces that can be used to specify and maintain access control policies. The access control mechanisms we propose in this thesis are by design policy language agnostic. As such, we adopt a declarative approach to access control policy specification, which can be mapped to existing

Table 7.3: Administration requirements

	Delegation	Consistency & Safety	Usability	Understandability
LinDAA	DAC	flexible framework	-	-
Abel et al.	-	-	-	-
Amini and JaliliEhsan et al.	case study	algorithm	-	-
Bao et al.	-	algorithm	-	-
Bonatti and Olmedilla	language	-	-	ProtuneX explanation
Chen and Stuckenschmidt	-	meta policies	-	-
Costabello et al.	-	-	-	-
Dietzold and Auer	-	-	-	-
Flouris et al.	-	-	-	-
Franzoni et al.	-	-	-	-
Gabillon and Letouzey	construct & describe	-	-	-
Jain and Farkas	-	algorithm	RACL admin module	-
Javanmardi et al.	-	-	-	-
Kagal and Finin	speech acts	-	-	-
Kim et al.	-	-	-	-
Kolovski et al.	-	-	-	analysis services
Li and Cheung	-	-	-	-
Muhleisen et al.	-	-	-	-
Oulmakhzoune et al.	-	-	-	-
Papakonstantinou et al.	-	-	-	-
Qin and Atluri	-	-	-	-
Reddivari et al.	-	-	-	-
Ryutov et al.	-	meta policies	RAW policy editor	RAW permission check
Sacco and Breslin	-	-	privacy preference manager	-
Toninelli et al.	-	-	-	-
Uszok et al.	-	-	KAoS Policy Admin Tool	policy disclosure

access control ontologies. However, in order to improve usability it would be beneficial to develop a user friendly interface, using a combination of input controls, containers and informational components to simplify access control specification.

Understandability. It should be possible for the system administrator to understand the interplay between existing policies and the effect new policies will have on the system. Although it is feasible to develop a user interface which would simplify the specification and maintenance of both authorisations and rules, access control administration over large datasets remains an open issue. When it comes to understandability an interesting avenue for future work would be to enable data visualisation and analytics over authorisations and propagation rules.

Table 7.4: Implementation requirements

	Effectiveness	Distributed Use case	Flexibility & Extensibility
LinDAA	performance & correctness evaluations	Linked Data use case & architecture	Java & Jena
Abel et al.	query rewriting performance	-	SeRQL, Protune
Amini and Jalili &Ehsan et al.	enforcement performance	case study	Prolog, GWT, Jena, JIP, Protege
Bao et al.	-	-	-
Bonatti and Olmedilla	explanation performance	demo	Java, TuProlog
Chen and Stuckenschmidt			Jena
Costabello et al.	enforcement performance BSBM & BTC datasets	mobile & Linked Data use cases	Java, Corese-KGRAM
Dietzold and Auer	-	-	RDF, SPARQL
Flouris et al.	annotation performance	-	Java, Jena, Sesame, Progress
Franzoni et al.	-	-	Java, SeRQL, Sesame
Gabillon and Letouzey	enforcement performance	-	Java, Tomcat, Sesame
Jain and Farkas	-	-	Java, Jena, Jess
Javanmardi et al.	policy reasoning	-	PELLET, SWRL
Kagal and Finin	-	use cases	Java, Prolog
Kim et al.	-	-	-
Kolovski et al.	policy reasoning Continue dataset	-	Pellet
Li and Cheung	-	-	-
Muhleisen et al.	enforcement performance BSBM dataset	demo	Joseki, Jena, Pellet
Oulmakhzoune et al.	- discussion	-	-
Papakonstantinou et al.	enforcement reasoning performance	-	PostgreSQL, MonetDB
Qin and Atluri	-	-	-
Reddivari et al.	query performance	-	Java, Jena, RDQL
Ryutov et al.	-	-	Java
Sacco and Breslin	enforcement performance	mobile & Linked Data use cases	Java
Toninelli et al.	-	-	-
Uszok et al.	-	use cases	Java

7.3.2.4 Implementation

From a non-functional requirements perspective we provide a set of correctness criteria, which can be used to verify the correctness of access control via query rewriting. In addition, we propose an authorisation architecture for Linked Data, called LinDAA, which can be used to both enforce and maintain access control over Linked Data. A summary of our support of the implementation requirements, identified in *Section* 3.4.4, is depicted in *Table* 7.4.

Effectiveness. In order to work in practice the proposed solution needs to be as efficient as possible. Like many of the existing proposals in this thesis we examine the performance our access control framework. However, in our case we also proposed a benchmark, based on the BSBM dataset, that can be used to compare different strategies for access control for Linked Data. In addition, we also highlight the need to verify the correctness of the proposed access control mechanisms. To this end, we propose a set of correctness criteria that can be used to compare access control via query rewriting to access control via data filtering.

Distributed. As RDF is a distributed data model, the proposed access control frameworks needs to be capable of enforcing access control over distributed data. Given our objective is to provide an access control mechanism for the LDW we propose an authorisation architecture and discuss how it can be used to enforce access control over distributed SPARQL endpoints, RDF documents, HTML document and RDB2RDF interfaces.

Flexibility & Extensible Architecture. RDF is a flexible and extensible data model. For this reason, any access control strategy needs to be able to cater for frequent changes to policies, users, access rights and resources. Our graph based flexible authorisation framework, which we call G-FAF, is not dependent on any particular vocabulary, and can be used to support a variety of authorisations, propagation rules, conflict resolution policies and integrity constraints.

7.4 Future Work

Although in this thesis, we demonstrate how together our authorisation framework and our authorisation architecture can be used to enforce access control over Linked Data, based on our critical analysis we have identified a number of possible avenues for future work.

Usability & Understandability.

Access control administration in general, and over large datasets in particular, can become extremely difficult to manage. Given that we allow for access control to be specified at multiple levels of granularity, and we also allow for reasoning over authorisations, the task of administration is even more cumbersome. An interesting avenue for future work, would be to investigate if graph based data clustering and visualisation techniques, such as those proposed by (Mutton and Golbeck, 2003), can be used to assist systems administrators to examine the interplay between authorisations and rules, and also determine the impact of new authorisations.

Explanations & Negotiation.

Earlier we saw that the benefits associated with explanations are two fold: (i) they allow the requester to understand what is required of them and (ii) they enable the policy owner to troubleshoot potential issues with existing policies. However, we believe that when it comes to explanations in particular and negotiation in general, there is a fine line between usability and security. As such, different levels of detail may need to be relayed to the requester depending on the context. In order to devise guidelines for access control explanations, we believe it would be beneficial to examine the different reasons for access denial and the potential security impact associated with both single and multiple explanations.

Privacy.

Given the strong link between access control and privacy, LinDAA could also be used to enforce privacy policies. A number of authors have proposed privacy vocabularies that can be used to specify privacy policies (Sacco and Breslin, 2012; Costabello et al., 2012b). Others investigate how social structures can be used to establish trust (Carroll et al., 2005; Golbeck and Hendler, 2006; Sacco and Breslin, 2014). Like our approach for access control over Linked Data, it would be interesting to demonstrate how reasoning over privacy policies can be used to simplify privacy policy specification and maintenance. From a subject perspective, it may be possible to leverage existing trust mechanisms that rely on social structures. From an access rights perspective, further analysis is required in order to determine the suitability of the proposed SPARQL operations. Whereas, from a resource perspective, it may be possible to identify specific categorisation mechanisms that can be used to simplify the administration of privacy policies.

Context Awareness.

As our authorisation framework uses credential matching to determine access, in its current state it can be used to enforce access control policies that are based on contextual information pertaining to the requester. A number of researchers have proposed context aware access control vocabularies that can be leveraged by our framework (Costabello et al., 2012b; Kagal and Finin, 2003). However, additional interfaces would need to be provided in order to obtain contextual information with respect to the system or the environment. Like privacy policies, it would be interesting to investigate how reasoning over contextual data can be used to simplify policy specification and maintenance. However, given that contextual data is highly dynamic, our framework may need to be updated to ensure efficient reasoning over streaming data.

Bibliography

Abel, F., De Coi, J., Henze, N., Koesling, A., Krause, D., Olmedilla, D., 2007. Enabling advanced and context-dependent access control in rdf stores. In: The Semantic Web. Vol. 4825 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–14.
URL `http://dx.doi.org/10.1007/978-3-540-76298-0_1`

Adida, B., Birbeck, M., McCarron, S., Herman, I., August 2013. RDFa Core 1.1 Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, available at `http://www.w3.org/TR/rdfa-core/`, W3C.

Alcaraz Calero, J., Martinez Perez, G., Gomez Skarmeta, A., December 2010. Towards an authorisation model for distributed systems based on the semantic web. Information Security, IET 4 (4), 411–421.

Amini, M., Jalili, R., December 2010. Multi-level authorisation model and framework for distributed semantic-aware environments. Information Security, IET 4 (4), 301–321.

Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J., September 2012. A Direct Mapping of Relational Data to RDF. W3C Recommendation, available at `http://www.w3.org/TR/rdb-direct-mapping/`, W3C.

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., Watson, V., Jun. 1976. System r: Relational approach to database management. ACM Trans. Database Syst. 1 (2), 97–137.
URL `http://doi.acm.org/10.1145/320455.320457`

Baader, F., Knechtel, M., Peñaloza, R., 2009. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In: Proceedings of the 8th International Semantic Web Conference. ISWC '09. Springer-Verlag, Berlin, Heidelberg, pp. 49–64.
URL `http://dx.doi.org/10.1007/978-3-642-04930-9_4`

Bao, J., Slutzki, G., Honavar, V., 2007. Privacy-preserving reasoning on the semanticweb. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence. WI '07. IEEE Computer Society, Washington, DC, USA, pp. 791–797.
URL `http://dx.doi.org/10.1109/WI.2007.147`

Berners-Lee, T., Cyganiak, R., Hausenblas, M., Presbrey, J., Seneviratne, O., Ureche, O.-E., 2009. Realising a read-write web of data. Technical Report, available at `http://web.mit.edu/presbrey/Public/rw-wod.pdf`.

Berners-Lee, Tim, 2006. Linked data-design issues. Article, available at `http://www.w3.org/DesignIssues/LinkedData.html`.

Bertino, E., Castano, S., Ferrari, E., May 2001. Securing xml documents with author-x. Internet Computing, IEEE 5 (3), 21–31.

Bertino, E., Samarati, P., Jajodia, S., Jan 1997. An extended authorization model for relational databases. Knowledge and Data Engineering, IEEE Transactions on 9 (1), 85–101.

Bertino, E., Sandhu, R., Jan 2005. Database security - concepts, approaches, and challenges. Dependable and Secure Computing, IEEE Transactions on 2 (1), 2–19.

Bertino, Elisa and Samarati, Pierangela and Jajodia, Sushil, 1993. Authorizations in Relational Database Management Systems. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. CCS '93. ACM, New York, NY, USA, pp. 130–139.
URL `http://doi.acm.org/10.1145/168588.168605`

Bonatti, P., De Coi, J., Olmedilla, D., Sauro, L., n.d. Protune: A rule-based provisional trust negotiation framework.

Bonatti, P., Olmedilla, D., June 2005. Driving and monitoring provisional trust negotiation with metapolicies. In: Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on. pp. 14–23.

Bonatti, P., Samarati, P., 2000. Regulating service access and information release on the web. In: Proceedings of the 7th ACM Conference on Computer and Communications Security. CCS '00. ACM, New York, NY, USA, pp. 134–143.
URL `http://doi.acm.org/10.1145/352600.352620`

Bonatti, P. A., Olmedilla, D., 2007. Rule-based policy representation and reasoning for the semantic web. In: Proceedings of the Third International Summer School Conference on Reasoning Web. RW'07. Springer-Verlag, Berlin, Heidelberg, pp. 240–268.
URL `http://dl.acm.org/citation.cfm?id=2391482.2391488`

Bradshaw, J., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M., Acquisti, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., Van Hoof, R., 2003. Representation and reasoning for daml-based policy and domain services in kaos and nomads. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS '03. ACM, New York, NY, USA, pp. 835–842.
URL `http://doi.acm.org/10.1145/860575.860709`

Bradshaw, J. M., Dutfield, S., Benoit, P., Woolley, J. D., 1997. Software agents. MIT Press, Cambridge, MA, USA, Ch. KAoS: Toward an Industrial-strength Open Agent Architecture, pp. 375–418.
URL `http://dl.acm.org/citation.cfm?id=267985.268008`

Brickley, D., Guha, R., February 2014. RDF Schema 1.1. W3C Recommendation, available at `http://www.w3.org/TR/2014/REC-rdf-schema-20140225/Overview.html`, W3C.

Carroll, J. J., Bizer, C., Hayes, P., Stickler, P., 2005. Named graphs, provenance and trust. In: Proceedings of the 14th International Conference on World Wide Web. WWW '05. ACM, New York, NY, USA, pp. 613–622.
URL `http://doi.acm.org/10.1145/1060745.1060835`

Chen, W., Stuckenschmidt, H., 2010. A model-driven approach to enable access control for ontologies. Business Services: Konzepte, Technologien, Anwendungen: Wirtschaftsinformatik 2009, 663–672.
URL `http://www.chenwilly.de/publications/wi2009.pdf`

Cirio, L., Cruz, I. F., Tamassia, R., 2007. A role and attribute based access control system using semantic web technologies. In: Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II. OTM'07. Springer-Verlag, Berlin, Heidelberg, pp. 1256–1266.
URL `http://dl.acm.org/citation.cfm?id=1780453.1780518`

Codd, E. F., Jun. 1970. A relational model of data for large shared data banks. Commun. ACM 13 (6), 377–387.
URL `http://doi.acm.org/10.1145/362384.362685`

Corradi, A., Montanari, R., Tibaldi, D., Sept 2004a. Context-based access control for ubiquitous service provisioning. In: Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International. pp. 444–451 vol.1.

Corradi, A., Montanari, R., Tibaldi, D., 2004b. Context-based access control management in ubiquitous environments. In: Proceedings of the Network Computing and Applications, Third IEEE International Symposium. NCA '04. IEEE Computer Society, Washington, DC, USA, pp. 253–260.
URL `http://dl.acm.org/citation.cfm?id=1025126.1025959`

Costabello, L., Villata, S., Delaforge, N., Gandon, F., 2012a. Linked data access goes mobile: Context-aware authorization for graph stores. In: LDOW - 5th WWW Workshop on Linked Data on the Web.
URL `http://hal.archives-ouvertes.fr/hal-00691256/`

Costabello, L., Villata, S., Gandon, F., 2012b. Context-aware access control for rdf graph stores. In: ECAI. pp. 282–287.

Costabello, L., Villata, S., Rodriguez Rocha, O., Gandon, F., 2013. Access control for http operations on linked data. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (Eds.), The Semantic Web: Semantics and Big Data. Vol. 7882 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 185–199.
URL `http://dx.doi.org/10.1007/978-3-642-38288-8_13`

Cranor, L., Langheinrich, M., Marchiori, M., April 2002. A P3P Preference Exchange Language (APPEL). W3C Working Draft, available at `http://www.w3.org/TR/P3P-preferences/`, W3C.

Cyganiak, R., Wood, D., Lanthaler, M., February 2014. RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, available at `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`, W3C.

Damiani, E., di Vimercati, S., Samarati, P., Dec 2005. New paradigms for access control in open environments. In: Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on. pp. 540–545.

Damianou, N., Dulay, N., Lupu, E., Sloman, M., 2001. The ponder policy specification language. In: Proceedings of the International Workshop on Policies for Distributed Systems and Networks. POLICY '01. Springer-Verlag, London, UK, UK, pp. 18–38.
URL `http://dl.acm.org/citation.cfm?id=646962.712108`

Das, S., Sundara, S., Cyganiak, R., September 2012. R2RML: RDB to RDF Mapping Language. W3C Recommendation, available at `http://www.w3.org/TR/r2rml/`, W3C.

De Coi, J., Karger, P., Koesling, A., Olmedilla, D., Jan 2008. Control your elearning environment: Exploiting policies in an open infrastructure for lifelong learning. Learning Technologies, IEEE Transactions on 1 (1), 88–102.

Denker, G., Kagal, L., Finin, T., Jan. 2005. Security in the semantic web using owl. Inf. Secur. Tech. Rep. 10 (1), 51–58.
URL `http://dx.doi.org/10.1016/j.istr.2004.11.002`

Di, W., Jian, L., Yabo, D., Miaoliang, Z., 2005. Using semantic web technologies to specify constraints of rbac. In: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies. PDCAT '05. IEEE Computer Society, Washington, DC, USA, pp. 543–545.
URL `http://dx.doi.org/10.1109/PDCAT.2005.247`

Dietzold, S., Auer, S., 2006. Access control on rdf triple stores from a semantic wiki perspective. In: Proceedings of the ESWC'06 Workshop on Scripting for the Semantic Web.

Ehsan, M. A., Amini, M., Jalili, R., 2009. A semantic-based access control mechanism using semantic technologies. In: Proceedings of the 2nd International Conference on Security of Information and Networks. SIN '09. ACM, New York, NY, USA, pp. 258–267.
URL http://doi.acm.org/10.1145/1626195.1626259

Evered, M., Bögeholz, S., 2004. A case study in access control requirements for a health information system. In: Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32. ACSW Frontiers '04. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 53–61.
URL http://dl.acm.org/citation.cfm?id=976440.976447

Fagin, R., Sep. 1978. On an authorization mechanism. ACM Trans. Database Syst. 3 (3), 310–319.
URL http://doi.acm.org/10.1145/320263.320288

Ferrini, R., Bertino, E., 2009. Supporting rbac with xacml+owl. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies. SACMAT '09. ACM, New York, NY, USA, pp. 145–154.
URL http://doi.acm.org/10.1145/1542207.1542231

Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W., Thuraisingham, B., 2008a. Rowlbac: Representing role based access control in owl. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies. SACMAT '08. ACM, New York, NY, USA, pp. 73–82.
URL http://doi.acm.org/10.1145/1377836.1377849

Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W. H., Thuraisingham, B., February 2008b. Using owl to model role based access control. Tech. rep., University of Maryland, Baltimore County.

Flouris, G., Fundulaki, I., Michou, M., Antoniou, G., 2010. Controlling access to rdf graphs. In: Proceedings of the Third Future Internet Conference on Future Internet. FIS'10. Springer-Verlag, Berlin, Heidelberg, pp. 107–117.
URL http://dl.acm.org/citation.cfm?id=1929268.1929280

Franzoni, S., Mazzoleni, P., Valtolina, S., Bertino, E., July 2007. Towards a fine-grained access control model and mechanisms for semantic databases. In: Web Services, 2007. ICWS 2007. IEEE International Conference on. pp. 993–1000.

Gabillon, A., 2004. An authorization model for xml databases. In: Proceedings of the 2004 Workshop on Secure Web Service. SWS '04. ACM, New York, NY, USA, pp. 16–28.
URL http://doi.acm.org/10.1145/1111348.1111351

Gabillon, A., Letouzey, L., Sept 2010. A view based access control model for sparql. In: Network and System Security (NSS), 2010 4th International Conference on. pp. 105–112.

Gandon, F., 2014. URL-URI-IRI. Accessed 30 December 2014, available at http://www-sop.inria.fr/members/Fabien.Gandon/docs/URL-URI-IRI.PNG.

Garcia, D., Toledo, M., July 2008. A web service privacy framework based on a policy approach enhanced with ontologies. In: Computational Science and Engineering Workshops, 2008. CSEWORKSHOPS '08. 11th IEEE International Conference on. pp. 209–214.

Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K., Winslett, M., 2004. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In: Bussler, C., Davies, J., Fensel, D., Studer, R. (Eds.), The Semantic Web: Research and Applications. Vol. 3053 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 342–356.
URL http://dx.doi.org/10.1007/978-3-540-25956-5_24

Gearon, P., Passant, A., Polleres, A., March 2013. SPARQL 1.1 Query Language. W3C recommendation, available at `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`, W3C.

Golbeck, J., Hendler, J., Nov. 2006. Inferring binary trust relationships in web-based social networks. ACM Trans. Internet Technol. 6 (4), 497–529.
URL `http://doi.acm.org/10.1145/1183463.1183470`

Gomez-Perez, A., Corcho, O., Jan 2002. Ontology languages for the semantic web. Intelligent Systems, IEEE 17 (1), 54–60.

Governatori, G., 2004. Defeasible description logics. In: Rules and Rule Markup Languages for the Semantic Web. Springer, pp. 98–112.

Griffiths, P. P., Wade, B. W., Sep. 1976. An authorization mechanism for a relational database system. ACM Trans. Database Syst. 1 (3), 242–255.
URL `http://doi.acm.org/10.1145/320473.320482`

Gutierrez, C., Hurtado, C., Vaisman, A., Feb 2007. Introducing time into rdf. Knowledge and Data Engineering, IEEE Transactions on 19 (2), 207–218.

Harris, S., Seaborne, A., March 2013. SPARQL 1.1 Update. W3C recommendation, available at `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`, W3C.

Hayes, P. J., Patel-Schneider, P. F., February 2014. RDF 1.1 Semantics. W3C recommendation, available at `http://www.w3.org/TR/rdf11-mt/`, W3C.

Heath, T., Bizer, C., 2011. Linked data: Evolving the web into a global data space. Vol. 1. Morgan & Claypool Publishers.
URL `http://linkeddatabook.com/`

Hollenbach, J., Presbrey, J., Berners-Lee, T., 2009. 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.

Inkster, T., Story, H., Harbulot, B., March 2014. WebID Authentication over TLS. W3C Editor's Draft, available at `http://www.w3.org/2005/Incubator/webid/spec/tls/`, W3C.

Jain, A., Farkas, C., 2006. Secure resource description framework: An access control model. In: Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies. SACMAT '06. ACM, pp. 121–129.
URL `http://doi.acm.org/10.1145/1133058.1133076`

Jajodia, S., Samarati, P., Sapino, M. L., Subrahmanian, V. S., Jun. 2001. Flexible support for multiple access control policies. ACM Trans. Database Syst. 26 (2), 214–260.
URL `http://doi.acm.org/10.1145/383891.383894`

Javanmardi, S., Amini, M., Jalili, R., 2006a. An Access Control Model for Protecting Semantic Web Resources. In: 2nd Semantic Web Policy Workshop. CEUR-WS.

Javanmardi, S., Amini, M., Jalili, R., GanjiSaffar, Y., 2006b. Sbac: A semantic based access control model. In: 11th Nordic Workshop on Secure IT-systems (NordSec'06), Linkping, Sweden. Vol. 22.

Johnson, M., Chang, P., Jeffers, R., Bradshaw, J., Soo, V., Breedy, M., Bunch, L., Kulkarni, S., Lott, J., Suri, N., et al., 2003. Kaos semantic policy and domain services: An application of daml to web services-based grid architectures. In: Proceedings of the AAMAS. Vol. 3.

Kagal, L., Berners-lee, T., 2005. Rein : Where policies meet rules in the semantic web. Tech. rep., Laboratory, Massachusetts Institute of Technology.

Kagal, L., Berners-Lee, T., Connolly, D., Weitzner, D., 2006. Using semantic web technologies for policy management on the web. In: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2. AAAI'06. AAAI Press, pp. 1337–1344.
URL http://dl.acm.org/citation.cfm?id=1597348.1597401

Kagal, L., Finin, T., 2003. A policy language for a pervasive computing environment. In: Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks. IEEE Comput. Soc, pp. 63–74.

Kagal, L., Finin, T., Joshi, A., 2003a. A policy based approach to security for the semantic web. In: Fensel, D., Sycara, K., Mylopoulos, J. (Eds.), The Semantic Web - ISWC 2003. Vol. 2870 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 402–418.
URL http://dx.doi.org/10.1007/978-3-540-39718-2_26

Kagal, L., Finin, T., Joshi, A., 2003b. A policy based approach to security for the semantic web. In: Fensel, D., Sycara, K., Mylopoulos, J. (Eds.), The Semantic Web - ISWC 2003. Vol. 2870 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 402–418.
URL http://dx.doi.org/10.1007/978-3-540-39718-2_26

Kifer, M., Subrahmanian, V., 1992. Theory of generalized annotated logic programming and its applications. The Journal of Logic Programming 12 (4), 335 – 367.
URL http://www.sciencedirect.com/science/article/pii/074310669290007P

Kim, J., Jung, K., Park, S., 2008. An introduction to authorization conflict problem in rdf access control. In: Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II. KES '08. Springer-Verlag, Berlin, Heidelberg, pp. 583–592.
URL http://dx.doi.org/10.1007/978-3-540-85565-1_72

Kirrane, S., 2011. Dc proposal: Knowledge based access control policy specification and enforcement. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (Eds.), The Semantic Web - ISWC 2011. Vol. 7032 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 293–300.
URL http://dx.doi.org/10.1007/978-3-642-25093-4_21

Kirrane, S., Abdelrahman, A., Mileo, A., Decker, S., 2013a. Secure manipulation of linked data. In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (Eds.), The Semantic Web - ISWC 2013. Vol. 8218 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 248–263.
URL http://dx.doi.org/10.1007/978-3-642-41335-3_16

Kirrane, S., Lopes, N., Mileo, A., Decker, S., 2013b. Protect your rdf data! In: Takeda, H., Qu, Y., Mizoguchi, R., Kitamura, Y. (Eds.), Semantic Technology. Vol. 7774 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 81–96.
URL http://dx.doi.org/10.1007/978-3-642-37996-3_6

Kirrane, S., Mileo, A., Decker, S., 2013c. Applying dac principles to the rdf graph data model. In: Janczewski, L., Wolfe, H., Shenoi, S. (Eds.), Security and Privacy Protection in Information Processing Systems. Vol. 405 of IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, pp. 69–82.
URL http://dx.doi.org/10.1007/978-3-642-39218-4_6

Klyne, G., Carroll, J. J., McBride, B., February 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, available at http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, W3C.

Knechtel, M., Stuckenschmidt, H., 2010. Query-based access control for ontologies. In: Hitzler, P., Lukasiewicz, T. (Eds.), Web Reasoning and Rule Systems. Vol. 6333 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 73–87.
URL http://dx.doi.org/10.1007/978-3-642-15918-3_7

Kodali, N., Farkas, C., Wijesekera, D., 2003. Multimedia access control using rdf metadata. In: Meersman, R., Tari, Z. (Eds.), On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops. Vol. 2889 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 718–731.
URL http://dx.doi.org/10.1007/978-3-540-39962-9_73

Kodali, N., Farkas, C., Wijesekera, D., 2004. An authorization model for multimedia digital libraries. International Journal on Digital Libraries 4 (3), 139–155.
URL http://dx.doi.org/10.1007/s00799-004-0080-1

Kolari, P., Ding, L., Shashidhara, G., Joshi, A., Finin, T., Kagal, L., June 2005. Enhancing web privacy protection through declarative policies. In: Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on. pp. 57–66.

Kolovski, V., Hendler, J., Parsia, B., 2007. Analyzing web access control policies. In: Proceedings of the 16th International Conference on World Wide Web. WWW '07. ACM, New York, NY, USA, pp. 677–686.
URL http://doi.acm.org/10.1145/1242572.1242664

Li, H., Zhang, X., Wu, H., Qu, Y., 2005. Design and application of rule based access control policies. In: ISWC Workshop on Semantic Web and Policy. pp. 34–41.

Li, J., Cheung, W., 2008. Query rewriting for access control on semantic web. In: Jonker, W., Petković, M. (Eds.), Secure Data Management. Vol. 5159 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 151–168.
URL http://dx.doi.org/10.1007/978-3-540-85259-9_10

Lopes, N., Bischof, S., Decker, S., Polleres, A., 2011. On the semantics of heterogeneous querying of relational, xml and rdf data with xsparql. EPIA2011–COLA Track, Lisbon, Portugal.

Lopes, N., Kirrane, S., Zimmermann, A., Polleres, A., Mileo, A., 2012. A Logic Programming approach for Access Control over RDF. In: Technical Communications of ICLP'12. Vol. 17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 381–392.

Lopes, N., Polleres, A., Straccia, U., Zimmermann, A., 2010. Anql: Sparqling up annotated rdfs. In: The Semantic Web - ISWC 2010. Vol. 6496 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 518–533.
URL http://dx.doi.org/10.1007/978-3-642-17746-0_33

McCollum, C., Messing, J., Notargiacomo, L., May 1990. Beyond the pale of mac and dac-defining new forms of access control. In: Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on. pp. 190–200.

McGuinness, D. L., van Harmelen, F., February 2004. OWL Web Ontology Language Overview. W3C recommendation, available at http://www.w3.org/TR/owl-features/, W3C.

Montanari, R., Toninelli, A., Bradshaw, J., Aug 2005. Context-based security management for multi-agent systems. In: Multi-Agent Security and Survivability, 2005 IEEE 2nd Symposium on. pp. 75–84.

Muñoz, S., Pérez, J., Gutierrez, C., Sep. 2009. Simple and efficient minimal rdfs. Web Semant. 7 (3), 220–234.
URL http://dx.doi.org/10.1016/j.websem.2009.07.003

Muhleisen, H., Kost, M., Freytag, J.-C., 2010. SWRL-based Access Policies for Linked Data. In: Proceedings of the Second Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2010). URL `http://CEUR-WS.org/Vol-576/paper1.pdf`

Mutton, P., Golbeck, J., July 2003. Visualization of semantic metadata and ontologies. In: Information Visualization, 2003. IV 2003. Proceedings. Seventh International Conference on. pp. 300–305.

Nematzadeh, A., Pournajaf, L., April 2008. Privacy concerns of semantic web. In: Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on. pp. 1272–1273.

O'Hara, K., Contractor, N. S., Hall, W., Hendler, J. A., Shadbolt, N., 2013. Web science: understanding the emergence of macro-level features on the world wide web. Foundations and Trends in Web Science 4 (2-3), 103–267.

Oulmakhzoune, S., Cuppens-Boulahia, N., Cuppens, F., Morucci, S., 2010. fquery: Sparql query rewriting to enforce data confidentiality. In: Foresti, S., Jajodia, S. (Eds.), Data and Applications Security and Privacy XXIV. Vol. 6166 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 146–161.
URL `http://dx.doi.org/10.1007/978-3-642-13739-6_10`

Papakonstantinou, V., Michou, M., Fundulaki, I., Flouris, G., Antoniou, G., 2012. Access control for rdf graphs using abstract models. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies. SACMAT '12. ACM, New York, NY, USA, pp. 103–112.
URL `http://doi.acm.org/10.1145/2295136.2295155`

Pérez, J., Arenas, M., Gutierrez, C., Sep. 2009. Semantics and complexity of sparql. ACM Trans. Database Syst. 34 (3), 16:1–16:45.
URL `http://doi.acm.org/10.1145/1567274.1567278`

Priebe, T., Dobmeier, W., Kamprath, N., 2006. Supporting attribute-based access control with ontologies. In: Proceedings of the First International Conference on Availability, Reliability and Security. ARES '06. IEEE Computer Society, Washington, DC, USA, pp. 465–472.
URL `http://dx.doi.org/10.1109/ARES.2006.127`

Priebe, T., Dobmeier, W., Schläger, C., Kamprath, N., 2007. Supporting attribute-based access control in authorization and authentication infrastructures with ontologies. Journal of Software 2 (1), 27–38.
URL `http://dblp.uni-trier.de/db/journals/jsw/jsw2.html#PriebeDSK07`

Priebe, T., Fernandez, E., Mehlau, J., Pernul, G., 2004. A pattern system for access control. In: Farkas, C., Samarati, P. (Eds.), Research Directions in Data and Applications Security XVIII. Vol. 144 of IFIP International Federation for Information Processing. Springer US, pp. 235–249.
URL `http://dx.doi.org/10.1007/1-4020-8128-6_16`

Qin, L., Atluri, V., 2003. Concept-level access control for the semantic web. In: Proceedings of the 2003 ACM Workshop on XML Security. XMLSEC '03. ACM, pp. 94–103.
URL `http://doi.acm.org/10.1145/968559.968575`

Reddivari, P., Finin, T., Joshi, A., 2005. Policy-based access control for an rdf store. In: Proceedings of the Policy Management for the Web workshop. Vol. 120. pp. 78–83.

Rissanen, E., January 2013. Extensible access control markup language (xacml) version 3.0. OASIS Standard, available at `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html`, OASIS Committee Specification.

Rubart, J., 2005. Context-based access control. In: Proceedings of the 2005 Symposia on Metainformatics. MIS '05. ACM, New York, NY, USA.
URL `http://doi.acm.org/10.1145/1234324.1234337`

Ryutov, T., Kichkaylo, T., Neches, R., 2009. Access control policies for semantic networks. In: Proceedings of the 2009 IEEE International Symposium on Policies for Distributed Systems and Networks. POLICY '09. IEEE Computer Society, pp. 150–157.
URL `http://dx.doi.org/10.1109/POLICY.2009.11`

Ryutov, T., Kichkaylo, T., Neches, R., Orosz, M., May 2008. Sfinks: Secure focused information, news, and knowledge sharing. In: Technologies for Homeland Security, 2008 IEEE Conference on. pp. 404–409.

Sacco, O., Breslin, J., 2014. In users we trust: towards social user interactions based trust assertions for the social semantic web. Social Network Analysis and Mining 4 (1).
URL `http://dx.doi.org/10.1007/s13278-014-0229-x`

Sacco, O., Breslin, J. G., 2012. PPO & PPM 2.0: Extending the privacy preference framework to provide finer-grained access control for the web of data. In: Proceedings of the 8th International Conference on Semantic Systems. I-SEMANTICS '12. ACM, pp. 80–87.
URL `http://doi.acm.org/10.1145/2362499.2362511`

Sacco, O., Collina, M., Schiele, G., Corazza, G., Breslin, J., Hauswirth, M., 2013. Fine-grained access control for rdf data on mobile devices. In: Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (Eds.), Web Information Systems Engineering – WISE 2013. Vol. 8180 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 478–487.
URL `http://dx.doi.org/10.1007/978-3-642-41230-1_40`

Sacco, O., Passant, A., 2011a. A privacy preference manager for the social semantic web. In: Semantic Personalized Information Management: Retrieval and Recommendation. CEUR-WS.
URL `http://ceur-ws.org/Vol-781/paper6.pdf`

Sacco, O., Passant, A., 2011b. A privacy preference ontology (ppo) for linked data. In: Linked Data on the Web. CEUR-WS.
URL `http://ceur-ws.org/Vol-813/ldow2011-paper01.pdf`

Sacco, O., Passant, A., Decker, S., Nov 2011. An access control framework for the web of data. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on. pp. 456–463.

Samarati, P., de Vimercati, S., 2001. Access control: Policies, models, and mechanisms. In: Focardi, R., Gorrieri, R. (Eds.), Foundations of Security Analysis and Design. Vol. 2171 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 137–196.
URL `http://dx.doi.org/10.1007/3-540-45608-2_3`

Sambra, A., Story, H., Berners-Lee, T., March 2014. Web Identity and Discovery. W3C Editor's Draft, available at `http://www.w3.org/2005/Incubator/webid/spec/identity/`, W3C.

Sandhu, R., Coyne, E., Feinstein, H., Youman, C., Dec 1994. Role-based access control: a multi-dimensional view. In: 10th Annual Computer Security Applications Conference. pp. 54–62.

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

Sandhu, R. S., 1998. Role-based access control. Advances in Computers, Elsevier 46, 237 – 286.
URL `http://www.sciencedirect.com/science/article/pii/S0065245808602065`

Seaborne, A., Prud'hommeaux, E., January 2008. SPARQL Query Language for RDF. W3C recommendation, available at `http://www.w3.org/TR/rdf-sparql-query/`, W3C.

Shen, H., Cheng, Y., 2011. A semantic context-based model for mobile web services access control. International Journal of Computer Network and Information Security, MECS Publisher 1, 18–25.

Speicher, S., Arwe, J., Malhotra, A., June 2014. Linked Data Platform 1.0. W3C Candidate Recommendation, available at `http://www.w3.org/TR/ldp/`, W3C.

Sporny, M., Inkster, T., Story, H., Harbulot, B., Bachmann-Gmür, R., Nov 2011. WebID 1.0 - Web Identification and Discovery. W3C working draft, W3C, available at `http://www.w3.org/2005/Incubator/webid/spec/`.

Stermsek, G., Strembeck, M., Neumann, G., 2004. Using subject-and object-specific attributes for access control in web-based knowledge management systems. In: Workshop on Secure Knowledge Management (SKM).

Straccia, U., 2009. A minimal deductive system for general fuzzy rdf. In: Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems. RR '09. Springer-Verlag, pp. 166–181.
URL `http://dx.doi.org/10.1007/978-3-642-05082-4_12`

Suri, N., Bradshaw, J., Burstein, M., Uszok, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J., 2003. Daml-based policy enforcement for semantic data transformation and filtering in multi-agent systems. In: Multi-Agent Systems and Applications III. Vol. 2691 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 122–136.
URL `http://dx.doi.org/10.1007/3-540-45023-8_13`

Tarjan, R., 1972. Depth-first search and linear graph algorithms. SIAM journal on computing 1 (2), 146–160.

Ter Horst, H., oct 2005. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. Web Semantics: Science, Services and Agents on the World Wide Web 3 (2-3), 79–115.
URL `http://dx.doi.org/10.1016/j.websem.2005.06.001`

Thuraisingham, B., 2007. Security and privacy for multimedia database management systems. Multimedia Tools and Applications 33 (1), 13–29.
URL `http://dx.doi.org/10.1007/s11042-006-0096-1`

Toninelli, A., Bradshaw, J., Kagal, L., Montanari, R., November 2005. Rule-based and ontology-based policies: Toward a hybrid approach to control agents in pervasive environments. In: Proceedings of the Semantic Web and Policy Workshop.

Toninelli, A., Corradi, A., Montanari, R., 2009. A quality of context-aware approach to access control in pervasive environments. In: Bonnin, J.-M., Giannelli, C., Magedanz, T. (Eds.), MobileWireless Middleware, Operating Systems, and Applications. Vol. 7 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, pp. 236–251.
URL `http://dx.doi.org/10.1007/978-3-642-01802-2_18`

Toninelli, A., Montanari, R., Kagal, L., Lassila, O., 2006. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (Eds.), The Semantic Web - ISWC 2006. Vol. 4273 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 473–486.
URL `http://dx.doi.org/10.1007/11926078_34`

Toninelli, A., Montanari, R., Kagal, L., Lassila, O., June 2007. Proteus: A semantic context-aware adaptive policy model. In: Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on. pp. 129–140.

Udrea, O., Recupero, D. R., Subrahmanian, V. S., Jan. 2010. Annotated rdf. ACM Transactions on Computational Logic 11 (2), 10:1–10:41.
URL `http://doi.acm.org/10.1145/1656242.1656245`

Ullman, J. D., 1988. Principles of Database and Knowledge-base Systems, Vol. I. Computer Science Press, Inc., New York, NY, USA.

Uszok, A., Bradshaw, J., Hayes, P., Jeffers, R., Johnson, M., Kulkarni, S., Breedy, M., Lott, J., Bunch, L., 2003a. Daml reality check: A case study of kaos domain and policy services. In: The International Semantic Web Conference (ISWC 03).

Uszok, A., Bradshaw, J., Jeffers, R., 2004a. Kaos: A policy and domain services framework for grid computing and semantic web services. In: Jensen, C., Poslad, S., Dimitrakos, T. (Eds.), Trust Management. Vol. 2995 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 16–26.
URL http://dx.doi.org/10.1007/978-3-540-24747-0_2

Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., Lott, J., 2003b. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks. POLICY '03. IEEE Computer Society, pp. 93–.
URL http://dl.acm.org/citation.cfm?id=826036.826850

Uszok, A., Bradshaw, J., Jeffers, R., Tate, A., Dalton, J., 2004b. Applying kaos services to ensure policy compliance for semantic web services workflow composition and enactment. In: McIlraith, S., Plexousakis, D., van Harmelen, F. (Eds.), The Semantic Web - ISWC 2004. Vol. 3298 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 425–440.
URL http://dx.doi.org/10.1007/978-3-540-30475-3_30

Uszok, A., Bradshaw, J., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S., Jul 2004c. Kaos policy management for semantic web services. Intelligent Systems, IEEE 19 (4), 32–41.

Uszok, A., Bradshaw, J. M., Jeffers, R., Johnson, M., Dalton, A. T. J., 2004d. Policy and contract management for semantic web services. In: Systems. Stanford University. AAAI Press.

Villata, S., Delaforge, N., Gandon, F., Gyrard, A., 2011. An access control model for linked data. In: On the Move to Meaningful Internet Systems: OTM 2011 Workshops. pp. 454–463.

W3C OWL Working Group, December 2012. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C recommendation, available at http://www.w3.org/TR/owl-features/, W3C.

W3C SPARQL Working Group, Jan 2013. SPARQL 1.1 Overview. W3C recommendation, available at http://www.w3.org/TR/sparql11-overview/, W3C.

Wang, Q., Yu, T., Li, N., Lobo, J., Bertino, E., Irwin, K., Byun, J.-W., 2007. On the correctness criteria of fine-grained access control in relational databases. In: Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB '07. VLDB Endowment, pp. 555–566.
URL http://dl.acm.org/citation.cfm?id=1325851.1325915

Weitzner, D. J., Hendler, J., Berners-Lee, T., Connolly, D., 2006a. Creating a policy-aware web: Discretionary, rule-based access. Web and Information Security, 1.

Weitzner, D. J., Hendler, J., Berners-Lee, T., Connolly, D., 2006b. Creating a policy-aware web: Discretionary, rule-based access. IGI Global, p. 1.

Wielemaker, J., Huang, Z., van der Meij, L., 2008. SWI-Prolog and the Web. Theory and Practice of Logic Programming 8 (3), 363–392.

Yagüe, M. I., Maña, A., López, J., Troya, J. M., 2003. Applying the semantic web layers to access control. In: Proceedings of the 14th International Workshop on Database and Expert Systems Applications. DEXA '03. IEEE Computer Society, Washington, DC, USA, pp. 622–.
URL http://dl.acm.org/citation.cfm?id=942790.942921

Zimmermann, A., Lopes, N., Polleres, A., Straccia, U., Mar. 2012. A general framework for representing, reasoning and querying with annotated semantic web data. Vol. 11. Elsevier, pp. 72–95.
URL `http://dx.doi.org/10.1016/j.websem.2011.08.006`

List of Definitions

List of Examples

List of Figures

List of Queries

List of Rules

List of Tables

List of Algorithms

List of Acronyms

ABAC Attribute Based Access Control

ACD Access Control Domain

ACL Access Control List

ACS Access Control Statement

APPEL A P3P Preference Exchange Language

aRDF Annotated RDF

aRDFS Annotated RDF Schema

ASCII American Standard Code for Information Interchange

BAP Basic Annotated Pattern

BBC British Broadcasting Corporation

BSBM Berlin SPARQL Benchmark

BGP Basic Graph Pattern

CBAC Context Based Access Control

CIM Common Information Model

CLAC Concept Level Access Control

CRM Customer Relationship Management

CRUD Create, Read, Update and Delete

DAML DARPA Agent Markup Language

DMS Document Management System

DTD Document Type Definition

DAC Discretionary Access Control

FAF Flexible Authorisation Framework

FOAF Friend Of A Friend

FGAC Fine Grained Access Control

G-FAF Graph-based Flexible Authorisation Framework

H-FAF Hierarchical Flexible Authorisation Framework

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HR Human Resources

IRI Internationalised Resource Identifier

JSON JavaScript Object Notation

MAC Mandatory Access Control

N3 Notation 3

LDP Linked Data Platform

LDW Linked Data Web

LinDAA Linked Data Authorisation Architecture

LOB Line of Business

OASIS Advanced Open Standards for the Information Society

OEM Object Exchange Model

OIL Ontology Inference Layer

OWL Web Ontology Language

P3P Platform for Privacy Preferences

PeLDS Policy-enabled Linked Data Server

PPM Privacy Preferences Manager

PPO Privacy Preferences Ontology

PSPL Portfolio and Service Protection Language

R2RML RDB to RDF Mapping Language

RBAC Role Based Access Control

RDB Relational Database

RDF Resource Description Framework

RDFa Resource Description Framework in Attributes

RDFS RDF Schema

RDB2RDF Relational Database to Resource Description Framework

SACL Semantic Access Control Language

SCBAC Semantic-aware Context-based Access Control Model

SQL Structured Query Language

SSL Secure Sockets Layer

SWRL Semantic Web Rule Language

TLS Transport Layer Security

TP Triple Pattern

URI Uniform Resource Identifier

VBAC View Based Access Control

WAC Web Access Control

W3C World Wide Web Consortium

WebID Web Identity and Discovery

WS-Policy Web Services Policy Framework

WWW World Wide Web

XACML eXtensible Access Control Markup Language

XML Extensible Markup Language