Title	A Logic Programming approach for Access Control over RDF
Author(s)	Lopes, Nuno; Kirrane, Sabrina; Mileo, Alessandra; Zimmermann, Antoine; Polleres, Axel
Publication Date	2012
Publication Information	Lopes, Nuno; Kirrane, Sabrina; Mileo, Alessandra (2012) A Logic Programming approach for Access Control over RDF International Conference on Logic Programming (ICLP), Technical Communications
Publisher	International Conference on Logic Programming (ICLP)
Link to publisher's version	http://www.deri.ie/sites/default/files/publications/a_logic_programming_approach_for_access_control_over_rdf.pdf
Item record	http://hdl.handle.net/10379/4557

A Logic Programming approach for Access Control over RDF

Nuno Lopes
Digital Enterprise Research Institute
email: nuno.lopes@deri.org

Sabrina Kirrane
Digital Enterprise Research Institute and Storm Technology
email: sabrina.kirrane@deri.org

Antoine Zimmermann
École des Mines de Saint-tienne
email: antoine.zimmermann@emse.fr

Axel Polleres
Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria
email: axel.polleres@siemens.com

Alessandra Mileo
Digital Enterprise Research Institute
email: alessandra.mileo@deri.org

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

The Resource Description Framework (RDF) is an interoperable data representation format suitable for interchange and integration of data, especially in Open Data contexts. However, RDF is also becoming increasingly attractive in scenarios involving sensitive data, where data protection is a major concern. At its core, RDF does not support any form of access control and current proposals for extending RDF with access control do not fit well with the RDF representation model. Considering an enterprise scenario, we present a modelling that caters for access control over the stored RDF data in an intuitive and transparent manner. For this paper we rely on Annotated RDF, which introduces concepts from Annotated Logic Programming into RDF. Based on this model of the access control annotation domain, we propose a mechanism to manage permissions via application-specific logic rules. Furthermore, we illustrate how our Annotated Query Language (AnQL) provides a secure way to query this access control annotated RDF data.

KEYWORDS: Logic Programming, Annotation, Access Control, RDF

1 Introduction

Enterprises rely on stand-alone systems, commonly known as Line Of Business (LOB) applications, to efficiently perform day-to-day activities: interactions with clients in a Customer Relationship Management (CRM) application, employee information in a Human Resources (HR) application, project documentation in a Document Management System (DMS), etc. 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 for representing such integrated data, however it does not provide any mechanisms to avoid the problem of information leakage. In this paper we rely on an integration solution that extracts information from the underlying LOB applications into RDF. Based on this integrated data, we define a mechanism to enforce access control over the resulting RDF graph, implemented via logic programming. Our approach provides a flexible representation for the access control policies and also caters for permission propagation via logic inference rules.

The solution we present builds upon an extension of the RDF data model to supply context information (called Annotated RDF), that provides a backwards compatible model to attach domain-specific metadata to each RDF triple. Our contribution in relation to access control over RDF data focuses on: (i) defining an annotation domain that models access control permissions in RDF; (ii) specifying the high-level system architecture required to enforce access control by relying on SPARQL, the standard query language for RDF; and (iii) illustrating how domain specific rules can be used to manage the access control annotations. Although in this paper we are considering that the access control annotated data stems from the integration of the data from the LOB applications, the presented model can be applied as a general model for access control in RDF, without requiring the information integration step.

The remainder of the paper is structured as follows: in Section 2 we briefly introduce concepts from the Semantic Web research area and their extension to the annotated case. Section 3 formalises the access control annotation domain and details our implementation of the domain in logic programming. Section 4 discusses how our formalism can be used to extend RDF with access control and provides a high-level overview of a system that enforces such access control. Finally, we describe the related work in Section 5 and present conclusions and directions for future work in Section 6.

2 Preliminaries

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 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 prototype implementation of Annotated RDFS and AnQL which is implemented in SWI Prolog.

2.1 Annotated RDFS Data Model

We present an overview of the concepts of RDF and its extension to Annotated RDFS.

Definition 1 (*RDF triple, RDF graph*)
Considering the disjoint sets **U**, **B** and **L**, representing respectively URIs, blank nodes and literals, an *RDF triple* is a tuple $(s, p, o) \in \mathbf{UB} \times \mathbf{U} \times \mathbf{UBL}$,[1] where s is called the *subject*, p the *predicate*, and o the *object*. An *RDF graph* G is a finite set of RDF triples.

An RDF triple has the intuitive meaning that the *subject* is connected to the *object* by the *predicate* relation. In this work, we avoid introducing details about the concrete syntaxes of RDF, and we omit minutiae. Please refer to Manola and Miller (2004) and Hayes (2004) for specifics.

Several extensions were presented 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; 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 Straccia et al. (2010). 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 $\oplus$ (cf Definition 2) operations defined by the annotation domain, can be specified in a *domain independent* fashion. Next we present the definition of an annotation domain while the Annotated RDFS inference rules are detailed in Section 2.2.

Definition 2 (*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 $\lambda, \lambda_1, \lambda_2 \in L$:

1. $\oplus$ is idempotent, commutative, associative;
2. $\otimes$ is commutative and associative;
3. $\bot \oplus \lambda = \lambda$, $\top \otimes \lambda = \lambda$, $\bot \otimes \lambda = \bot$, and $\top \oplus \lambda = \top$;
4. $\otimes$ 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 $\lambda_1 \oplus \lambda_2 = \lambda_2$.

[1] For conciseness, we represent the union of sets simply by concatenating their names.

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

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

For the definitions of other domains, such as the temporal domain, the reader is referred to Straccia et al. (2010). In Section 3.1 we present the definition of an annotation domain to model access control.

Further to the above annotation domain definition, we extend RDF towards annotated RDFS:

Definition 3 (*Annotated triple, 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 Straccia et al. (2010).

2.2 Inference Rules

RDF Schema (RDFS) (Brickley and Guha 2004) consists of a predefined vocabulary that assigns specific meaning to certain URIs, 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 2). Such rules can be represented by the folowing meta-rule:

$$\frac{\tau_1\colon \lambda_1,\ \ldots,\ \tau_n\colon \lambda_n, \{\tau_1, \ldots \tau_n\} \vdash_{\mathsf{RDFS}} \tau}{\tau\colon \bigotimes_i \lambda_i} \ . \tag{1}$$

This rule reads that 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. The $\oplus$ operation is 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:

$$\frac{\tau\colon \lambda_1,\ \tau\colon \lambda_2}{\tau\colon \lambda_1 \oplus \lambda_2} \ . \tag{2}$$

It is also possible to specify a custom set of rules in order to provide application specific inferencing. One of the contributions of this paper, the definition of custom rules for managing permissions in the access control domain, is presented in Section 4.1.

2.3 AnQL: Annotated Query Language

The proposed query language for Annotated RDFS is AnQL (Lopes et al. 2010), which consists of an extension to the W3C recommended query language for RDF, SPARQL (Seaborne and Prud'hommeaux 2008), while also taking into consideration features from the upcoming SPARQL 1.1 language revision. Consider $\mathbf{V}$ a set of variables disjoint from $\mathbf{UBL}$. 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 $\mu\colon \mathbf{V} \to \mathbf{UBL}$.

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 $\mathbf{UBLV}$ and λ be an annotation value from 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).

An *AnQL query* is defined as a triple $Q = (P, G, V)$ where: (1) P is an annotated graph pattern; (2) G is an annotated RDF graph; and (3) $V \subseteq \mathbf{VA}$ 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 2, the annotated graph pattern P is specified following the WHERE keyword where the variables are specified after the SELECT keyword.

Example 2 (*AnQL query*)
Considering the fuzzy domain presented in Example 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 (*BAP matching, extends (Pérez et al. 2009, Definition 2)*)
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 Straccia et al. (2010).

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) and a combined overview of Annotated RDFS and AnQL is provided by Zimmermann et al. (2012).

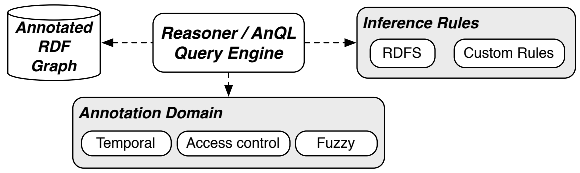

Fig. 1. Annotated RDFS implementation schema

2.4 Implementation

The system architecture of our prototype implementation, based on SWI-Prolog's Semantic Web library (Wielemaker et al. 2008), is sketched in Figure 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** (Straccia et al. 2010) 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 abstracts the specific details of each domain. An example of an Annotated RDFS rule is presented in Example 3.

Example 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 (as presented in Definition 2).

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

3 Access Control Annotation Domain

In this section we formalise our access control annotation domain, following the definitions presented in Section 2.1, starting by defining the entities and annotation values and then presenting the $\otimes$ and $\oplus$ domain operations. Finally, we briefly describe the implementation of the presented annotation domain.

3.1 Entities and Annotations

For the modelling of the Access Control Domain (ACD) consider, in addition to the previously presented sets of URIs **U**, blank nodes **B**, and literals **L**, a set of credential elements **C**. The elements of **C** are used to represent *usernames*, *roles*, and *groups*. To represent *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 $\mathbf{T}$.[2] We allow shortcuts to represent intervals of integers, for example "$(:age, [25, 30])$" to indicate that all entities with attribute "$:age$" between 25 and 30 are allowed access to the triple.

Considering an element $e \in \mathbf{CT}$, e and $\neg e$ are *access control elements*, where e is called a *positive* element and $\neg e$ is called a *negative* element.[3] 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 $e \in \mathbf{CT}$, 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 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 (*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:

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

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 $access \notin \mathbf{CT}$ 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 $\mathbf{CT}$ 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.

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

[3] 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.

Example 5 (*Datalog Representation of an ACL*)
Taking into account the annotation example presented in Example 4. The nr-datalog$^\neg$ program corresponding to the ACL $[[it], [hr, \neg js]]$ is:

$$access \leftarrow it.$$

$$access \leftarrow hr, \neg js.$$

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 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), the following rules can be added to the nr-datalog$^\neg$ program from Example 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.

These rules can be used not only to express hierarchies between entities but any form of nr-datalog$^\neg$ rules are allowed.

3.2 Annotation Domain

We now turn to the annotation domain operations $\otimes$ and $\oplus$ that, as presented in Section 2.2, allow for the combination of annotation values catering for RDFS inferences. A naive implementation of these domain operations may produce ACLs which 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* (allow access); or (ii) *safe conflict resolution* (deny access). This is achieved during the normalisation step, through the *resolve* function, by removing the appropriate element: $\neg jb$ for brave or jb for safe conflict resolution. In our current modelling, we are assuming safe conflict resolution. The normalisation process is defined as follows:

Definition 5 (*Normalise*)
Let A be an ACL. We define the reduction of A into its consistent form, denoted $norm(A)$, as:

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

The $\oplus$ operation is used to combine annotations when the same triple is deduced from different inference steps (cf. Rule (2)). 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$^\neg$ program consisting of the union of all the rules from the original nr-datalog$^\neg$ programs. Formally,

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

The following example presents an application of the $\oplus_{ac}$ operation:

Example 7 ($\oplus_{ac}$ *operation*)
Consider the annotated triples $\tau_1 = (\texttt{:johnSmith}, \texttt{:salary}, \texttt{40000})\colon [[js]]$ and $\tau_2 = (\texttt{:johnSmith}, \texttt{:salary}, \texttt{40000})\colon [[hr]]$. Combining these triples with the $\oplus_{ac}$ operation (by applying Rule (2)) should result in providing access to all the entities which are allowed to access the premises:

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

In turn, the $\otimes$ operation is used when inferring new triples, with the application of Rule (1), and for the access control domain, this operation ($\otimes_{ac}$) consists of merging the rules belonging to both annotation programs and then performing the 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. Formally, the $\otimes$ operations corresponds to:

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

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 8 ($\otimes_{ac}$ *operation*)
Consider the triples $\tau_1 = (\texttt{:westportCars}, \texttt{:netIncome}, \texttt{1000000})\colon [[hr, \neg jb]]$ and $\tau_2 = (\texttt{:netIncome}, \mathsf{dom}, \texttt{:Company})\colon [[it, jb]]$. 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 premisses. Thus we can infer, not only that `:westportCars` is of type `:Company`, but also the appropriate annotation value:

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

Please note that the aforementioned conflict resolution mechanism has simplified $[\neg jb, jb]$ into $[\neg jb]$.

Lastly, 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 $e \in \mathbf{CT}$: $\bot_{ac} = []$ 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 $\perp_{ac}$ corresponds to an empty program. However, for practical reasons, it might be necessary to assume a "super-user" role, for example represented as the reserved element "su", which will be allowed access to all triples and therefore would be used as the $\perp_{ac}$ annotation. For this paper we are mostly ignoring this issue, and simply assume these issues can be handled by domain specific rules (as presented in Section 4.1).

Definition 6 (*Access Control Annotation Domain*)
Let **F** be the set of annotation values over **CT**, i.e. consistent ACLs. The access control annotation domain is formally defined as:

$$D_{ac} = \langle \mathbf{F}, \oplus_{ac}, \otimes_{ac}, \perp_{ac}, \top_{ac} \rangle \ .$$

The presented modelling of the access control domain 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 would not be attached to any specific triple but rather as a graph-wide permission and thus is out of scope for this modelling. In this paper, we are considering only *read* permissions in the description of the domain and thus restrict the modelling to a single access control list. It is worth noting that the support for *create* and *update* of RDF is only included in the forthcoming W3C SPARQL 1.1 Recommendation (Harris and Seaborne 2012).

3.3 Prolog Implementation

Considering the prototype described in Section 2.4, the implementation of the access control annotation domain 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. The annotation values are represented by 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 2, where the salary information is only available to the respective employee. In this figure we are representing the RDF triples and annotation element

```
@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]].
```

Fig. 2. RDF triples annotated with access control permissions

using the NQuads RDF serialisation.[4] Using AnQL, the extension of the SPARQL query language described in Section 2.3, 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 9 (*AnQL Query Example*)
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 2) under SPARQL semantics, i.e. if the annotation was omitted, would be:

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

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:

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

4 Access Control Framework

In order to provide a complete framework to handle authorisation in RDF, the domain modelling presented in the previous section needs to be part of a system that is capable of enforcing the access control policies. Figure 3 provides a high level overview of the individual components required for such a system. In our enterprise data integration use-case, the **Integration Service** takes care of translating the underlying data and access control mechanisms from the LOB applications into the annotated RDF data model. However, our modelling is flexible enough to cater for scenarios where permissions cannot be extracted from the underlying databases.

The **Access Control Layer** corresponds to the system architecture presented in Figure 1, where we fix the annotation domain as the presented access control domain. The data and the access permissions are stored in the Annotated RDF

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

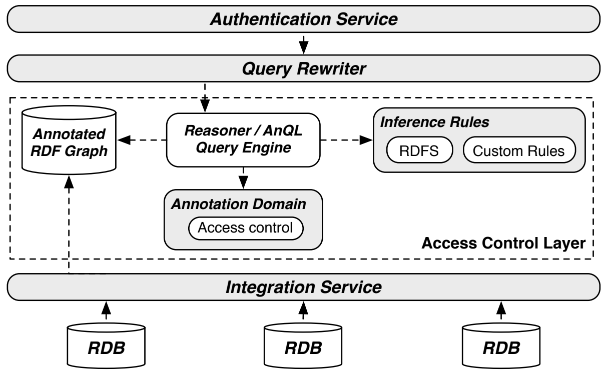

Fig. 3. Access Control System Architecture

Graph and we rely on the existing Annotated RDF reasoner and query engine (described in Section 2.4) for evaluating queries and domain-specific rules.

The **Authentication Service** returns the user credentials that are composed of one or more *usernames*, *roles*, *groups* and *attributes* that may also be extracted from each of the LOB applications. The **Query Rewriter** takes a query specified using the SPARQL query language for RDF and, using the list of credentials provided by the **Authentication Service**, expands it into an AnQL query, which ensures that users only have access to the information they have been granted access to. We also extract end user credentials from the LOB applications, however as authentication is outside the scope of this paper, we simply assume a mapping between the enterprise employee and their usernames, groups and roles.

Next we present some of the challenges that exist in the proposed system architecture: how to handle permission management and propagation and how to ensure that users cannot circumvent the access control policies.

4.1 Rules for Permission Management

In many LOB applications, two forms of implicit access control are present: (i) hierarchies between entities in the access control annotations; and (ii) restrictions over resources in the data, for example collections of relational tuples or RDF triples. Hierarchies of form (i) were addressed in Section 3.1 by adding rules to the nr-datalog$^\neg$ program that evaluates the annotations. As for (ii), permissions granted to a resource should be propagated to all related triples. Such propagation chains can be broken by explicitly specifying permissions on a particular triple.

Taking into consideration our access control domain modelling and the use-case of extracting data (and permissions) from their original sources, one option is to incorporate this business logic into the Extract-Transform-Load (ETL) process. However, to handle permission management in RDF stores, it is not possible to rely on the ETL process. Therefore we propose to use domain specific rules which our reasoner is capable of processing in order to propagate the access permissions or to enforce any domain specific policies. Such rules can be written similarly to the

Annotated RDF rules, described in Section 2.4, giving us access to the existing data and annotations and allowing us to create new Annotated RDF triples or update existing ones.

Example 10 (*Domain Specific Rule*)
In an enterprise scenario, if an employee is given access to a Company record, as per the following triple (C, `type`, `:Company`), that employee should be given access to all triples relating to the Company.

$$\frac{(C, \texttt{type}, \texttt{:Company})\colon \lambda_1,\ (C, P, O)\colon \lambda_2}{(C, P, O)\colon \lambda_1 \oplus_{ac} \lambda_2}\ ,$$

where C, P, O and λ_1, λ_2 are variables. Applying this rule to the sample dataset presented in Figure 2, would cause the access permission of the triple $(\texttt{:westportCars}, \texttt{type}, \texttt{:Company})\colon [[jb]]$ to be propagated to the second triple, yielding the following new annotated triple: $(\texttt{:westportCars}, \texttt{:netIncome}, \texttt{1000000})\colon [[jb]]$.

4.2 Transparent Access Control

This section describes issues that need to be addressed when considering extending RDF with access control information. The first is regarding SPARQL query injection, where users should be prevented from specifying any access credentials manually while the second addresses the problem of interchanging RDF graphs annotated with access control information.

It is possible to use AnQL directly to query RDF data annotated with access control information, as presented in Example 9. However, allowing users to perform AnQL queries is not secure since they could bypass the access control due to the lack of enforcement of the supplied credentials. Our proposed solution for the enforcement of the access control is based on query rewriting. The user is allowed to write SPARQL queries and the system transparently extends each triple pattern of the query with the user credentials given as an annotation value, thus generating an AnQL query. The AnQL query is then executed against the Annotated RDF graph, which guarantees that the user can only access the triples appropriate for the provided credentials. This query rewriting step relies on the credentials provided by the external authentication system, which are represented as an annotation control element and thus can be easily added into any SPARQL Basic Graph Pattern to obtain an AnQL Basic Annotated Pattern. For this paper we are not considering any specific implementation of the authentication system but rather assume a secure implementation that relies, for example, on WebID (Sporny et al. 2011).

Another issue is maintaining the support for the interchangeability of RDF graphs, where RDF graphs can be serialised using one of the RDF representations. For the access control domain, we are interested in ensuring that each user is only allowed access to authorised information. In such scenario, the information contained in an RDF store should not be *dumped* as Annotated RDF but rather as a standard RDF graph containing the data the user is allowed access. The proposed modelling should be used as an internal representation for the access control information in RDF stores and should not be exposed to the end-user.

5 Related Work

The topic of access control has been studied in different research areas, here we give a small overview of related works in the database community and then present further comparisons in the RDF realm, that are analogous to our approach.

The topic of access control has been long studied in relational databases and the approach of enforcing access policies by query rewriting was also considered for the Quel query language by Stonebraker and Wong (1974). However, the presented system does not rely on annotating the relational data but rather access control is specified using constraints over the user credentials which are then included in the rewritten query. A good overview of common issues, existing models and languages for access control is provided by di Vimercati et al. (2005), who focus on topics also discussed in this paper such as user hierarchy, allowing and denying access and conflict resolution.

For the Semantic Web, well known policy languages such as KAoS (Bradshaw et al. 1997), Rei (Kagal and Finin 2003) and PROTUNE (Bonatti et al. 2009) are based on logical formalisms and consequently have well defined semantics. Although such policy languages enable policy specification using semantic web languages in their current form, they do not support reasoning based on RDF data relations. These policy languages are complementary to our work as their policies could be mapped to our annotations using rules.

In contrast, Javanmardi et al. (2006), Ryutov et al. (2009), and Amini and Jalili (2010) propose access control models for RDF graphs and like us allow for policy propagation and inference based on semantic relations. The policy language proposed by Javanmardi et al. (2006) 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.

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

Similar access control annotations are attached to axioms in an ontology by Knechtel and Stuckenschmidt (2010) and Baader et al. (2009) in order to allow access to subsets of the ontology to specific users, or applied to the problem of 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 paper, some of the algorithms for efficient annotation calculation may be ported to our modelling.

6 Conclusions and Future Work

The Resource Description Framework (RDF) can be used for large scale integration of information from existing LOB applications. In this paper, we propose an access control model that can be used to protect RDF data and demonstrate how a combination of Annotated RDF and SPARQL can be used to control access to integrated enterprise data. Our model is based on the previously proposed Annotated RDF framework and attaches the access control information on a triple basis i.e. each RDF triple can contain different annotation values. The proposed solution provides a flexible representation method for the access control annotations, based on access control rules that define which entities have access to the triple. However, on very large datasets, challenges will arise with respect to optimal access control policy administration. To tackle this issue we propose managing permissions by specifying domain-specific inference rules for the annotation domain. We also suggest a possible implementation structure for a framework to enforce the access control based on rewriting a SPARQL query into an Annotated SPARQL query (AnQL) which relies on a secure authentication service.

Our initial work touches on how rules can be used to simplify the management of RDF access control permissions. In future work, we propose to investigate the interdependencies between usernames, groups, roles, and attributes and how we can further exploit the RDF graph structure to streamline the management of RDF access control policies. Although the modelling presented in this paper provides a suitable representation model for the annotation values, its implementation and evaluation for large RDF graphs remains an open issue. To provide acceptable query performance when compared to its non-annotated counterpart, different optimisation strategies for both annotation storage and query evaluation will be necessary.

Acknowledgements This work is supported in part by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2), the Irish Research Council for Science, Engineering and Technology Enterprise Partnership Scheme and Storm Technology Ltd. We would like to thank Gergely Lukácsy, Aidan Hogan, and Umberto Straccia for their comments on this paper.

References

ABEL, F., COI, J. L. D., HENZE, N., KOESLING, A. W., KRAUSE, D., AND OLMEDILLA, D. 2007. Enabling Advanced and Context-Dependent Access Control in RDF Stores. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, K. Aberer, K.-S. Choi, N. F. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds. Vol. 4825. Springer, 1–14.

AMINI, M. AND JALILI, R. 2010. Multi-level authorisation model and framework for distributed semantic-aware environments. *IET Information Security 4,* 4, 301.

BAADER, F., KNECHTEL, M., AND PEÑALOZA, R. 2009. A Generic Approach for Large-Scale Ontological Reasoning in the Presence of Access Restrictions to the Ontology's Axioms. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, Eds. Vol. 5823. Springer, 49–64.

BONATTI, P., DE COI, J., OLMEDILLA, D., AND SAURO, L. 2009. Rule-based policy representations and reasoning. In *Semantic techniques for the web*. 201–232.

BRADSHAW, J., DUTFIELD, S., BENOIT, P., AND WOOLLEY, J. 1997. KAoS: Toward an industrial-strength open agent architecture. In *Software Agents*. 375–418.

BRICKLEY, D. AND GUHA, R. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C. February. Available at `http://www.w3.org/TR/rdf-schema/`.

DI VIMERCATI, S. D. C., SAMARATI, P., AND JAJODIA, S. 2005. Policies, Models, and Languages for Access Control. In *Databases in Networked Information Systems, 4th International Workshop, DNIS 2005, Aizu-Wakamatsu, Japan, March 28-30, 2005, Proceedings*, S. Bhalla, Ed. Vol. 3433. Springer, 225–237.

DIETZOLD, S. AND AUER, S. 2006. Access Control on RDF Triple Stores from a Semantic Wiki Perspective. In *Proc. of 2nd Workshop on Scripting for the Semantic Web at ESWC, Budva, Montenegro.*, C. Bizer, S. Auer, and L. Miller, Eds. Vol. 183.

GUTIERREZ, C., HURTADO, C. A., AND VAISMAN, A. A. 2007. Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering 19,* 2 (February), 207–218.

HARRIS, S. AND SEABORNE, A. 2012. SPARQL 1.1 Query Language. W3C working draft, W3C. Jan. Available at `http://www.w3.org/TR/2012/WD-sparql11-query-20120105/`.

HAYES, P. 2004. RDF Semantics. W3C Recommendation, W3C. February. Available at `http://www.w3.org/TR/rdf-mt/`.

HOLLENBACH, J., PRESBREY, J., AND 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)*, T. Tudorache, G. Correndo, N. Noy, H. Alani, and M. Greaves, Eds. Vol. 514. CEUR-WS.org.

JAVANMARDI, S., AMINI, M., JALILI, R., AND GANJISAFFAR, Y. 2006. SBAC: A Semantic Based Access Control Model. In *11th Nordic Workshop on Secure IT-systems (NordSec'06), Linkping, Sweden*.

KAGAL, L. AND 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, 63–74.

KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of Generalized Annotated Logic Programming and its Applications. *J. Log. Program. 12,* 3&4, 335–367.

Knechtel, M. and Stuckenschmidt, H. 2010. Query-Based Access Control for Ontologies. In *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings*, P. Hitzler and T. Lukasiewicz, Eds. Vol. 6333. Springer, 73–87.

Lopes, N., Polleres, A., Straccia, U., and Zimmermann, A. 2010. AnQL: SPARQLing Up Annotated RDF. In *Proceedings of the International Semantic Web Conference (ISWC-10)*. Number 6496 in LNCS. Springer-Verlag, 518–533.

Manola, F. and Miller, E. 2004. RDF Primer. W3C Recommendation, `http://www.w3.org/TR/rdf-primer/`, W3C. February.

Pérez, J., Arenas, M., and Gutiérrez, C. 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems 34,* 3, 1–45.

Ryutov, T., Kichkaylo, T., and Neches, R. 2009. Access Control Policies for Semantic Networks. In *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 150–157.

Seaborne, A. and Prud'hommeaux, E. 2008. SPARQL Query Language for RDF. W3C Recommendation, W3C. Jan. 15. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

Sporny, M., Inkster, T., Story, H., Harbulot, B., and Bachmann-Gmr, R. 2011. WebID 1.0 - Web Identification and Discovery. W3C working draft, W3C. Nov. Available at `http://www.w3.org/2005/Incubator/webid/spec/`.

Stonebraker, M. and Wong, E. 1974. Access control in a relational data base management system by query modification. In *Proceedings of the 1974 annual conference - Volume 1*. ACM '74. ACM, New York, NY, USA, 180–186.

Straccia, U. 2009. A Minimal Deductive System for General Fuzzy RDF. In *RR*, A. Polleres and T. Swift, Eds. Vol. 5837. Springer, 166–181.

Straccia, U., Lopes, N., Lukacsy, G., and Polleres, A. 2010. A General Framework for Representing and Reasoning with Annotated Semantic Web Data. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, M. Fox and D. Poole, Eds. AAAI Press.

ter Horst, H. J. 2005. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *J. Web Sem. 3,* 2-3, 79–115.

Udrea, O., Recupero, D. R., and Subrahmanian, V. S. 2010. Annotated RDF. *ACM Trans. Comput. Logic 11,* 2, 1–41.

Wielemaker, J., Huang, Z., and van der Meij, L. 2008. SWI-Prolog and the Web. *Theory and Practice of Logic Programming 8,* 3, 363–392.

Zimmermann, A., Lopes, N., Polleres, A., and Straccia, U. 2012. A general framework for representing, reasoning and querying with annotated Semantic Web data. *Web Semantics: Science, Services and Agents on the World Wide Web 11,* 0, 72 – 95.