

Protect Your RDF Data!

Sabrina Kirrane^{1,2}, Nuno Lopes¹, Alessandra Mileo¹, and Stefan Decker¹

¹ Digital Enterprise Research Institute
National University of Ireland, Galway

<http://www.deri.ie>
{firstname.lastname}@deri.ie

² Storm Technology, Ireland
<http://www.storm.ie>

Abstract. The explosion of digital content and the heterogeneity of enterprise content sources have pushed existing data integration solutions to their boundaries. Although RDF can be used as a representation format for integrated data, enterprises have been slow to adopt this technology. One of the primary inhibitors to its widespread adoption in industry is the lack of fine grained access control enforcement mechanisms available for RDF. In this paper, we provide a summary of access control requirements based on our analysis of existing access control models and enforcement mechanisms. We subsequently: (i) propose a set of access control rules that can be used to provide support for these models over RDF data; (ii) detail a framework that enforces access control restrictions over RDF data; and (iii) evaluate our implementation of the framework over real-world enterprise data.

1 Introduction

Data on the web and within the enterprise is continuously increasing and despite advances in Information Technology (IT), it is still difficult for employees to find relevant information in a timely manner. This problem is further magnified when related information is segregated in different software systems. Bridging the information contained in such systems is necessary to support employees in their day-to-day activities. For instance, a high-level view of a customer would enable IT support staff to quickly respond to issues raised by that customer or additional information on drug compounds would allow a pharmaceutical company to identify potential issues with a new drug early in the drug development process.

The Resource Description Framework (RDF) is a flexible format used to represent data on the web which can also be used for data integration [21]. An additional benefit is that RDF data can easily be supplemented with complementary information from Linked Open Data (LOD) sources. It is possible to use Relational Database to RDF (RDB2RDF) techniques to translate existing relational data into RDF. However, given security is extremely important to enterprises we cannot simply extract the data and ignore the access control policies

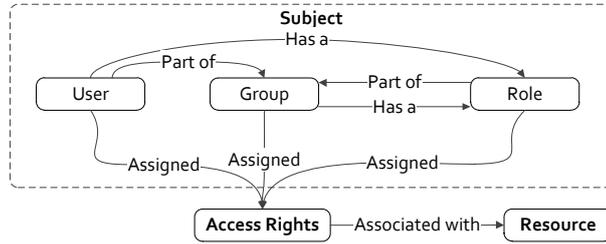


Fig. 1. Components of an access control statement

that have been placed on that data. Although we could extend RDB2RDF to extract both the data and the access control information, RDF still does not have a mechanism to enforce existing access control policies over RDF.

Javanmardi et al. [9], Ryutov et al. [16], Amini and Jalili [1], Costabello et al. [4], Sacco et al. [17] all propose mechanisms to supplement RDF with access control. However, each of the authors adopt a top-down approach 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 based on existing software engineering and database access control approaches. Based on this analysis, we make the following contributions (i) identify the core access control models that need to be supported; (ii) propose a set of rules that are necessary for the enforcement of these models over RDF; and (iii) present an 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 paper is structured as follows: in Section 2, we provide an overview of commonly used access control models. Section 3 describes how these models can be used in conjunction with RDF and details the rules necessary to propagate access control policies, based on existing access control models, over RDF data. The framework and our specific implementation are described and evaluated in Section 4. Finally, in Section 5 we discuss related work and we conclude and outline directions for future work in Section 6.

2 Analysis of Enterprise Access Control Models

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 authorizations and 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. Fig. 1 depicts the relationship between the access control terms that are used in this paper. A *Resource* is used

Table 1. Categorisation of access control models

<i>Enterprise</i>	<i>Data Models</i>	<i>Open Systems</i>
MAC	VBAC	ABAC
DAC	OBAC	
RBAC		

to denote the information to be protected (e.g. database records, application objects or website Uniform Resource Identifiers (URIs)). *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 (e.g. contributors, supervisors or management). *Roles* are used to assign a set of access rights to a set of individuals and groups, for example by department (human resources, sales or marketing) or task (insurance claim processing, reporting or 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.

A high level categorisation of the models discussed in this section is presented in Table. 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 open 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)* [18]. In this model access to resources is restricted through mandated policies determined by a central authority (e.g. mainframes and lightweight directory services).

M_{DAC} : *Discretionary Access Control (DAC)* [18]. Similar to MAC, access to resource is constrained by a central access control policy, however in contrast to MAC users are allowed to override the central policy (e.g. employees may be allowed to grant others access to their own resources).

M_{RBAC} : *Role Based Access Control (RBAC)* [19]. Generally speaking, RBAC involves grouping a set of access rights that describe the responsibilities or tasks that can be performed by a role (e.g. manager, sales person and clerk). RBAC is the most commonly used access control model in enterprise applications.

M_{VBAC} : *View-based Access Control (VBAC)* [8]. 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) [7] where access rights are granted to application objects.

M_{ABAC} : *Attribute Based Access Control (ABAC)* [14]. In distributed open environments where the requester is unknown prior to the submission of the request, an alternative means of granting access is required. In ABAC [14],

access control decisions are based on attributes (e.g. *employer=storm, policyNumber=565656*), in the form of digitally signed documents, that are sent by the requester to the server.

3 Extending RDF to Support Existing Access Control Models

In this section we examine how the access control models identified in the previous section can be used to protect RDF data. We propose a set of access control rules required to propagate and enforce access control policies, based on these models, over RDF data.

3.1 Annotated RDF Model

Our work relies on an extension of RDF, called Annotated RDF [22], that allows domain specific meta-information to be attached to RDF triples. The access rights are represented as annotations attached to individual RDF triples. The Annotated RDF access control domain model and the annotated RDFS inference rules are based on our previous work presented in [13] and therefore we only provide a short overview of the domain model below. For the definitions of other domains, namely the fuzzy, temporal and provenance domains, and an overview of how such domains can be combined the reader is referred to Zimmermann et al. [23].

In the following examples an NQuads [5] format is used to associate access control meta-information with a triple. However the framework itself does not dictate the use of an NQuads RDF serialisation, therefore alternative approaches such as using reification or mapping annotations to triple hash-codes would work also. Throughout the paper we assume the default prefix `http://urq.deri.org/enterprise#`.

In Annotated RDF, the domain annotations must follow a pre-defined structure that facilitates inferencing and querying. Software applications generally provide four basic functions of persistent storage *Create, Read, Update and Delete* (CRUD). Intuitively data permissions and prohibitions are modelled in a similar fashion, whereby subjects are granted CRUD access to resources. *Read, update, and delete* could be represented as a 3-tuple ACL (R, U, D), where R specifies the formula for *read* permission, U for *update* permission and D for *delete*. A *create* permission has a different behaviour as it would not be attached to any specific triple but rather to a named graph and as such enforcement would need to be handled using rules. In this paper we use separate *Access Control Lists* (ACLs) to represent the permissions as follows:

```
:TopSecret1 a :Project "(readACL, updateACL, deleteACL)"
```

However as our current implementation focuses on read access, we omit both the *updateACL* and *deleteACL* in the following examples and present only the *readACL* as "(*readACL*)". In the access control domain, each ACL is composed of one or more *Access Control Statements* (ACSSs) which in turn contain

Access Control Elements (ACEs) in the form of *usernames*, *roles*, *groups* or *attributes*. Negated elements are used to explicitly deny access to an entity, for example \neg :mary (where :mary is a username) indicates that :mary is *denied* access. If no annotation is present it is assumed that there is no access control information to be applied to the triple.

Example 1 (RDF Annotation). The following annotated triple:

```
:TopSecret1 a :Project "([[ :manager ], [¬:mary]])"
```

states that the users identified with :manager have read access to the triple however :mary is explicitly denied access. \square

To avoid duplicate and conflicting ACLs, the model relies on a normalisation procedure that checks for redundant ACSs and applies a conflict resolution mechanism if necessary. The conflict resolution is applied in scenarios where an annotation statement contains both a positive and a negative access control element for the same entity, e.g [:mary, ¬:mary]. There are two different ways to resolve conflicts in annotation statements, either apply: (i) a *brave conflict resolution* (allow access); or (ii) a *safe conflict resolution* (deny access).

3.2 Support for existing Access Control Models

In this section we demonstrate how each of the access control models, presented in 2, can be handled either by the domain model or the enforcement framework.

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 and system users to perform discretionary updates to some policies.

M_{RBAC} . RBAC can be represented using a single value element notation. The following quad demonstrate how a single annotation can 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 subjects employer can be represented using key-value pairs.

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

M_{VBAC} . VBAC is concerned with granting access to resources based on a logical abstraction as opposed to the physical representation, as such we propose a number of rules that allow us to grant access based on common data abstractions. For example, the ability to grant access to RDF data that share, a common RDF subject (RDB tuples) or the same RDF object (RDB attributes). In addition we support traditional access control mechanisms by allowing access to be granted based on hierarchical data structures.

Table 2. Overview of access control rules

	M_{MAC}	M_{DAC}	M_{RBAC}	M_{VBAC}	M_{ABAC}	$CRUD$
R1				✓		
R2			✓			
R3	✓	✓	✓	✓	✓	
R4	✓	✓	✓	✓	✓	
R5						✓

3.3 Proposed Access Control Rules

This section introduces a set of access control rules that enable the administration and the enforcement of the different models presented in the previous section. Table. 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 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 for 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 $?λ$ 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 (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, a given access control element is included in the access control list provided. 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 1. Assuming that we have access rights, denoted by λ_1 , associated with a triple. We can use this rule to propagate λ_1 to all triples with the same subject.

$$\frac{?S ?P_1 ?O_1 ?\lambda_1, ?S ?P_2 ?O_2 ?\lambda_2}{?S ?P_2 ?O_2 (? \lambda_2 \oplus_{ac} ?\lambda_1)} \quad (R1)$$

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 2. 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 $?E_2$.

$$\frac{?S ?P ?O ?\lambda_1, ?E_2 :inheritsFrom ?E_1, member(?E_1, ?\lambda_1)}{?S ?P ?O (? \lambda_1 \oplus_{ac} ?E_2)} \quad (R2)$$

Hierarchical Subjects Subsumption. Like $R2$ 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 rule differs from Rule (R2) only in the given vocabulary, 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 3. 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 $?S_2$.

$$\frac{?S_1 ?P_1 ?O_1 ?\lambda_1, ?S_2 ?P_2 ?O_2 ?\lambda_2, ?S_2 :isPartOf ?S_1}{?S_2 ?P_2 ?O_2 (? \lambda_2 \oplus_{ac} ? \lambda_1)} \quad (R3)$$

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 access rights assigned to the 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

```

we can infer that:

```

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

```

Rule 4. 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 $?O_2$.

$$\frac{?S_1 ?P_1 ?O_1 ?\lambda_1, ?S_2 ?P_2 ?O_2 ?\lambda_2, ?O_2 a ?O_1}{?S_2 ?P_2 ?O_2 (? \lambda_2 \oplus_{ac} ? \lambda_1)} \quad (R4)$$

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 5. 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.

$$\frac{?S ?P ?O (? \lambda_1, ? \lambda_2, ? \lambda_3)}{?S ?P ?O ((? \lambda_1 \oplus_{ac} ? \lambda_2 \oplus_{ac} ? \lambda_3), (? \lambda_2 \oplus_{ac} ? \lambda_3), ? \lambda_3)} \quad (R5)$$

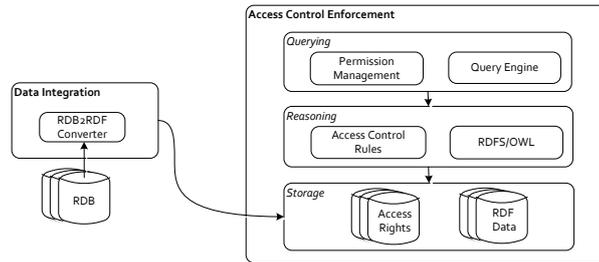


Fig. 2. RDF Data Integration and Access Control Enforcement Framework

4 Framework, Implementation and Evaluation

Based on our analysis of existing access control models this section describes the minimal set of components necessary for a data integration and access control enforcement framework. It provides an overview of our implementation and presents an experimental evaluation of our prototype which focuses on the: (i) the *RDB2RDF* data integration; (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 at improving scalability and thus do not propose any kind of optimisations.

4.1 Access Control Enforcement Framework

An overview of the proposed framework which is depicted in Fig. 2, is composed of two core 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 paper. Authentication can be achieved by relying on WebId and self-signed certificates that work 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 [6] mapping that extracts and converts the relational data into RDF. In a pure Semantic Web scenario, the data integration module is optional, in which case the data and access rights need to be populated either manually or automatically via specialised software systems.

Storage. The integrated data retrieved from the original relational databases is stored in the RDF Data store and 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 ACSs which may be stored as quads, associated with reified triples or mapped to triple hash-codes.

Reasoning. For this component we consider two distinct forms of inference: (a) data inference, where new triples are deduced from existing ones; and (b) access rights inference, where new ACSs are deduced from existing ones. For data inferencing we rely on well established forms of inference in the Semantic Web such as RDF Schema (RDFS) or the Web Ontology Language (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. In addition Rule based Access Control reasoning is required to support both permission management and data querying over RDF based on access control policies. Either backward or forward chaining could be used for both the data and the access rights inference.

Querying. SPARQL [20] 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 RDF triples the user has been granted access to either explicitly or implicitly.

4.2 Implementation

In our prototype the data integration is performed using XSPARQL [12], 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 [23] 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 paper we focus on relational databases but the ability to extract XML, from files or web-services, is desirable.

Annotated RDF. Our Prolog Annotated RDF [23] 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 [13], was provided in 3.1. The reasoning component is implemented by 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, by allowing permissions to be propagated according to the rules presented in Section 3.3. With the exception of (R5) 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 (R5) because the existing implementation only handles *read* permissions. However we are in the process of extending this prototype to support the other forms of access rights.

AnQL. Our query engine relies on an extension of SPARQL, AnQL [11], 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 paper, we simply extract end user credentials from the RDB, and maintain a mapping between the enterprise employees and their usernames, groups 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.

4.3 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 3.3 to propagate the permissions to relevant triples. In the case of the TS we only assigned existing access rights to one triple and let R1 propagate the permissions to all triples with the same subject. As for the DMS we extracted the existing URI hierarchy and the associated permissions and used R3 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 R4. Finally we input the organisation structure as facts in the Prolog application and used R2 to propagate permissions based on this hierarchy. As our prototype only supports read access we did not consider R5 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 3. Dataset description

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

Table 4. Prototype evaluation

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

Table 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 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 of the access control rules detailed in 3.3 (sec), and the number of triples inferred in this process. Fig. 3a 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 but more data intensive tests are still required. As the inferencing times are highly dependent on both the rules and the data further experimentation is required in this area.

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 ($\cancel{\exists}$). The results displayed in Table 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 Fig. 3b. However queries with a combination of annotations and either two or three triple patterns, Fig. 3c and Fig. 3d 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 experiments do not show a significant performance increase over the four

Table 5. Query execution time in seconds

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

datasets, between queries with annotations Fig. 3e and those without annotations Fig. 3f. 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.

5 Related Work

In recent years, there has been an increasing interest access control policy specification and enforcement using Semantic Technology, KAos [3], Rein [10] and PROTUNE [2] are well known works in this area. Policy languages can be categorised as general or specific. In the former the syntax caters for a diverse range of functional requirements (e.g. access control, query answering, service discovery and negotiation), whereas in contrast the latter focuses on just one functional requirement. Two of the most well-known access control languages, KAos [3] and Rei [10], are in fact general policy languages. Although the models and the corresponding frameworks are based on semantic technology the authors do not consider applying their model or framework to RDF data.

The authors of Concept-Level Access Control (CLAC) [15], Semantic-Based Access Control (SBAC) [9] and the semantic network access control models proposed by Ryutov et al. [16] and Amini and Jalili [1] respectively, all propose access control models for RDF graphs and focus on policy propagation and enforcement based on semantic relations. Qin and Atluri [15] propose policy propagation of access control based on the semantic relationships among concepts. Javanmardi et al. [9], Ryutov et al. [16] and Amini and Jalili [1] enhance the semantic relations by allowing policy propagation based on the semantic relationships between the subjects, objects, and permissions.

Costabello et al. [4] 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. [17] present a privacy preference ontology which allows for the application of fine-grained

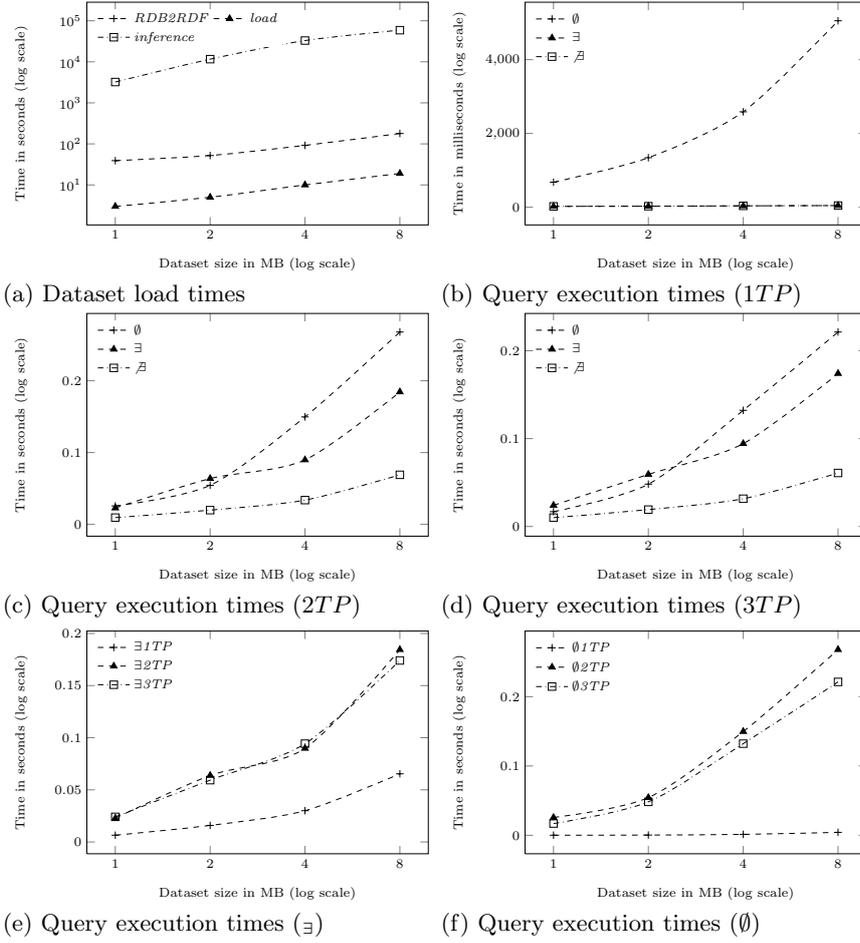


Fig. 3. Load and Query times across 4 datasets

access control policies to an RDF file. Both Costabello et al. [4] and Sacco et al. [17] 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 technology (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.

6 Conclusions and Future Work

RDF is a flexible data representation format that 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 their own 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, for example with other business partners, supplier or clients. In this paper we proposed a set of rules that allow for the enforcement of commonly used access control models and presented the components necessary for an RDF access control enforcement framework, while detailing our own implementation of this framework. We also presented some preliminary performance evaluation of the prototype by using enterprise data supplied by our project partner. Our approach is to transform SPARQL queries and user credentials into AnQL queries. As a side effect from the user perspective queries with two or three patterns appear to run faster. However as we have seen from the benchmark results, it is still not scalable enough for deployment in an enterprise environment.

As for future work, a combination of both the access control and the provenance domains is planned along with a more scalable implementation of the proposed framework. In addition, we propose a study of whether more expressive rules are required and their application over access control annotated data.

Acknowledgements. This work is supported in part by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), the Irish Research Council Enterprise Partnership Scheme and Storm Technology Ltd. We would like to thank Aidan Hogan for his valuable comments.

References

1. M. Amini and R. Jalili. Multi-level authorisation model and framework for distributed semantic-aware environments. *IET Information Security*, 4(4):301, 2010.
2. P.A. Bonatti, J.L. De Coi, Daniel Olmedilla, and Luigi Sauro. Rule-based policy representations and reasoning. In *Semantic techniques for the web*, pages 201–232, 2009.
3. J.M. Bradshaw, Stewart Dutfield, Pete Benoit, and J.D. Woolley. KAoS: Toward an industrial-strength open agent architecture. In *Software Agents*, 1997.
4. Luca Costabello, Serena Villata, and Nicolas Delaforge. Linked data access goes mobile: Context-aware authorization for graph stores. In *LDOW - 5th WWW Workshop on Linked Data on the Web*, 2012. URL <http://hal.archives-ouvertes.fr/hal-00691256/>.
5. Richard Cyganiak, Andreas Harth, and Aidan Hogan. N-Quads: Extending N-Triples with Context, 2009.
6. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. Candidate Recommendation, W3C, February 2012.
7. Mark Evered. A case study in access control requirements for a health information system. *Second workshop on Australasian information Security*, 2004.

8. PP Griffiths. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.
9. S Javanmardi, M Amini, R Jalili, and Y. GanjiSaffar. SBAC:A SemanticBased Access Control Model. In *11th Nordic Workshop on Secure IT-systems (NordSec'06), Linkping, Sweden*, 2006.
10. L. Kagal and T. Finin. A policy language for a pervasive computing environment. In *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74. IEEE Comput. Soc, 2003.
11. Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. AnQL: SPARQLing Up Annotated RDF. In *Proceedings of the ISWC-10*, number 6496 in LNCS, pages 518–533. Springer-Verlag, 2010.
12. Nuno Lopes, Stefan Bischof, Stefan Decker, and Axel Polleres. On the Semantics of Heterogeneous Querying of Relational, XML and RDF Data with XSPARQL. In Paulo Moura and Vitor Beires Nogueira, editors, *EPIA2011 – COLA Track*, Lisbon, Portugal, October 2011.
13. Nuno Lopes, Sabrina Kirrane, Antoine Zimmermann, Axel Polleres, and Alessandra Mileo. A Logic Programming approach for Access Control over RDF. In *Technical Communications of ICLP'12*, volume 17, pages 381–392. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.
14. CJ McCollum, JR Messing, and L. Notargiacomo. 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*, pages 190–200. IEEE, 1990.
15. Li Qin and Vijayalakshmi Atluri. Concept-level access control for the Semantic Web. In *Proceedings of the 2003 ACM workshop on XML security - XMLSEC '03*, page 94. ACM Press, 2003.
16. Tatyana Ryutov, Tatiana Kichkaylo, and Robert Neches. Access Control Policies for Semantic Networks. In *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 150–157. Ieee, July 2009.
17. Owen Sacco, Alexandre Passant, and Stefan Decker. An Access Control Framework for the Web of Data. *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 456–463, November 2011.
18. Pierangela Samarati. Access control: Policies, models, and mechanisms. *Foundations of Security Analysis and Design*, 2001.
19. RS Sandhu. Role-based access control. *Advances in computers*, pages 554–563, 1998.
20. Andy Seaborne and Eric Prud'hommeaux. SPARQL Query Language for RDF. W3C Recommendation, available at <http://www.w3.org/TR/rdf-sparql-query/>, W3C, January 2008.
21. Susie Stephens. The Enterprise Semantic Web. In Jorge Cardoso, Martin Hepp, and Miltiadis D. Lytras, editors, *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond Computing for Human Experience*, pages 17–37. Springer, 2007.
22. Octavian Udrea, Diego Reforgiato Recupero, and V. S. Subrahmanian. Annotated RDF. *ACM Trans. Comput. Logic*, 11(2):1–41, 2010.
23. Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia. A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. *J. Web Sem.*, 11:72 – 95, 2012.