Protect Your RDF Data!

Title	Protect Your RDF Data!
Author(s)	Kirrane, Sabrina;Lopes, Nuno;Mileo, Alessandra;Decker, Stefan
Publication Date	2012

Protect Your RDF Data!

Sabrina Kirrane[1,2], Nuno Lopes[1], Alessandra Mileo[1], and Stefan Decker[1]

[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 an analysis of existing enterprise access control models and enforcement mechanisms. We subsequently: (i) propose a minimum set of 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. However, 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 enables IT support staff to quickly respond to customer issues or a complete picture of compounds used enables pharmaceutical companies to identify potential issues with new drugs.

The Resource Description Framework (RDF) is a flexible format for representing data on the web which can also be used for data integration [**?**]. An additional benefit is that RDF data can easily be supplemented with complementary information from Linked Open Data (LOD) sources. Relational Database to RDF (RDB2RDF) techniques enable existing relational data to be translated into RDF. However, given security is extremely important to enterprises we cannot simply extract the data and ignore the access control policies that have been placed on the data. Although we could extend RDB2RDF to extract both the

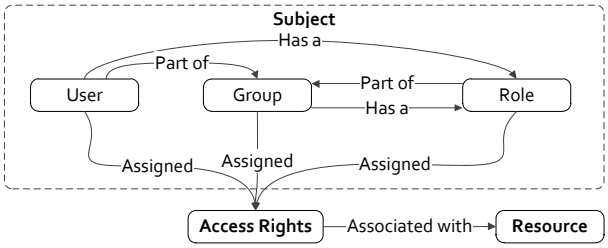


Fig. 1. Components of an access control statement

data and the access control information, RDF still does not have a mechanism to enforce existing access control policies over RDF.

A number of researchers have investigated how to supplement RDF with access control mechanisms [**? ? ? ? ?**]. 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 policies over RDF data. Therefore, we adopt a bottom up approach examining the access control requirements of existing software systems.

For this paper we analysed several classes of enterprise software systems and, based on this analysis, we make the following contributions (i) identify the core access control models that need to be supported; (ii) propose the minimum 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 relevant 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 over RDF data. The framework and our specific implementation is described and evaluated in Section 4. Finally, in Section 5 we discuss approaches to date 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 to denote the information to be protected (e.g. database records, application objects, website Uniform Resource Identifiers (URIs)). *Users* represent individuals

Table 1. Categorisation of access control models

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

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 and management). *Roles* are used to assign a set of access rights to a set of individuals and groups, for example by department (e.g. human resources, sales and marketing) or task (e.g. 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.

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 systems. The traditional access control models considered are those that are predominately used in enterprises. Given RDF is a data representation format we also examine access control models relating to both the Object Oriented (OO) and the RDB data models. As the overall goal is to propose a general access control framework for RDF, we also take into consideration models that have already been proposed for distributed systems. It is our belief that together these models could be used to efficiently enforce access control over RDF. 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) [**?**].* 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) [**?**].* 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) [**?**].* 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) [**?**].* In relational database systems VBAC is used to simultaneously grant access to one or more tables, tu-

ples or fields. A similar approach is used in Object Oriented Access Control (OBAC) [**?**] where access rights are granted to application objects.

M_{ABAC}*: Attribute Based Access Control (ABAC) [***?** *].* 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 [**?**], 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 the minimum set of 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 [**?**], that allows domain specific meta-information to be attached to RDF triples. The access rights are represented as annotations attached to individual RDF triples. Our Annotated RDF access control domain model and the annotated RDFS inference rules has been previously presented in [**?**] 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 **?**].

In the following examples an NQuads [**?**] format is used to associate access control meta-information with a triple. However the framework itself does not dictate the use of NQuads RDF serialisation, therefore alternatives approaches such as using reification or triple hash-codes to associate annotations with statement 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. Based on our analysis it was evident that data permissions are often generalised as CRUD. *Read*, *update*, and *delete* could be represented as an 3-tuple of ACL $\langle R, U, D \rangle$, 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* (ACSs) which in turn contain *Access Control Elements* (ACEs) in the form of *usernames*, *roles*, *groups* and *attributes*. Negated elements are used to explicitly deny access to an entity, for example ¬`mary` (where `mary` is a username) indicates that `mary` is *denied* access. If the annotation is missing 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. □

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 use the representation defined above to demonstrate how each of the access control models, presented in 2, can be illustrated as RDF ACLs.

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 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]]>".
```

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 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.

In each of the rules, we assume the default prefix `http://urq.deri.org/enterprise#` and 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. At this point we do not consider the effect blank nodes would have on the proposed rules. Furthermore, we syntactically separate the premises from the conclusion by a line. 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, 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 1. 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.

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

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 $?E_2$.

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

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 the organisation structure is stored as RDF, (R2) can be used to propagate permissions based on this structure. 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 (R2) 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 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 $?S_2$.

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

Such propagation chains can be broken by explicitly specifying permissions on a particular triple. However, this rule may need to be extended to consider provenance as otherwise we could propagate permissions to related data from different sources.

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 $?S_2$ is a type of $?O_1$ then the access rights of triples with $?O_1$ i.e. λ_1 should be propagated to $?S_2$.

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

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]], [[]]>"
:Read :isPartOf :Update
:Update :isPartOf :Delete
```

we can infer that:

```
:Invoice3 a :Document "<[[:john]], [[:john]], [[]]>"
```

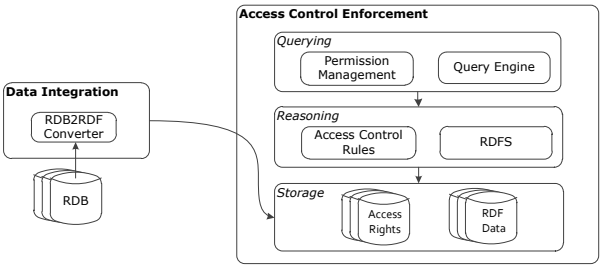


Fig. 2. RDF Data Integration and Access Control Enforcement Framework

Rule 5. Assuming that the ACL is a 3-tuple $\langle R, U, D\rangle$ 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.

$$\frac{?S\ ?P\ ?O\ \langle ?\lambda_1, ?\lambda_2, ?\lambda_3\rangle,\quad \texttt{:Read :IsPartOf :Update},\quad \texttt{:Update :IsPartOf :Delete}}{?S\ ?P\ ?O\ \langle (?\lambda_1 \oplus_{ac} ?\lambda_2 \oplus_{ac} ?\lambda_3), (?\lambda_2 \oplus_{ac} ?\lambda_3), ?\lambda_3\rangle} \quad \text{(R5)}$$

4 Framework, Implementation and Evaluation

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 is depicted in Fig. 2, 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 paper. 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 [**?**] 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 manually or from 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 Access Rights store which may be stored as quads, reified triples or associated with triple hash-codes, possibly in a separate RDF store or a restricted space of the RDF store containing the data.

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 permissions 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. 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 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 [**?**], 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 [**?**] 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 [**?**] 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 [**?**], was provided in 3.1. 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 3.3. With the exception of (R5) our prototype provides support for the presented rules, either in the form of inference rules or by explicit rules in the domain modelling. Our prototype does not support rule (R5) because the existing implementation only supports *read* permissions. However we are currently in the process of extending this prototype to support the other forms of access rights.

Our query engine relies on an extension of SPARQL, AnQL [**?**], 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 applications, and maintain a mapping between the enterprise 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.

4.3 Evaluation Results

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 extract both the data and the access rights from two separate software application databases using XSPARQL. The different datasets (DS_1, DS_2, DS_3, and DS_4) use the

Table 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. 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

same databases, tables, and XSPARQL queries and differ only on the number of records that are retrieved from the databases. 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. Furthermore, 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 domain rules (sec), and the number of triples inferred in this process. Fig. 3 (a) 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 composed of three distinct queries with: single triple patterns QTP_1; two triple patterns QTP_2; and three triple patterns QTP_3; Each query was run without annotations (none), with a credential that appears in the dataset (match) and with a credential that does not appear in the dataset (nomatch). 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. 3 (b). However queries with a combination of annotations and either two, Fig. 3 (c), or three, Fig. 3 (d), triple patterns out perform their non annotated counterparts. Such behaviour is achieved in our implementation by performing additional joins based on the annotations as opposed to querying based on the triple patterns and filtering the results based on the annotations. The results do not show a significant performance overhead between queries with no annotations, Fig. 3 (e), and those with annotations,

Table 5. Query execution time in seconds

		DS_1	DS_2	DS_3	DS_4
QTP_1	none	0.0000	0.0003	0.0013	0.0042]
	match	0.0065	0.0159	0.0300	0.0654
	nomatch	0.0081	0.0189	0.0316	0.0670
QTP_2	none	0.0247	0.0544	0.1497	0.2679
	match	0.0228	0.0638	0.0898	0.1845
	nomatch	0.0094	0.0198	0.0338	0.0690
QTP_3	none	0.0169	0.0482	0.1322	0.2213
	match	0.0241	0.0593	0.0943	0.1741
	nomatch	0.0101	0.0192	0.0316	0.0609

Fig. 3 (f). In addition there is no noticeable difference between queries with matching annotations and those that do not return a match.

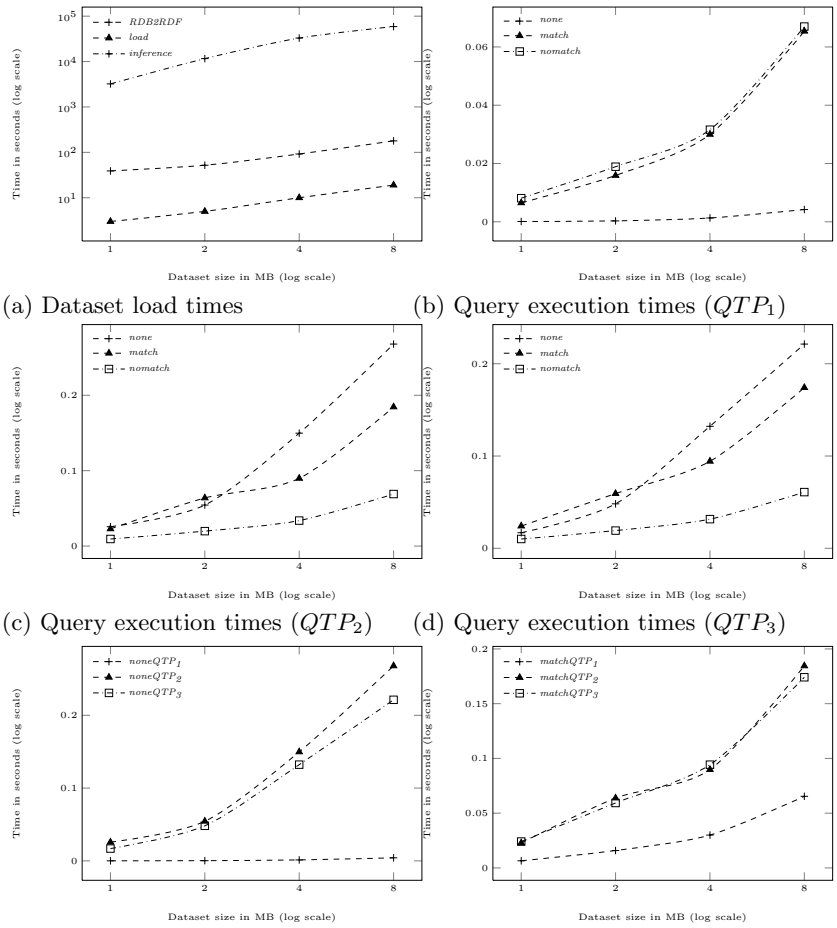
5 Related Work

In recent years, there has been an increasing interest access control policy specification and enforcement using Semantic Technology, KAos [**?**], Rein [**?**] and PROTUNE [**?**] 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, negotiation etc...), whereas in contrast the latter focuses on just one functional requirement. Two of the most well-known access control languages, KAos [**?**] and Rei [**?**], are in fact general policy languages. Although the models and the corresponding frameworks are based on semantic technology the authors do not consider applying the model or framework to RDF data.

The authors of Concept-Level Access Control (CLAC) [**?**], Semantic-Based Access Control (SBAC) [**?**] and the semantic network access control models detailed in [**? ?**] all propose access control models for RDF graphs and focus on policy propagation and enforcement based on semantic relations. **?**] allow propose propagation of access controls based on the semantic relationships among concepts. [**?**], [**?**] and [**?**] enhance the semantic relations by allowing propagation based on the semantic relationships between the subjects, objects, and permissions.

?] propose a lightweight vocabulary which defines fine-grained access control policies for RDF data. They focus specifically on modelling and enforcing access control in a mobile context. **?**] present a privacy preference ontology which allows for the application of fine-grained access control policies to an RDF file. Both **?**] and **?**] 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).



(a) Dataset load times (b) Query execution times (QTP_1)

(c) Query execution times (QTP_2) (d) Query execution times (QTP_3)

(e) Query execution times (*none*) (f) Query execution times (*match*)

Fig. 3. Load and Query times across 4 datasets

We propose an integration solution which extracts data and access rights from the enterprise software systems into RDF and an enforcement framework which allows us to reason over both the data and the policies. Approaches to date can be summarised as top-down approaches where the authors model access control based on RDF data structures and open data access control requirements. We adopt a bottom up approach examining existing access control models to determine access control requirements based on existing software engineering and database access control approaches.

6 Conclusions and Future Work

RDF is a flexible data representation format that can greatly benefit an enterprise, not only as a representation 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 enterprises to share restricted information, for example with other business partners, in a secure manner. In this paper we proposed a minimal set of rules that allow for the enforcement of commonly used enterprises 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 prototype implementation is capable of enforcing the access rights over the RDF data but, 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 more scalable implementation of the proposed framework is planned along with a study of whether more expressive rules are required and their application over the 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 for Science, Engineering and Technology Enterprise Partnership Scheme and Storm Technology Ltd. We would like to thank Aidan Hogan for his valuable comments.