# Secure Manipulation of Linked Data

Sabrina Kirrane[1,2], Ahmed Abdelrahman[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.** When it comes to publishing data on the web, the level of access control required (if any) is highly dependent on the type of content exposed. Up until now RDF data publishers have focused on exposing and linking public data. With the advent of SPARQL 1.1, the linked data infrastructure can be used, not only as a means of publishing open data but also, as a general mechanism for managing distributed graph data. However, such a decentralised architecture brings with it a number of additional challenges with respect to both data security and integrity. In this paper, we propose a general authorisation framework that can be used to deliver dynamic query results based on user credentials and to cater for the secure manipulation of linked data. Specifically we describe how graph patterns, propagation rules, conflict resolution policies and integrity constraints can together be used to specify and enforce consistent access control policies.

## 1 Introduction

In the early days, the Web was primarily used as a medium for sharing and linking static information. However it wasn't until challenges with respect to data confidentiality, authenticity and integrity were addressed that electronic business became common place. It is not surprising that the Semantic Web is following a similar evolution. With the advent of SPARQL 1.1, an update language for RDF graphs, it is possible for the Semantic Web to evolve from a medium for publishing and linking data to a dynamic read/write distributed data source, that can support the next generation of electronic business applications. However, in order to make the move from simply exposing to maintaining linked data we must first provide solutions for data security and integrity.

To date researchers have focused primarily on the specification of access control policies for RDF stores based on RDF patterns [13, 8, 4, 1, 6] or the specification and enforcement of access control ontologies over linked data [3, 14]. Although some of these authors touch upon reasoning over access control policies, they do not propose a general authorisation framework which can support reasoning based on a combination of propagation rules, conflict resolution policies and integrity constraints.

In previous work, we provided a summary of access control requirements that are needed to cater for Discretionary Access Control (DAC) over RDF data [11]. In this paper, we demonstrate how authorisations together with stratified Datalog rules can be used to enforce DAC over the RDF data model. The contributions of the paper can be summarised as follows: We (i) demonstrate how the hierarchical Flexible Authorisation Framework [9] can be adapted to work with graph data; (ii) provide a formal definition of an RDF instantiation of the framework, which we refer to as the "Graph based Flexible Authorisation Framework" or G-FAF; (iii) describe how together pattern matching and propagation rules can be used to ease the maintenance of access control policies for linked data sources; and (iv) show how conflict resolution policies and integrity constraints can ensure access control policy integrity.

The remainder of the paper is structured as follows: Section 2, examines alternative approaches for the enforcement and administration of access control over RDF data. Section 3, provides an overview of DAC requirements in the context of the RDF data model and describes the Flexible Authorisation Framework, which has been successfully applied to both the relational and the xml data models. Section 4, demonstrates how the authorisation framework can be extended to cater for the RDF graph data model. Section 5, details how graph patterns, propagation rules, integrity constraints and conflict resolution policies can be used to specify and enforce access control over the RDF data model. Whereas Section 6, discusses how the extended framework can be used to enforce access control over linked data sources and details the results of our performance evaluation. Finally Section 7, summarises the contributions and outlines directions for future work.

## 2 Related Work

Initially Semantic Web researchers focused on the modelling and the enforcement of access control over RDF stores. A number of authors have proposed access control policies based on RDF patterns that can be mapped to one or more RDF triples [13, 8, 4, 1]. Reddivari et al. [13] define a set of actions required to manage an RDF store and demonstrate how query based access control can be used to permit or prohibit access based on these actions. The authors propose default and conflict preferences that can simply be set to either permit or deny. Jain and Farkas [8] propose a data level security model which can be used to protect both explicit and inferred triples. They provide formal definitions for a number of RDF security objects and define an algorithm which generates security labels, based on a security policy and a conflict resolution strategy. Limited details of the implementation are supplied and no evaluation is performed. Whereas Abel et al. [1] propose the evaluation of access control policies at both the query and the data layers. Access conditions that are not dependent on RDF data are evaluated by a policy engine. Whereas access conditions that are dependent on

RDF data are injected into the query. Such an approach requires the substitution of variables to ensure uniqueness however in doing so they are able to leverage the highly optimized query evaluation features of the RDF store. The authors adopt a denial by default conflict resolution strategy.

Gabillon and Letouzey [6] highlight the possible administration burden associated with the maintenance of access control policies that are based on triple patterns. They propose the logical distribution of RDF data into SPARQL views and the subsequent specification of access control policies based on existing RDF graphs or predefined views. They describe a query based enforcement framework whereby each user defines a security policy for the RDF graphs/views that they own. The authors acknowledge the need for conflict resolution however, they do not propose a conflict resolution strategy.

More recently the focus has shifted to the specification and enforcement of access control policies over web resources. Costabello et al. [3] and Sacco et al. [14] both propose access control ontologies and enforcement frameworks that rely on SPARQL ASK queries to determine if the requester possesses the attributes necessary to access the requested resource. Costabello et al. [3] use context data supplied by the requester to limit the scope of the SPARQL query to authorised named graphs. The authors propose the disjunctive evaluation of policies thus circumventing the need for a conflict resolution mechanism. Whereas Sacco et al. [14] provide a filtered view of a data providers FOAF profile based on a matching between the data providers privacy preferences and the requesters attributes. Policies can be specified for an entire graph, one or more triples or individual subjects, predicates and objects. The authors do not propose any conflict resolution strategy.

In our early work, we demonstrated how annotated RDF can be used to limit access to triples and to derive access rights for inferred triples using annotated RDFS inference rules [12]. In this paper, we allow for the specification of authorisations based on quad patterns, thus catering multiple levels of granularity (i.e. one or more graphs, triples, classes or properties). Moreover, we provide a general mechanism for the administration and enforcement of access control policies using a combination of propagation rules, integrity constraints and conflict resolution policies.

## 3 Preliminaries

In previous work [11], we examined how DAC principles, that have been successfully applied to relational and XML data, can be applied to the RDF data model. In this paper, we introduce the hierarchical *Flexible Authorisation Framework* [9], henceforth referred to as H-FAF, and demonstrate how it can be extended to cater for DAC over the RDF graph data model, which we intuitively name G-FAF. We start by providing a summary of DAC requirements for the RDF data model, before providing the necessary background information about the H-FAF data system and authorisation architecture.

## 3.1 Discretionary Access Control for RDF Data

In DAC access to resources is constrained by a central access control policy however, users are allowed to override the central policy by passing their access rights on to others [16], known in the literature as delegation. DAC principles that have been successfully integrated into a number of operating systems, databases and information systems developed by well known software vendors (e.g. Oracle, Microsoft, SAP, IBM). However, our decision to base our work on the DAC model was threefold: it has been adopted by several relational DBMS vendors; its inherent flexibility makes it particularly suitable for distributed data; and its potential for handling context based authorisations in the future. Based on our analysis, of DAC for both the relational and XML data models [11], an authorisation framework needs to be able to cater the following requirements:

- In order to ensure the expressivity and the maintainability of access control policies it should be feasible to specify *authorisations* at multiple levels of granularity, from both a data (i.e. nodes, arcs, triples, collection of triples and name graphs) and a schema (i.e. classes and properties) perspective.
- Like the relational and XML data models RDF access rights should be tightly coupled with the operations performed on the data model. However, as graph update operations can only be applied to triples and graph management operations are only appropriate for graphs, *integrity constraints* are needed to ensure the consistency of the access control policies.
- In both the relational and hierarchical data models authorisations can be derived based on the schema. When it comes to the RDF data model similar *derivations* are highly desirable as they simplify authorisation maintenance.
- In DAC access to resources is constrained by a central access control policy however, users are permitted to pass their own access rights on to others [16], known formally as *delegation*.
- As conflicts can occur as a result of inconsistent explicit, derived and delegated policies *conflict resolution* strategies are required to ensure a conclusion can always be reached. Samarati [15] highlights the need for a flexible conflict resolution mechanism which can support different *conflict resolution* strategies depending on the situation.

## 3.2 H-FAF Data System and Authorisation Framework

The H-FAF is an authorisation framework that can be used to restrict access to different classes of data objects (e.g. files, relations, objects, images), with different access control requirements. The authors provide a general definition for a data system and devise a modular architecture which together with declarative rules can be used to ease access control policy administration, by exploiting the hierarchical structure of the data system components.

**Data System Components.** An authorisation framework describes the items to be protected (`data items`), to whom access is granted (users, groups, roles collectively known as `authorisation subjects`) and the operations that need to be protected (`access rights`). Together these components are known as a `data system` formally defined by Jajodia and Samarati [9] as follows:

**Definition 1 (Data System).** A *Data System (DS)* is a 5-tuple $\langle OTH, UGH, RH, A, Rel\rangle$ where: $OTH$ is an *object-type hierarchy*; $UGH$ is a *user-group hierarchy*; $RH$ is a *role-hierarchy*; $A$ is a set of *access rights* and $Rel$ is a set of n-ary *relationships* over the different elements of $DS$.

**Authorisation Framework.** In addition to the formal `data system` definition Jajodia and Samarati [9] propose a number of distinct components that together provide support for access control enforcement and administration:

- *Authorisations* are rules that dictate the `access rights` that `authorisation subjects` are allowed/prohibited to perform on `data items`.
- *Propagation Rules* enable the derivation of implicit authorisations from explicit authorisations and the hierarchical structure of `data system` components.
- *Conflict Resolution Policies* are rules that provide flexible support for different conflict resolution strategies.
- *Integrity Constraints* are rules that enforce restrictions on authorisation specification thus decreasing the potential for runtime errors.

Rules are expressed in stratified Datalog with negation and are constructed from a combination of explicit authorisations, historical authorisations and both the hierarchical structure of and the relationship between the different `data system` components.

## 4 From a Hierarchical to a Graph Data System

As both the hierarchical structure of and the relationship between the `data system` components can be recorded as RDF in this paper we adapt and extend the original `data system` and rule definitions to work with the RDF data model. As per the original framework we chose a declarative approach as it has been proven to work well and is based on familiar concepts. We start by describing the individual G-FAF data system components in the context of RDF and extend the original formal definition of a `data system` to cater for graph data structures. Although we are dealing with graph data the authorisations and the rules can also be expressed using stratified Datalog with negation. Throughout the paper, we use examples from the Berlin SPARQL Benchmark (BSBM) Dataset [3], as it is a well known dataset which is sufficiently complex to represent a real world use case. Prefixes are used as a shorthand notation for each vocabulary (e.g. rdf, rdfs, bsbm) and variables are represented using a ? prefix. The following default prefix is used to increase readability:
(`http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1`).

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

### 4.1 Individual Data System Components

In G-FAF `data items`, `access rights` and `authorisation subjects` are represented as one or more graphs that may or may not be disjoint.

**Data Items.** In the Semantic Web information is represented as *RDF triples* that are used to make statements about resources in the form of subject-predicate-object expressions. An *RDF graph* is a finite set of RDF triples. *Named graphs* are used to collectively refer to a number of RDF statements. Although there are several RDF representation formats in this paper we use *nquads*.

**Definition 2 (RDF Quad).** An *RDF Quad* is formally defined as a 4-tuple $\langle S, P, O, G \rangle \in \mathbf{UB} \times \mathbf{U} \times \mathbf{UBL} \times \mathbf{U}$ [4], where S is called the *subject*, P the *predicate*, O the *object* and G the *named graph*. **U**, **B** and **L**, are in turn used to represent *URIs*, *blank nodes* and *literals* respectively.

*Example 1 (RDF Quad).* The following quad states that there exists a triple in the Graph2013 dataset stating that Vendor1 is a vendor.
`:Vendor1 rdf:type bsbm:Vendor :Graph2013` □

**Access Rights.** Like databases and file systems access can be restricted based on the operations that a user attempts to execute on the data items [11]. In the case of RDF these operations take the form of: graph query operations (`SELECT`, `CONSTRUCT`, `ASK` and `DESCRIBE`); graph update operations (`INSERT, DELETE, DELETE/INSERT`); and a number of operations specifically for graph management (`DROP`, `COPY`, `MOVE` and `ADD`). Three additional access rights are required to facilitate access control administration, namely: `GRANT`, `REVOKE` and `FULL ACCESS`. The `GRANT` privilege allows users to grant access to others based on their own privileges. Whereas the `REVOKE` privilege allows users to revoke the access rights they have granted to others. `FULL ACCESS` is a super access right that subsumes all other access rights. We model the operations as one or more RDF graphs and use vocabularies such as RDFS to define a partial order over the operations. Although it is possible to infer implicit access rights based on the partial order, we do not provide specific details in this paper.

**Authorisation Subjects.** Subject is an umbrella term used to collectively refer to different user credentials. We propose the verification of access based on credential matching, as such we make no distinction between a user *playing* a role as opposed to *belonging* to a group. Therefore, we merge both the user-group and role hierarchies and refer to them simply as `authorisation subjects`. Such a merge does not impact the specification or enforcement of authorisations and in fact affords a greater degree of flexibility with respect to the inclusion of additional types of user credentials. As RDF is a web based distributed data model we extend the subject definition, to include user attributes. Combined

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

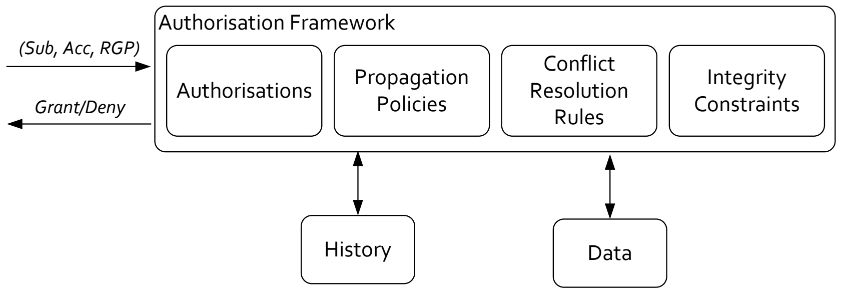


**Fig. 1.** Authorisation Framework

users, groups, roles and attributes can be represented as one or more RDF graphs possibly disjoint.

### 4.2 Extending the Original Data System Definition

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

**Definition 3 (Graph Data System).** A *Graph Data System (GDS)* is defined as a 3-tuple $\langle DIG, ASG, ARG \rangle$ where: *DIG* represents one or more *data graphs*, that may be disjoint; *ASG* denotes one or more *subject graphs*; and *ARG* stands for graphs of *access rights* used to restrict access to the data items in *DIG*.

## 5 G-FAF Authorisation Enforcement and Administration

Given an arbitary but fixed `Graph Data System`, we describe the individual G-FAF components (Fig. 1) and demonstrate how together these components can be used to deliver dynamic query results based on user credentials and to cater for the secure manipulation of RDF graph data (Section. 6). We extend the original framework to include the *Data* component, which is necessary to infer new access control policies based on a combination of RDF data and rules. Although in this paper, we do not examine the role of the *History* component, it is worth noting that historical information is important for accountability and also to cater for contextual access control policies that rely on historical data.

## 5.1 Authorisations

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

*Example 2 (RDF Quad Pattern).* A single quad pattern containing variables in both the subject ($?S$) and graph ($?G$) positions is used to match products spanning multiple graphs.
$?S$ `rdf:type bsbm:Product` $?G$. □

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

*Example 3 (RDF Graph Pattern).* A graph pattern with variables in both the subject ($?S$) and graph ($?G$) positions is used to match all products for Producer1 from multiple graphs.
$?S$ `rdf:type bsbm:Product` $?G_1$.
$?S$ `bsbm:producer bsbm-inst:Producer1` $?G_2$. □

In order to cater for certain conflict resolution strategies and for the delegation of access rights we extend the original authorisation definition to include *Type* and *By* attributes. The *Type* attribute is necessary to differentiate between explicit and inferred authorisations, whereas the *By* attribute is used to denote the person who created the authorisation. By default the *Type* attribute is set to *E* for explicit and the *By* attribute defaults to a reserved literal *OWNER*.

**Definition 4 (Authorisation).** An authorisation is represented as a 6-tuple $\langle Sub, Acc, Sign, RGP, Type, By\rangle$. *Sub* represents the *authorisation subject. Acc* is used to denote *access rights. Sign* indicates if the user is *granted* or *denied* access. *RGP* symbolises the *RDF Graph Pattern. Type* is used to indicate if the authorisation is explicit (E) or implicit (I) and *By* represents the person who created the authorisation.

*Example 4 (Authorisation).* Using the following authorisation a bsbm:admin can grant all bsbm:partners `UPDATE` access to all triples in the :Graph2008 graph.
⟨ `bsbm:Partner`, `UPDATE`, +, ⟨ $?S, ?P, ?O$, `:Graph2008`⟩, `E`, `bsbm:Admin` ⟩ □

## 5.2 Propagation Policies

Propagation policies can be used to simplify authorisation administration by allowing for the derivation of implicit authorisations from explicit ones. For example, we can derive new authorisations based on the logical organisation of `authorisation subjects`, `access rights` and `data items` [10] or the RDF Schema vocabulary [11]. We provide a formal definition for a propagation rule which can be used as a blueprint for both general and specific derivation rules

(Def. 5). In addition, we present a propagation algorithm (Alg. 1) which can be used to either evaluate the propagation policy at query time or alternatively to materialise implicit authorisations when authorisations are added or removed.

**Definition 5 (Propagation Policy).** A *Propagation Policy* is a a rule of the following format: $\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle RGP_y\rangle, ?Type_y, ?By_y\rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle RGP_x\rangle, ?Type_x, ?By_x\rangle$ , $[\langle RGP_1\rangle \wedge ... \wedge \langle RGP_x\rangle \wedge ... \wedge \langle RGP_y\rangle \wedge ... \wedge \langle RGP_n\rangle]$
The premise is composed of an *Authorisation* and an *RDF Graph Pattern*. If the *Authorisation* exists in the list of Authorisations and the *RDF Quad Pattern* exists in the Data then we can infer the conclusion.

*Example 5 (Subject Hierarchy Inheritance).* Using the following rule the access rights assigned to employees can be derived for all managers.
$\langle$ `bsbm:Mgr`, $?Acc, ?Sign, \langle ?S, ?P, ?O, ?G\rangle, I, ?By\rangle \leftarrow$
$\langle$ `bsbm:Emp`, $?Acc, ?Sign, \langle ?S, ?P, ?O, ?G\rangle, ?Type, ?By\rangle$ ,
$[\langle$ `bsbm:Mgr`, `rdf:type`, `bsbm:Emp`, $?G\rangle \wedge \langle ?S, ?P, ?O, ?G\rangle]$ □

*Example 6 (Class to Instance Propagation).* The following rules propagates the access rights assigned to a *bsbm:Product* class to all instances of the class.
$\langle ?Sub, ?Acc, ?Sign, \langle ?Z, ?Y, ?A, ?G_x\rangle, I, ?By\rangle \leftarrow$
$\langle ?Sub, ?Acc, ?Sign, \langle$ `bsbm:Product`, `rdf:type`, `rdf:Class`, $?G_y\rangle, ?Type, ?By\rangle$ ,
$[\langle ?Z$, `rdf:type`, `bsbm:Product`, $?G_z\rangle \wedge \langle ?Z, ?Y, ?A, ?G_x\rangle]$

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

### 5.3 Conflict Resolution Rules

Rather than propose a conflict resolution strategy we provide a formal definition for a conflict resolution rule (Def. 6) that can be used to determine access given several different conflict resolution strategies. For example conflict resolution policies based on the structure of the `graph data system` components; the sensitivity of the data requested; or contextual conditions pertaining to the requester. In order to cater for type matching the Authorisation *RDF Graph Pattern* is replaced with an *Extended RDF Graph Pattern* which includes the reserved words `CON` and `VAR`, that are used to match all constants and variables respectively. As multiple conflict resolution rules may be applicable, each rule should be assigned a priority and rules should be evaluated based on priority until the conflict has been resolved. The default rule which matches everything is assigned the lowest priority thus ensuring a conclusion can always be drawn.

**Definition 6 (Conflict Resolution Rule).** A *Conflict Resolution Rule* is a rule of the following format:
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ERGP_x\rangle, ?Type_x, ?By_x\rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ERGP_x\rangle, ?Type_x, ?By_x\rangle > \langle ?Sub_y, ?Acc_y, ?Sign_y, \langle ERGP_y\rangle, ?Type_y, ?By_y\rangle$
where $>$ indicates the authorisation to the left of the $>$ symbol takes precedence

```
Data: Authorisations, PropPolicy, RDFData
Result: Authorisations
forall the pol in PropPolicy do
    if Authorisations CONTAINS pol.premise.authorisation then
        if RDFData CONTAINS pol.premise.graphPattern then
            Authorisations += pol.conclusion.authorisation
        end
    end
end
return Authorisations
```
**Algorithm 1:** Applying the Propagation Rules

over the authorisation to the right; $?Sub$, $?Acc$, $?Type$ and $?By$ match authorisation *subjects*, *access rights*, *type* and *by* attributes, represented as constants or variables, and *ERGP* denotes an *Extended RDF Graph Pattern.*

*Example 7 (Most Specific takes precedence).* The following rule states that authorisations assigned to specific subject, predicate and objects in a graph override authorisations assigned to the whole graph.
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON} \rangle, ?Type_x, ?By_x \rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON} \rangle, ?Type_x, ?By_x \rangle >$
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle \texttt{VAR}, \texttt{VAR}, \texttt{VAR}, \texttt{CON} \rangle, ?Type_y, ?By_y \rangle$ □

*Example 8 (Explicit overrules Implicit).* Using the following rule it is possible to state that explicit authorisations override implicit authorisations.
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x, ?P_x, ?O_x, ?G_x \rangle, E, ?By_x \rangle \leftarrow$
$\langle ?Sub_x, ?Acc_x, ?Sign_x, \langle ?S_x, ?P_x, ?O_x, ?G_x \rangle, E, ?By_x \rangle >$
$\langle ?Sub_y, ?Acc_y, ?Sign_y, \langle ?S_y, ?P_y, ?O_y, ?G_y \rangle, I, ?By_y \rangle$ □

### 5.4 Integrity Constraints

Integrity constraints are used to restrict authorisation creation based on the existing relationships between SPARQL operations and RDF data items. For example, `INSERT` and `DELETE` can only be applied to an RDF quad whereas `DROP`, `CREATE`, `COPY`, `MOVE` and `ADD` can only be associated with a named graph. As per conflict resolution rules the integrity constraints use the *Extended RDF Graph Pattern* which includes the reserved words `CON` and `VAR` that are used to match all constants and variables respectively. We provide a formal definition of an integrity constraint (Def. 7) and demonstrate how general rules can be used to constrain the specification of the `INSERT` (Ex. 9) and the `CREATE` (Ex. 10) access rights.

**Definition 7 (Integrity Constraint).** An *Integrity Constraint* is a rule of the following format:
$\text{error} \leftarrow [\neg]\ \langle ?Sub, ?Acc, ?Sign, \langle ERGP_x \rangle, ?Type, ?By \rangle$
where square brackets [] are used to denote the optional classical negation prefix

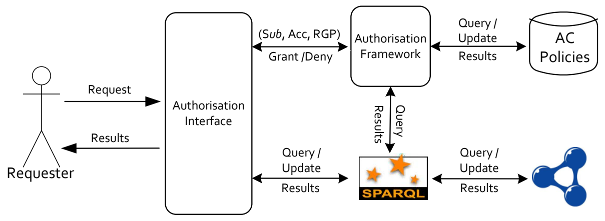


**Fig. 2.** Authorisation Architecture

($\neg$); $?Sub$, $?Acc$, $?Type$ and $?By$ match authorisation *subjects*, *access rights*, *type* and *by* attributes, represented as constants or variables and $ERGP$ denotes an *Extended RDF Graph Pattern.*

*Example 9 (INSERT Constraint).* Using an integrity constraint we can ensure that the `INSERT` access right is only applied to RDF quads.
error $\leftarrow \neg\langle ?Sub, \texttt{INSERT}, ?Sign, \langle \texttt{CON}, \texttt{CON}, \texttt{CON}, \texttt{CON}\rangle, ?Type, ?By\rangle$ □

*Example 10 (CREATE Constraint).* The following integrity constraint ensures that the `CREATE` graph management access right is only associated with named graphs.
error $\leftarrow \neg\langle ?Sub, \texttt{CREATE}, ?Sign, \langle \texttt{VAR}, \texttt{VAR}, \texttt{VAR}, \texttt{CON}\rangle, ?Type, ?By\rangle$ □

## 6 Application and Evaluation

RDF data is mostly exposed on the web via sparql endpoints. Although the architecture we propose will work with any query language in this paper we describe how it can be used in conjunction with SPARQL to enforce and administer access control over RDF. First, we discuss how the framework can be used to enforce and administer access control over linked data sources. Next, we examine the performance of our Java implementation of the framework.

### 6.1 Applying the Framework to linked data

The *Authorisation Architecture* in Fig. 2 depicts how G-FAF can be used for the enforcement and administration of access control policies over linked data sources. We do not focus on authentication in this paper, and thus we assume that the credentials supplied by the requester have been successfully authenticated via alternative means, for example WebId and self-signed certificates, working transparently over HTTPS.

**Enforcement of Authorisations.** In addition to the usual sparql query the requester must submit their credentials, which are verified by an external authentication system. The *Authorisation Interface* maps the sparql query to an *Authorisation Request* of the form $\langle Sub, Acc, RGP\rangle$ (a subset of Def. 4) and

```
**Data**: AuthRequest, AuthHashMap, ConflictPolicy
**Result**: grant/deny
key = AuthRequest.sub + AuthRequest.acc
authHashSet = getAuthHashSet(key, AuthHashMap)
quadHashMap = createQuadHashMap(authHashSet)
dominantAuth = quadHashMap.Auth
while quadHashMap CONTAINS AuthRequest.RGP do
    authMatches += quadHashMap.Auth
end
if authMatches CONTAINS true and authMatches CONTAINS false then
    dominantAuth = resolveConflict(authMatches, ConflictPolicy)
end
return dominantAuth.Sign
```

**Algorithm 2:** Authorisation Enforcement Algorithm

submits it to the *Authorisation Framework* (Fig. 2). The authorisation algorithm (Alg. 2) checks if the *Authorisation Request* can be derived using the *Authorisations* and the *Conflict Resolution Policies*. If the algorithm manages to successfully derive the authorisation, access to the requested data is granted otherwise the request is denied. If access is granted the *Authorisation Interface* passes the sparql query to the *Query Engine*, which in turn processes the query in the normal way. Finally, the query results are returned to the *Requester* via the *Authorisation Interface*. In the current implementation the `subject` must be granted access to each triple in order to be permitted to execute the query. In future work we plan to investigate the data integrity implications of granting access to subsets of the graph pattern through query rewriting. For example, if a user requests the names of all employees who earn more than 50,000, and that user is denied access to salary data, all employees would be returned leading them to believe that this is the answer to their query.

**Administration of Authorisations.** We propose an ownership model, whereby the data producer is granted `FULL ACCESS` to the data items they create. When a user issues a graph update or graph management query, access is verified using the authorisation algorithm (Alg. 2). If authorisation succeeds the sparql query is passed to the *Query Engine*. For `INSERT`, `ADD` or `COPY` operations, if the query succeeds the administration algorithm (Alg. 3) ensures it adheres to the integrity constraints prior to creating a new authorisation. For `DELETE`, `DROP` or `MOVE` operations, if the query succeeds the administration algorithm (Alg. 3) simply deletes relevant authorisations from the access control policy. In both instances the update of both the RDF graph and the authorisation table should be wrapped in a transaction to ensure that either both or neither succeed.

**Delegation of Access Rights.** In order to cater for delegation of access control, a number of administration modules are required. For example the ability to list your own access rights, grant/revoke access rights to others and view the access rights you have delegated. Based on the ownership model data producers are granted `FULL ACCESS` to the data items they create and have the ability to

```
Data: AuthRequest, AuthHashMap, IntegrityPolicy, PropRules
Result: true/false
newAuth = AuthRequest + sign.Grant + type.E + by.Owner
if AuthRequest.Acc==INSERT or AuthRequest.Acc==ADD or
AuthRequest.Acc==COPY then
    if checkIntegrity(AuthRequest, IntegrityPolicy) = true then
        AuthHashMap += newAuth
        AuthHashMap = applyPropRules(AuthHashMap, PropRules)
        return true
    end
end
else if AuthRequest.Acc==DELETE or AuthRequest.Acc==DROP or
AuthRequest.Acc==MOVE then
    AuthHashMap -= newAuth
    return true
end
return false
```

**Algorithm 3:** Authorisation Administration Algorithm

**Table 1.** Dataset and Authorisations description

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

`GRANT` and `REVOKE` access to/from others. As neither the grant nor the revoke algorithms are dependent on the data model traditional revocation approaches such as cascading [5, 7] and non-cascading [2] can be used in conjunction with the proposed framework.

### 6.2 Performance Evaluation

For the evaluation of G-FAF we created three separate experiments to: (i) examine the overhead associated with access control over different data sets; (ii) deduce the impact given an increasing number of authorisations; and (iii) determine the performance increase for a number of propagation rules (the most expensive administration operation). The benchmark system has an Intel(R) Xeon(R) CPU 8 core 2.13GHz processor, 64 GB of memory and runs Debian 6.0.3. The authorisation framework was written in Java and the evaluation was performed over an in memory store using Jena ARQ. Both the datasets (Table. 1) and the queries were generated from the Berlin SPARQL Benchmark (BSBM) dataset. Two separate query sets were created: (i) $QS_S$ which contained *10* SELECT queries; and

**Table 2.** Queries over increasing datasets

| | | $DS_1$ | $DS_2$ | $DS_3$ | $DS_4$ | $DS_5$ |
|---|---|---|---|---|---|---|
| | $\emptyset$ | 345 | 657 | 1432 | 2604 | 5005 |
| $QS_S$ *query time (ms)* | $\exists$ | 429 | 700 | 1164 | 2549 | 5149 |
| | $\emptyset$ | 8 | 8 | 9 | 8 | 9 |
| $QS_U$ *query time (ms)* | $\exists$ | 9 | 9 | 9 | 9 | 9 |

**Table 3.** Queries over increasing authorisations

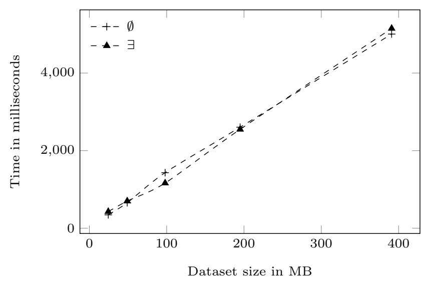
| | $AS_1$ | $AS_2$ | $AS_3$ | $AS_4$ | $AS_5$ |
|---|---|---|---|---|---|
| $QS_S$ *query time (ms)* | 5056 | 4801 | 4861 | 4892 | 4869 |
| $QS_U$ *query time (ms)* | 9 | 8 | 9 | 8 | 9 |

(ii) $QS_U$ which contained *5* INSERT and *5* DELETE queries. In both instances the queries composed of a combination of one, two and three triple patterns. Access was granted or restricted to all quads (?S ?P ?O ?G); a particular graph (?S ?P ?O G1); all quads of type offer (?S rdf:type bsbm:Offer ?G); all classes (?S rdf:type rdf:Class); and all properties (?S rdf:type rdf:Property). Users were either assigned ( select; select & insert; select, insert & delete ) or denied ( delete; insert & delete; select, insert & delete; ) access to single quad patterns. The integrity constraints presented in Examples 9 and 10 were added, to ensure that `INSERT` and `DELETE` operations were only applied to RDF quads. The conflict resolution rules presented in Examples 7 and 8 along with an additional denial takes precedence rule, were executed in the event of a conflict. The datasets, queries and the conflict resolution, integrity and propagation rules used in the experiments can be found at `http://gfaf.sabrinakirrane.com/`. All calculations presented were based on an average of 20 response times excluding the two slowest and fastest times.

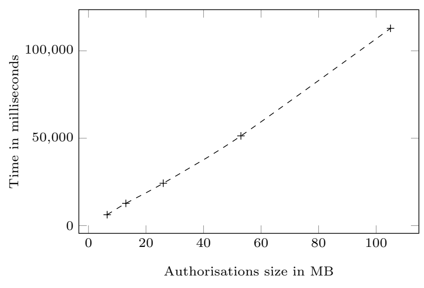
In order to evaluate the enforcement algorithm we ran both the select ($QS_S$) and the update ($QS_U$) query sets, without access control ($\emptyset$), with access control for users who were granted access ($\exists$), over an authorisation set containing *588,000* grant and *402,001* deny authorisations. As expected the results indicate that select query execution times are not impacted when the dataset is increased (Table 2). However, little or no increase in performance times over increasing authorisations (Table 3 and Fig. 3a) was unexpected. Such behavior can be attributed to the fact that all authorisations are indexed by a combined `subject access right` key and subsequently by `graph pattern` (*seeAlg.* 2). For the evaluation of the propagation rules we examined the impact associated with three schema based derivations from: classes to all instances of that class; properties to all instances of that property; and an instance to property values associated with that instance. Again we ran the experiment over increasing datasets and authorisations (Table 4). Based on the results we can see that reasoning behaves linearly when the number of authorisations are increased (Fig. 3b), whereas there is little or no impact when the dataset was increased.

**Table 4.** Propagation rules performance

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



(a) Select query performance



(b) Rules over increasing authorisations

**Fig. 3.** Query and Propagation times

## 7 Conclusions and Future Work

With the introduction of RDF update languages, such as SPARQL 1.1, it is now feasible to both query and manage distributed and linked RDF data. However like web applications and web services, SPARQL endpoints need to protect the security of the data source and the privacy and the integrity of the data therein. In this paper, we discussed how the hierarchical Flexible Authorisation Framework, proposed by Jajodia and Samarati [9], can be adapted to cater for secure manipulation of RDF graph data. We provided a formal definition of authorisations, propagation rules, conflict resolution policies and integrity constraints, within the context of RDF, and describe how together these components can simultaneously provide access control over interlinked RDF graphs. The results of our initial performance evaluation are very promising, as in general they show only a negligible increase in query processing time and a linear increase in derivation times over increasing authorisations.

To date we have focused on the application of access control to simple graph pattern queries. In future work we will look into handling more expressive queries, for example those that include filters, subqueries, aggregates etc, and investigate integrity issues with respect to query rewriting. We also plan to extend the integrity constraints to ensure the integrity of both the rules and conflict resolution policies.

**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 and Nuno Lopes for their valuable comments on the paper.

## References

1. Fabian Abel, JL De Coi, Nicola Henze, and AW Koesling. Enabling advanced and context-dependent access control in RDF stores. *The Semantic Web*, 2007.
2. Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. Authorizations in relational database management systems. *Proceedings of the 1st ACM conference on Computer and communications security - CCS '93*, pages 130–139, 1993.
3. 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.
4. S Dietzold and S Auer. Access control on RDF triple stores from a semantic wiki perspective. *ESWC Workshop on Scripting for the Semantic Web*, 2006.
5. Ronald Fagin. On an authorization mechanism. *ACM Transactions on Database Systems (TODS)*, 3(3):310–319, 1978.
6. Alban Gabillon and Leo Letouzey. A View Based Access Control Model for SPARQL. *2010 Fourth International Conference on Network and System Security*, pages 105–112, September .
7. PP Griffiths and BW Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1:242–255, 1976.
8. Amit Jain and Csilla Farkas. Secure resource description framework: an access control model. *ACM SACMAT*, 2006.
9. Sushil Jajodia and P Samarati. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 1(212), 2001.
10. Sabrina Kirrane, Nuno Lopes, Alessandra Mileo, and Stefan Decker. Protect Your RDF Data! In *In Proceedings of the 2nd Joint International Semantic Technology Conference*, 2012.
11. Sabrina Kirrane, Alessandra Mileo, and Stefan Decker. Applying DAC principles to the RDF graph data model. In *28th IFIP TC-11 International Information Security and Privacy Conference*, 2013.
12. 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.
13. Pavan Reddivari, Tim Finin, and Anupam Joshi. Policy-Based Access Control for an RDF Store. In *Proceedings of the IJCAI-07 Workshop on Semantic Web for Collaborative Knowledge Acquisition*, January 2007.
14. Owen Sacco, Alexandre Passant, and Stefan Decker. An Access Control Framework for the Web of Data. *10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011.
15. Pierangela Samarati. Access control: Policies, models, and mechanisms. *Foundations of Security Analysis and Design*, 2001.
16. RS Sandhu and P Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 1994.