Applying DAC Principles to the RDF Graph Data Model

Sabrina Kirrane[1,2], 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. In this paper we examine how Discretionary Access Control principles, that have been successfully applied to relational and XML data, can be applied to the Resource Description Framework (RDF) graph data model. The objective being to provide a baseline for the specification of a general authorisation framework for the RDF data model. Towards this end we provide a summary of access control requirements for graph data structures, based on the different characteristics of graph models compared to relational and tree data models. We subsequently focus on the RDF data model and identify a list of access rights based on SPARQL query operations; propose a layered approach to authorisation derivation based on the graph structure and RDFSchema; and demonstrate how SQL GRANT and REVOKE commands can be adapted to cater for delegation of privileges in SPARQL.

1 Introduction

A *Data Model* is an abstraction used to represent real world entities, the relationship between these entities and the operations that can be performed on the data. Database models can be broadly categorised as relational, tree and graph based. An important requirement for any of Database Management Systems (DBMSs) is the ability to protect data from unauthorised access. An *Access Control Model* is a blueprint for defining authorisations which restrict access to data. Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC) are the predominant access control models both found in the literature and used in practice. In this paper we focus specifically on DAC and examine how it can be used to restrict access to RDF data. We base our work on the DAC model as: it has been successfully adopted by several relational DBMS vendors; because of its inherent flexibility; and its potential for handling context based authorisations in the future.

Several researchers have investigated how to add access control to RDF data. Existing approaches can be categorised as ontology based [9, 23], rules based and [15, 11, 18] and inference based [15, 20, 16, 21, 1]. In previous work, which would

L.J. Janczewski, H.B. Wolfe, and S. Shenoi (Eds.): SEC 2013, IFIP AICT 405, pp. 69–82, 2013.

also be categorised as inference based, we demonstrated how Annotated RDF can be used to propagate permissions assigned to triples based on RDFSchema [19]. We proposed a number of rules that can be used for the derivation of access rights based on subject, access rights and resource hierarchies [22]. In this paper, we examine the specification, derivation and delegation of access control over RDF graph data guided by DAC principles and experiences applying these principles to the relational and tree based data models. Based on our analysis, we make the following contributions:(i) discuss how DAC principles can be used to restrict access to RDF data; (ii) describe how the graph structure can be used to derive implicit access rights; (iii) propose a set of rules that are necessary for derivation of authorisations based on RDFSchema; and (iv) demonstrate how SQL grant and revoke commands can be adapted to manage RDF authorisations. Together these contributions provide a solid building block for DAC policy enforcement for the RDF data model.

The remainder of the paper is structured as follows: In Section 4 we discuss related work. Section 2 describes how DAC is used to restrict access to relational and tree based data models. Issues applying DAC to graph data are discussed and possible handling mechanisms are proposed in Section 3. Finally, we conclude and outline directions for future work in Section 5.

2 Preliminaries

DAC policies limit access to data resources based on access rules stating the actions that can be performed by a subject. The term *subject* is an umbrella term used to collectively refer to users, roles, groups and attribute-value pairs. In DAC access to resources are constrained by a central access control policy, however users are allowed to override the central policy and can pass their access rights on to other users [25], known formally as delegation. Over the years the DAC model has been extended to consider: constraint based authorisations (e.g. time, location); access to groups of users, resources and permissions; support for both positive and negative authorisations; and conflict resolution mechanisms [24]. In this section we describe how DAC is used to protect relational and tree based data models.

2.1 Applying DAC to the Relational Model

In the relational model (Fig.1), data items are grouped into *n-ary relations*. A relation header is composed of a set of named data types known as *attributes*. The relation body is in turn made up of zero of more *tuples* i.e. sets of attribute values. A *primary key*, composed of one or more attributes which uniquely identifies each tuple, is defined for each relation. Relations are connected when one or more attributes (i.e. a *foreign key*) in a relation are linked to a primary key in another relation. Relations can be categorised as base relations or views. Base relations are actually stored in the database whereas views are virtual relations derived from other relations. Views are commonly used to: (i) provide access to

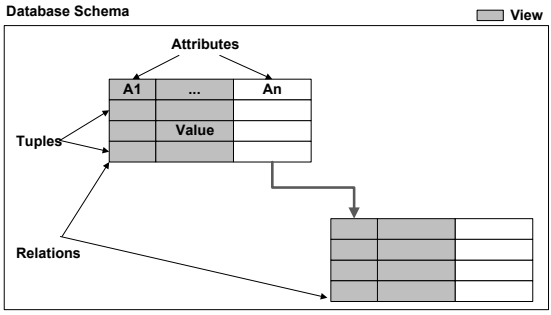

Fig. 1. Relational Data Model

information from multiple relations; (ii) restrict access to particular attributes or tuples; and (iii) derive data (e.g. sum, average, min and max).

In relational databases access is restricted both at a schema level (database, relations and attributes) and a data level (tuples and values). The access rights themselves are tightly coupled with database operations such as `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `DROP`. In addition the `GRANT` privilege allows users to grant access to others based on their own privileges. Griffiths and Wade [14] describe how DAC is implemented in System R [3] an experimental DBMS developed to carry out research on the relational data model. Two of the underpinning principles of DAC are derivation of implicit authorisations from explicit authorisations and the delegation of access rights.

Authorisations explicitly defined at schema level are implicitly inherited by other database entities, for example (i) database authorisations are inherited by all database resources; (ii) relation authorisations are inherited by all tuples; and (iii) attribute authorisations are inherited by all attribute values. Aside from Schema level derivations Griffiths and Wade [14] describe how views can be used to implicitly grant access to one or more tables, attributes or tuples spanning multiple relations. Under DAC database users are granted sole ownership of the tables and views that they create. They can subsequently grant access rights to other database users. Griffiths and Wade [14] and Bertino et al. [8] discuss how the revocations process is complicated due to recursive delegation of permissions and propose algorithms which are used to revoke access rights.

2.2 Applying DAC to the Tree Model

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

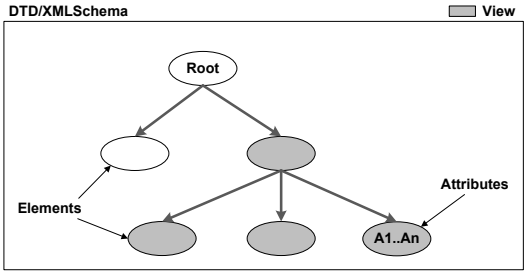

Fig. 2. XML Tree Model

In an XML data model (Fig.2) relations are represented as *elements* that can contain textual information and zero or more *attribute-value* pairs. Simple elements contain data values whereas complex data types are constructed from other elements and/or attributes giving XML it's hierarchical structure. A Document Type Definition (DTD) or an XMLSchema describe the structure of an XML document. In contrast to the relational model, XML data is not necessarily an instance of some schema.

Bertino et al. [6] describe how DAC is implemented in Author-X a prototype developed to demonstrate how access control policies can be applied to XML documents, that may or may not conform to a DTD/XMLSchema, exposed on the web. Similar to the relational model, tree based access control can also be specified at both schema and data levels. From a schema perspective access can be restricted based on the structure of the document/data item, a DTD or an XMLSchema. Whereas data level restrictions can be applied to specific elements and attributes. Similar to the relational model the access rights reflect the operations commonly performed on an XML document for example `READ`, `APPEND`, `WRITE`, `DELETE` and `INSERT`.

Propagation of authorisations based on the is-part-of relationship between documents, elements, sub-elements and attributes is one of the key features of DAC for XML [5]. Although implicit authorisations simplify access control administration, a knock on effect is that exceptions need to be catered for. In XML inheritance chains can be broken by explicitly specifying authorisations for leaf nodes. In addition, a combination of positive and negative authorisation can be used to grant access in the general case and deny access for specific instances. The introduction of negative authorisation brings with it the need for conflict resolution mechanisms (e.g. denial takes precedence). Gabillon [13] describes how delegation of privileges can be adapted to work for XML databases. The author defines a security policy language for XML which incorporates SQL GRANT and REVOKE commands.

3 Applying DAC to RDF

In this section we describe how the graph data model differs from the tree data model. We discuss how DAC can be applied to graph-based data and detail

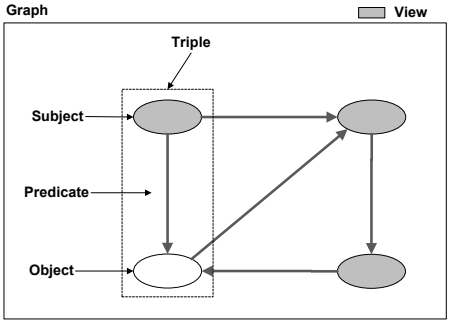

Fig. 3. RDF Graph Model

the implication such structural differences have on access control in general. Although in this paper we focus on RDF specifically a number of observations can be applied to other graph data models.

3.1 The RDF Data Model

The graph data model extends the tree model by allowing each *node* to have multiple parent relations, resulting in a generalized *graph* structure. Undirected and directed binary relationships between nodes are represented as *edges* and *arcs* respectively. In a directed graph the node from which an arc originates is called the *head* and the destination the *tail*. Graphs differ from trees in several aspects, for example: a graph doesn't have a top or bottom; a node can have more than one parent; a node can be its own ancestor; and multiple paths between nodes are permissible. The graph data model is often used where information about the graph topology is just as important as the data itself. An overview of several graph based models is provided by Angles and Gutierrez [2].

The RDF graph model is designed to represent knowledge in a distributed manner. RDF captures the semantics of data and presents it in a machine readable format. *Uniform Resource Identifiers* (URIs) in turn are used to uniquely identify data items. The fundamental building block of the RDF data model (Fig.3) is an RDF *triple* which constitutes a statement about the relationship between two nodes. An RDF triple is represented as a tuple $\langle S, P, O \rangle \in \mathbf{UB} \times \mathbf{U} \times \mathbf{UBL}$ [1], where S is called the *subject*, P the *predicate*, and O the *object* and U, B and L, are used to represent ***URIs***, ***blank nodes*** and ***literals*** respectively. The following triples represented using N3 [2] use the FOAF [3] vocabulary, a subset of which is presented in Fig. 4, to state that the `JoeBloggs` is a person who's first name is Joe and lastname is Bloggs:

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

[1] For conciseness, we represent the union of sets simply by concatenating their names.
[2] `http://www.w3.org/TeamSubmission/n3/`
[3] FOAF Vocabulary Specification, `http://xmlns.com/foaf/spec/`

```
1  foaf:Person rdf:type rdfs:Class.
2  foaf:givenName rdf:type rdf:Property.
3  foaf:givenName rdfs:domain foaf:Person.
4  foaf:lastName rdf:type rdf:Property.
5  foaf:lastName rdfs:domain foaf:Person.
```

Fig. 4. Subset of FOAF Vocabulary

```
1   entx:G1 {
2   entx:salary rdf:type rdf:Property.
3   entx:salary rdfs:domain foaf:Person.
4   entx:JoeBloggs rdf:type foaf:Person.
5   entx:JoeBloggs foaf:givenName "Joe".
6   entx:JoeBloggs foaf:lastName "Bloggs".
7   entx:JoeBloggs entx:salary "40000".
8   entx:MayRyan rdf:type  foaf:Person.
9   entx:MayRyan foaf:givenName "May".
10  entx:MayRyan foaf:lastName "Ryan".
11  entx:MayRyan entx:salary "80000".
12  }
```

Fig. 5. Snapshot of Enterprise Employee Data

RDFSchema is a set of classes and properties used to describe RDF data. Unlike XMLSchema, RDFSchema does not describe the structure of an RDF graph. Instead RDFSchema provides a framework used by *vocabularies* (known formally as *ontologies*) to describe classes, properties and relations.

In RDF there is a tight coupling between the schema and instance data. Unlike the relational and XML data models classes, properties and instances cannot be identified based on the structure alone. Like XML, *Namespaces* are used to uniquely identify a collection of RDF resources. Prefixes are used as a shorthand notation for ontology Namespaces. In this paper, we use the following ontologies RDF, RDFSchema(RDFS), FOAF and a sample enterprise ontology:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix entx: <http://urq.deri.org/enterprisex#> .
```

An RDF *graph* is a finite set of RDF triples. *Named graphs* are used to collectively refer to a number of RDF statements. In this paper we use TriG [4], an extension of N3, which uses curly brackets to group triples into multiple graphs identifiable by a URI (Fig.5). In practice the triple and the named graph are stored as *quads*.

[4] http://www.w3.org/2010/01/Turtle/Trig/

3.2 Graph Based and Schema Based Authorisations

The first step in the identification of access control requirements for RDF data is to identify the resources that need to be protected and the access rights required. The graph data model alone is quite limiting when it comes to the management of access rights. Therefore in Section 3.3, we examine how RDFSchema can be used to define more expressive authorisations.

RDF Resources. From a data perspective access can be restricted to a node (subject or object), an arc (property), two connected nodes (triple), a collection of nodes and edges (multiple triples that share a common subject) or arbitrary views of the data (named graphs). Whereas from a schema perspective authorisations can be applied to classes and properties. Given the tight coupling between schema and data items, authorisations based on classes (e.g. `foaf:person`) and properties (e.g. `foaf:givenName`) would need to be derived using schematic vocabulary such as RDFSchema or Web Ontology Language (OWL) [5]. In Section 3.3, we examine how permissions can be derived based on both the graph structure and RDFSchema.

Access Rights. The access rights of both the relational and XML data models are very similar and differ primarily by vocabulary. SPARQL proposes several operations similar to ones that exist for relational and XML data (`SELECT`, `INSERT`, `DELETE/INSERT`, `DELETE` and `DROP`). However SPARQL also defines a number of additional query operations (`CONSTRUCT`, `ASK` and `DESCRIBE`) and a number of operations specifically for graph management (`CREATE`, `COPY`, `MOVE` and `ADD`). Notable omissions from the list of SPARQL operations are the `GRANT` and `REVOKE` commands which allows users to *grant access to* or *revoke access from* others access based on their own privileges. In Section 3.4 we discuss how the grant and revoke operations could be accommodated in SPARQL.

Access Control Policy. An *Access Control Policy* details the actual authorisations and access restrictions to be enforced. Each authorisation is represented as a quad $\langle Sub, Acc, Sign, Res\rangle$ where *Sub* denotes the *subject* (not to be confused with an RDF triple subject), *Acc* the *access rights*, *Sign* indicates if the user is *granted* or *denied* access and *Res* represents the *resource* to be protected (i.e. rdf quad with optional variables, represented using a ? prefix, in any position). A matrix outlining the access rights that are appropriate for each operation (represented by an `X`) is provided in Table 1. For conciseness two connected nodes is abbreviated to *Con*, a collection of nodes and edges to *Col* and RDF properties to *Prop*. A sample access control policy is in turn presented in Table 2. Each authorisation is labelled (A_n) to ease referenceability. (A_1), (A_2) and (A_3) grant access rights SELECT, INSERT and DELETE to the graph *entx:G1*, (A_4) grants access to a particular class and (A_5) denies access to the salary property.

[5] `http://www.w3.org/TR/owl2-overview/`

Table 1. Relationship between Access Rights and Resources

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

Table 2. Sample Access Control Policy

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

If no explicit or implicit policy exists it is possible to adopt either a closed policy (deny access by default) or an open policy (grant access by default).

3.3 Derivation of Authorisations

In both the relational model and the tree model authorisations can be derived based on the data schema. When it comes to the RDF data model similar derivations are highly desirable as they simplify authorisation administration. Existing RDF database vendors adopt a view based approach to derivation, organising triples into named graphs based on the access control requirements and granting access to the entire graph. Although similar to views in relational databases in this instance the graph is materialised. An alternative approach would be to derive permissions based on the graph structure. However as it isn't possible to distinguish between schema and instance data such an approach alone is quiet limited. Therefore we propose an additional layer of derivations based on a vocabulary such as RDFSchema to define rules that leverage the semantic relations between nodes and edges. In the following rules both the premises (above the line) and the conclusion (below the line) are represented as a 5 tuple

$\langle S, P, O, \gamma, \lambda \rangle$. Where: (i) S represents a *subject*, P a *predicate* and O an *object* (together they represent a triple); (ii) γ is used to denote a named graph (which may or may not be the same for each triple); and (iii) λ is used to represent permissions i.e. *authorisation subject*, *access rights* and *sign* attributes $\langle Sub, Acc, Sign \rangle$. By including the named graph in the derivation rules it is possible to constrain the derivation to a particular graph or alternatively to span multiple graphs. Such graphs in turn can be distributed across multiple data sources.

Derivation Based on the Graph Structure. Similar to the tree model we could assign permissions to a node and recursively derive authorisations for all nodes connected to it by arcs. Another approach would be to derive authorisations for all nodes along a particular path. Existing graph search algorithms, such as those proposed by Tarjan [26], could be used to recursively traverse the graph and assign permissions to the nodes. A thorough investigation into the application of graph traversal and access control is proposed in future work.

Derivation Based on RDFSchema. One limitation of the RDF data model is that it isn't possible to distinguish between schema and instances from the graph structure alone. For example to restrict access to attributes we would need a means to derive permissions for all instances of a particular property type. Likewise to restrict access to a relation we would need to derive permissions for all properties that are instances of a particular class. To accommodate schema based derivation a combined data approach to derivation is warranted. The following rules can be used to derive access rights based on the RDFSchema vocabulary.

Rule 1. Using this rule we can derive λ, which has been assigned to a class, for all instances of that class.

$$\frac{?X \texttt{ rdf:type rdf:Class } \gamma\ \lambda,\ ?Z \texttt{ rdf:type } ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma}{?Z\ ?Y\ ?A\ \gamma\ \lambda} \quad \text{(R1)}$$

Rule 2. In this rule, λ which has been assigned to a property of a class, is derived for all instances of that property.

$$\frac{\begin{array}{r}?X \texttt{ rdf:type rdf:Class } \gamma,\ ?Y \texttt{ rdf:type rdf:Property } \gamma\ \lambda, \\ ?Y \texttt{ rdfs:domain } ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma\end{array}}{?Z\ ?Y\ ?A\ \gamma\ \lambda} \quad \text{(R2)}$$

Rule 3. The following rule propagates λ, assigned to an instance of a class, to property values associated with that instance.

$$\frac{\begin{array}{r}?X \texttt{ rdf:type rdf:Class } \gamma,\ ?Z \texttt{ rdf:type } ?X\ \gamma\ \lambda, \\ ?Y \texttt{ rdfs:domain } ?X\ \gamma,\ ?Z\ ?Y\ ?A\ \gamma\end{array}}{?Z\ ?Y\ ?A\ \gamma\ \lambda} \quad \text{(R3)}$$

Table 3. Snapshot of Derived Authorisations

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

Given a snapshot of the FOAF ontology (Fig. 4), a subset of an enterprise RDF dataset (Fig. 5) and a sample access control policy (Table. 2), we can derive additional authorisations such as those summarised in Table 3. (DA_1) to (DA_8) were derived by applying (R1) to (A_4). Whereas (DA_9) and (DA_{10}) were inferred from (R2) and (A_5).

Two additional rules which use the `rdfs:subclass` (R4) and `rdfs:subproperty` (R5) properties are proposed to demonstrate flexibility that can be gained from RDFSchema. More expressive rules based on richer vocabularies such as OWL could also be used. The database should be flexible enough to allow derivations to be turned on and off on a case by case basis.

Rule 4. In this rule we use the RSFSchema subclass inheritance mechanism to derive the permissions λ assigned to a class for all subclasses.

$$\frac{?X \texttt{ rdf:type rdf:Class } \gamma\ \lambda,\ ?Y \texttt{ rdf:type rdf:Class } \gamma,\ ?Y \texttt{ rdfs:subClassOf } ?X\ \gamma}{?Y \texttt{ rdf:type rdf:Class } \gamma\ \lambda} \quad \text{(R4)}$$

Rule 5. Similar to the above rule however in this instance we use the subproperty inheritance to derive the permissions λ assigned to a property for all subproperties.

$$\frac{?X \texttt{ rdf:type rdf:Property } \gamma\ \lambda,\ ?Y \texttt{ rdf:type rdf:Property } \gamma,\ ?Y \texttt{ rdfs:subPropertyOf } ?X\ \gamma}{?Y \texttt{ rdf:type rdf:Property } \gamma\ \lambda} \quad \text{(R5)}$$

3.4 Delegation of Access Rights

In both relational and XML databases GRANT and REVOKE commands are used to manage delegation of access rights. The SPARQL 1.1 update language does not currently support the GRANT and REVOKE commands. It thus needs

to be extended to cater for authorisation administration and delegation of access rights. We propose an adapted version of the SQL GRANT (Def.1) and REVOKE (Def.2) commands that caters for named graphs. We adopt the USING NAMED clause from other SPARQL 1.1 operations.

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

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

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

`Privilege_name` denotes the privileges identified in Section 3.1 (SELECT, CONSTRUCT, ASK, DESCRIBE, INSERT, DELETE/INSERT, DELETE, DROP, COPY, MOVE, ADD). `Resource_name` represents one or more instances of the following RDF resources (NAMED GRAPH, CLASS, PROPERTY, TRIPLE). `User_name, role_name, attribute_value` are used to identify users, roles and attributes respectively and a reserved word PUBLIC is used to assign access to all users. Finally the `WITH GRANT OPTION` is used to provide users with the ability to delegate the access right(s) to others.

Definition 1 (GRANT command).

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

Definition 2 (REVOKE command).

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

As revocation is not dependent on the data model existing approaches, such as cascading [12, 14] and non-cascading [7], devised for relational databases would also work for rdf databases (datastores).

3.5 Conflict Resolution

Conflicts can occur as a result of inconsistent: explicit; derived; and delegated policies. Samarati [24] discusses the need for different conflict resolution depending on the situation. Earlier in this section, we proposed a number of derivation rules to ease RDF access control administration and stated that implicit authorisations should be overridden by explicit authorisations. It is important that the conflict resolution strategy proposed is in keeping with both the derivation rules and overriding mandate. In this paper, we propose three complementary approaches to conflict resolution that fit well with DAC: (i) *explicit policies*

override implicit policies (ensures that positive explicit authorisations will always prevail over negative implicit authorisations); (ii) *most specific along a path takes precedence* (allows users to grant access in the general case and deny access for specific instances); and (iii) *denial takes precedence* (caters for scenarios where we have a conflict between two explicit or two implicit authorisations). In Section 3.3, we seen how derivation rules can result in conflicting authorisations, for example Table 3 (DA_4) and (DA_9) or (DA_8) and (DA_{10}). As both policies are implicit the *explicit policies override implicit policies* strategy is not applicable. In this instance the negative authorisation would prevail based on the *most specific along a path takes precedence*, as a policy assigned to a property is more specific than one applied to a class.

4 Related Work

Both Costabello et al. [9] and Sacco et al. [23] propose access control vocabularies and frameworks that can be used to enforce access control policies over RDF Data. In both instances the authors provide a filtered view of data using SPARQL ask queries. However the authors do not perform any reasoning over the access control policies they propose.

Other researchers adopt a rule based approach to access control. Dietzold and Auer [11] define access control requirements from a Semantic Wiki perspective. The authors propose a filtered data model using a combination of SPARQL queries and SWRL rules. Li et al. [18] also adopt a rule based approach providing users with a more intuitive way to specify access control policies. Both Dietzold and Auer [11] and Li et al. [18] use rules to give a more explicit meaning to the access control policies as opposed to authorisation derivation in our case.

Several reseachers Qin and Atluri [20], Javanmardi et al. [16], Ryutov et al. [21], Amini and Jalili [1] propose access control models for RDF graphs and focus on policy propagation and enforcement based on semantic relations. None of the authors examine access control from either a data model or a database perspective. Similar to us, Jain and Farkas [15] derive authorisations and propose conflict resolution mechanisms. They adopt a multilevel label-based approach to access control where policies are specified in terms of RDF patterns associated with an instance, a schema and a security classification. The derivations they propose are however limited to RDFSchema entailment rules.

Only Jain and Farkas [15] and Javanmardi et al. [16] actually mention DAC and even then they just describe DAC and do not examine how their approach compares or contrasts. A number of authors who use Semantic Technology for access control however do not apply their approach to the RDF data model, detail their support for DAC Kodali et al. [17], Damianou et al. [10], Berners-Lee et al. [4]. However to the best of our knowledge to date no one has investigated the application of DAC to the RDF data model. We fill this gap by examining how DAC has been used to protect the relational and tree data models and by proposing strategies that allow us to apply DAC to the RDF graph model. We identify the resources to be protected and the access rights required, based

on the RDF data model and SPARQL [6] the predominant RDF query language respectively. In addition we propose mechanisms to assist with access control administration through derivation of authorisation, delegation of permissions and conflict resolution

5 Conclusions and Future Work

Although the RDF data model has been around for over a decade, little research has been conducted into the application of existing access control administration to RDF data. In this paper we discussed how the DAC model could be applied to RDF, a distributed graph based data model. We identified the resources to be protected and the access rights required based on the graph data model and SPARQL query operations respectively. We proposed a layered approach to authorisation derivation based on the graph structure and RDFSchema. We subsequently identified a number of rules that can be used to manage authorisations in an intuitive manner. Furthermore we demonstrated how SQL GRANT and REVOKE commands could be adapted to cater for authorisation administration over RDF data. As for future work, we propose to further investigate how enforcement of access control policies can be improved by exploiting the graph data structure and to examine complexity issues related to management of authorization over graph 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 Nuno Lopes and Aidan Hogan for their valuable comments on the paper.

References

1. Amini, M., Jalili, R.: Multi-level authorisation model and framework for distributed semantic-aware environments. IET Information Security 4(4), 301 (2010)
2. Angles, R., Gutierrez, C.: Survey of graph database models. Computing Surveys 1(212) (2008)
3. Astrahan, M.M., Blasgen, W., Chamberlin, D.D., Eswaran, K.P., Gray, J.N., Griffiths, P.P.: System R: Relational Management Approach to Database 1(2), 97–137 (1976)
4. Berners-Lee, T., Weitzner, D.J., Hendler, J.: Creating a Policy-Aware Web: Discretionary, Rule-based Access for the World Wide Web. Web and Information Security (2006)
5. Bertino, E., Sandhu, R.: Database security - concepts, approaches, and challenges. IEEE Transactions on Dependable and Secure Computing 2(1), 2–19 (2005)
6. Bertino, E., Castano, S., Ferrari, E.: Securing XML documents with Author-X. IEEE Internet Computing 5(3), 21–31 (2001)

[6] `http://www.w3.org/TR/sparql11-update/`

7. Bertino, E., Samarati, P., Jajodia, S.: Authorizations in relational database management systems. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, CCS 1993, pp. 130–139 (1993)
8. Bertino, E., Samarati, P., Jajodia, S., Member, S.: An Extended Authorization Model for Relational Databases 9(1), 85–101 (1997)
9. Costabello, L., Villata, S., Delaforge, N.: Linked data access goes mobile: Context-aware authorization for graph stores. In: 5th WWW Workshop on Linked Data on the Web, LDOW (2012)
10. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
11. Dietzold, S., Auer, S.: Access control on RDF triple stores from a semantic wiki perspective. In: ESWC Workshop on Scripting for the Semantic Web (2006)
12. Fagin, R.: On an authorization mechanism. ACM Transactions on Database Systems (TODS) 3(3), 310–319 (1978)
13. Gabillon, A.: An authorization model for XML databases. In: Proceedings of the 2004 Workshop on Secure Web Service, SWS 2004, pp. 16–28 (2004)
14. Griffiths, P.P., Wade, B.W.: An authorization mechanism for a relational database system. ACM Transactions on Database Systems (TODS) 1(3), 242–255 (1976)
15. Jain, A., Farkas, C.: Secure resource description framework: an access control model. In: ACM SACMAT, pp. 121–129 (2006)
16. Javanmardi, S., Amini, M., Jalili, R., GanjiSaffar, Y.: SBAC: A Semantic Based Access Control Model. In: 11th Nordic Workshop on Secure IT-systems (NordSec 2006), Linkping, Sweden (2006)
17. Kodali, N., Farkas, C., Wijesekera, D.: Multimedia access control using RDF metadata (2003)
18. Li, H., Zhang, X., Wu, H., Qu, Y.: Design and application of rule based access control policies. In: Proc. of the Semantic Web and Policy Workshop, pp. 34–41 (2005)
19. Lopes, N., Kirrane, S., Zimmermann, A., Polleres, A., Mileo, A.: A Logic Programming approach for Access Control over RDF. In: Technical Communications of ICLP 2012 (2012)
20. Qin, L., Atluri, V.: Concept-level access control for the Semantic Web. In: Proceedings of the 2003 ACM Workshop on XML Security, XMLSEC 2003, p. 94. ACM Press (2003)
21. Ryutov, T., Kichkaylo, T., Neches, R.: Access Control Policies for Semantic Networks. In: 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, pp. 150–157. IEEE (July 2009)
22. Kirrane, S., Lopes, N., Mileo, A., Decker, S.: Protect Your RDF Data! In: Proceedings of the 2nd Joint International Semantic Technology Conference (2012)
23. Sacco, O., Passant, A., Decker, S.: An Access Control Framework for the Web of Data. In: 10th International Conference on Trust, Security and Privacy in Computing and Communications (2011)
24. Samarati, P., de Capitani di Vimercati, S.: Access control: Policies, models, and mechanisms. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 137–196. Springer, Heidelberg (2001)
25. Sandhu, R.S., Samarati, P.: Access control: principle and practice. IEEE Communications Magazine (1994)
26. Tarjan, R.: Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)