Access Control on RDF Triple Stores from a Semantic Wiki Perspective

Sebastian Dietzold[1] and Sören Auer[1,2]

[1]Universität Leipzig
Department of Computer Science
{dietzold|auer}@informatik.uni-leipzig.de

[2]University of Pennsylvania
Department of Computer and Information Science
auer@seas.upenn.edu

Abstract. RDF triple stores are used to store and query large RDF models. Semantic Web applications built on top of such triple stores require methods allowing high-performance access control not restricted to per model directives. For the growing number of lightweight, scripted Semantic Web applications it is crucial to rely on access control methods which maintain a balance between expressiveness, simplicity and scalability. Starting from a Semantic Wiki application scenario we collect requirements for useful access control methods provided by the triple store. We derive a basic model for triple store access according to these requirements and review existing approaches in the field of policy management with regard to the requirements. Finally, a lightweight access control framework based on rule-controlled query filters is described.

Introduction

The efficient implementation of a Semantic Web application particularly depends on the underlying RDF API and triple store. Today's RDF triple stores are mostly build upon relational database management systems with dedicated database schemata and corresponding API methods which rewrite knowledge base queries into database queries (e.g. [1] and [2]). However, access control at the level of the underlying relational database lacks granularity if based on the entities table, row and database. There exist APIs which do not depend on a relational database (e.g. Redland [3] and Sesame [4]) but also when basing on them, the Semantic Web application has to establish access control mechanisms on their own.

We are convinced, that future RDF triple stores will be used as backends for application systems in analogy to existing relational databases. Assuming this, it is important to develop access control methods which do rely on the RDF data model and enable access controll with respect to metamodels based on the RDF data model, such as RDF-Schema and the different OWL flavors. A longer term aim is to make such methods integral part of future triple stores.

To have a more solid starting point for the formulation of requirements we selected the application scenario of Semantic Wikis. We define a Semantic Wiki as a collaborative software for modifying a shared knowledge base. Further, we assume this knowledge base to be an RDF graph which consists of RDF triples. A Semantic Wiki supports the collaborative process of instance acquisition and curation with respect to an often fuzzy and not well defined goal. This definition does not make any assumptions about the user frontend, because these wiki design principles are defined in [5] and are independent of the prefix "semantic".

1 Requirements on Access Control for a Semantic Wiki RDF Triple Store

In a 'classic' wiki we are used to think in access categories like account and page: an account has certain rights with respect to a page[1]. The specific rights which can be granted or revoked are typically the rights to read, modify, delete and annotate the page and to grant such rights for this page to some other account. Further, pages can be arranged in a page tree, which makes the application of access control rules on a page subtree necessary.

For a Semantic Wiki, the base entities we have to consider are not accounts and pages but sets of triples. Also, accounts and annotations can be identified using URI references and information about them represented as triples. Hence, access control should work on the granularity level of triples as well as on higher levels, such as the description of a resource (i.e. all triples having the same subject) or instances of a certain class.

Based upon our Semantic Wiki definition and the work with our prototype 3ba.se [6] we identified the following requirements for access control on the underlying RDF triple store:

- Efficiency and scalability should have precedence over expressive power. In modern web applications with complex and dynamic user frontends, query processing has to be as fast as possible. This requirement is more important than expressive power of the access control language since there are usually hundreds of queries to the store triggered by a single web request.
- As a minimal requirement we need context- and content-sensitive triple filtering in a declarative way. This means the access to a tiple set depends on the accounts metadata (e.g. membership information) as well as on the content of the wanted triple set itself (e.g. enforce to give all needed attributes to some resource).
- Access control declaration should be able to use organisational information like command structure and group membership information from inside the controlled or another RDF model. For the most common architectures used for the storage of organisational data inside a company etc., methods exist to migrate or RDF-ify such organizational memory (e.g. [7] for relational

[1] It is important to distinguish between an idealistic wiki with absolutely no access control and realistic wikis, where access control and wiki are not mutually exclusive.

databases and [8] for LDAP directories). It should be possible to use this data to express access control declarations.

In the next section we survey shortly existing research projects which are related to the topic of access control on RDF triple stores.

2 Related work

The research field policy management for the Semantic Web addresses machine interpretable policies to control programs, services and agents on the Web. It is not restricted to security and privacy but also tackles problems related to trust (e.g. trust in resource quality or agents), information filtering, accountability and others. An overview of the current projects is available in the workshop proceedings [9] and [10]).

However, most projects have a different intentions than this work. A policy-based management framework in the sense of [11] aims at an open semantic network environment. In this network the behavior of agents and services is controlled by reasoned decisions over policies. This is necessary due to the complexity of the global approach of controlling all possible agents and services with all possible actions. An example for such a system is Rei [12] which supports specification of policies, analysis and reasoning in pervasive computing applications.

Due to the fact that reasoning procedures are still not scalable to scope with larger knowledge bases, such capabilities can not applied in RDF triple stores today. Another objection to reasoning is the open world assumption, because no external sources are used and access control answers in a closed triple store are limited to yes and no.

Policy management is not only access control but also information filtering based on quality and trust properties. This is necessary whilst operating in a network of distributed resources which are not trustworthy per default. The TriQL.P [13] browser uses queries for filtering information from different sources and qualities. This filter approach is also part of the framework described here.

Another possible approach is the usage of explicit rules (which our approach also makes use of). An example for such a system is [14]. Again, the scope of this system is not an RDF triple store but distributed resources on a network and the access to these resources.

Summarizing we can state that all these systems operate with a different communication model. However, an RDF triple store can be seen as an agent in these frameworks, while the access control layer for the RDF triple store itself operates on a more basic and lightweight model.

3 A Basic Model for Access on RDF Triple Stores

In order that we can develop an access control framework which solves the given requirements, we have to specify a clear communication model for the target

environment. In this basic model for access on RDF triple stores, we define three atomic actions:

- *Reading a set of triples* from a stored model: The account queries the triple store with a formal query language (e.g. [15], [16] and [17]) or selects some triples with a more simple method (e.g. a triple pattern). The answer of the triple store is a set of triples which constitutes the intersection of the wanted and the allowed triple set. Most of todays query languages can query not only for submodels. A common result value is a set of variable bindings. Nevertheless there was a requested set of triple which was necessary for the computation of the result set.
- *Adding a set of triples* to a stored model: The account sends the dedicated triple set to the triple store. The store removes all the triples which are not allowed to be written and adds each of the remaining triples, if they are not already contained in the model.
- *Removing a set of triples* from a stored model: The account sends the dedicated triple set to the triple store. The store removes all the triples which are not allowed to be deleted and deletes each of the remaining triples, if they exist in the stored model.

The approach presented in [18] adds more atomic actions here to the above listed ones. They distinguish between one-triple actions, triple-set actions and reasoned-set actions. In this basic model neither one-triple actions nor reasoned-set actions need to be considered because:

- A one-triple action can be seen as a specialization of a triple-set action.
- We define a reasoner application as an agent which holds a specific account with certain rights to a triple store. Following this, reasoned-set actions are combinations of normal triple-set actions which are performed by the reasoner agent.

In the next section, we describe an access control framework which is based on this communication model.

4 Lightweight Framework for Access Control on RDF Triple Stores

As denoted in section 2, we use explicit rules and query filters as the primarily parts of our framework. The whole framework consists of four parts:

- A *query engine* which can apply subset selection query filters to a given model. In this paper we assume that this is a query engine for the SPARQL query language [15] but generally the approach is not limited to a specific query language.
- A *rule processor* which decides whether a query filter is fired for a given action or not. We assume that the used decision rules are described by using the Semantic Web Rule Language (SWRL, [19]) but also, this part can be replaced by an equivalent one.

– A minimalistic *RDF schema* called Lightweight Access Control Schema (LACS, [20]), which describes a basic vocabulary to store rules and query filters.
– The *access control processor*, which starts the query engine and rule processor as needed and maintains some session data.

A fundamental concept of the framework is the presentation of a virtual model to the account instead of the real one. This virtual model is created from the real model and modified through the query filters selected by the rule processor. Thereby, the decision rules can reference to and use resources from the following three different models:

– *Session Model*: This model holds information about the active session (which account is doing what). The triple of this model are dynamically created for every new action on the triple store.
– *User model*: This is the data which the account wants to get access to but it can used by the decision rules too.
– *Maintenance Model*: This model consists of decision rules and filters as well as all other maintenance data like group or account information. The vocabulary for the filter and rule description comes from the lightweight access control schema and from the SWRL specification. The maintenance data which is used by the rules to decide the application of a query filter is not fixed, so rules can be created for every available environment, e.g. a FOAF database or an LDAP backend.

The following example maintenance model consists of two filters and rules. They are created for the following two reading conditions:

– All admins can read every triple.
– All accounts which are from type `foaf:Person`[2] may read only triples where the subject is of type `foaf:Person`.

The rules to effect this behavior are:

$$\begin{aligned} & \texttt{rdf:type}(\texttt{lacs:CurrentAction}, \texttt{lacs:Read}) \\ \wedge\ & \texttt{rdf:sameAs}(\texttt{lacs:CurrentAccount}, ?a) \\ \wedge\ & \texttt{foaf:member}(\texttt{:Admins}, ?a) \\ \rightarrow\ & \texttt{lacs:addAndStop}(\texttt{:currentAction}, \texttt{:AllFilter}) \end{aligned}$$

$$\begin{aligned} & \texttt{rdf:type}(\texttt{lacs:CurrentAction}, \texttt{lacs:Read}) \\ \wedge\ & \texttt{rdf:sameAs}(\texttt{lacs:CurrentAccount}, ?a) \\ \wedge\ & \texttt{rdf:type}(\texttt{foaf:Person}, ?a) \\ \rightarrow\ & \texttt{lacs:add}(\texttt{:currentAction}, \texttt{:FoafOnlyFilter}) \end{aligned}$$

They reference triples in the maintenance model, which describe a group and a member of this groups with the commonly used FOAF vocabulary:

[2] We assume, that the namespaces `rdf`, `rdfs`, `foaf` and `ruleml` are predefined for all examples. The namespace `lacs` is used for the vocabulary described in [20]. All RDF examples are given in Notation 3 (N3, [21]).

```
:Admins a foaf:Group;
  foaf:member :UserSD.

:UserSD a foaf:Person;
  foaf:name "Sebastian Dietzold".
```

These rules reference two query filters. These query filters are given in a specific query syntax and are represented in the RDF with the LACS vocabulary:

```
:AllFilter a lacs:Filter;
  rdfs:label "no restriction filter";
  lacs:sparql "CONSTRUCT { ?s ?p ?o } WHERE { ?s ?p ?o }".

:FoafOnlyFilter a lacs:Filter;
  rdfs:label "read only FOAF address book";
  lacs:sparql """CONSTRUCT { ?s ?p ?o }
    WHERE {?s rdf:type foaf:Person . ?s ?p ?o }""".
```

To give explicit instructions for the access control processor, the rules are represented in RDF and enriched with metadata. The first one is annotated by using the SWRL vocabulary referenced by the namespace `ruleml`, the latter by using the LACS vocabulary.

The next part defines two `lacs:rule` entities which references to SWRL implication rules (not displayed here). Important for the access control processor is the priority of the rules, since the rule selection (see figure 1) is ordered by this property.

```
_:123 a lacs:Rule;
  rdfs:label "Admins can read everything";
  lacs:priority 10;
  lacs:swrlImp [
    a ruleml:imp;
    # ... rule definition ...
    ].

_:321 a lacs:Rule;
  rdfs:label "User can read only foaf:Persons";
  lacs:priority 100;
  lacs:swrlImp [
    a ruleml:imp;
    # ... rule definition ...
    ].
```

Based on this example maintenance model, a sample reading action is processed according figure 1.

First of all, the Session Model is modified by the access control processor to represent the current session. Again, the LACS vocabulary is used:

```
lacs:currentUser = :User2.
lacs:currentAction a lacs:Read.
```

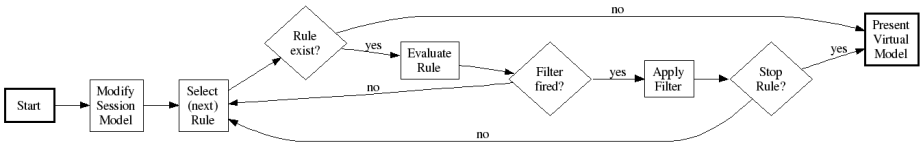

Fig. 1. Rule Processing Flowchart

After that, the first rule is selected and evaluated by the rule engine. In the example maintenance model, this is the "Admins can read everything" rule. Due to this rule, no filter is fired because the user is not member of the admin group. The next rule will be selected and due to this rule, the filter "read only FOAF address book" is fired. So the query process creates the virtual model as it applies filters to the user model. Because there is no other rule, the processor leaves this cycle and presents the virtual model to the user.

This was an example for a reading action. For this type of action, the user query is processed against the virtual model. For writing actions, the filters are not processed against the user model but rather against the model which is supplied by the user (i.e. the triples, he wants to add or delete). After modifying this model according the rules, the add or delete action is processed.

5 Conclusion

We have presented a lightweight access control framework for RDF triple stores based on requirements derived from usage scenarios within a Semantic Wiki application. The basic idea of this framework is the presentation of a virtual model instead of the real one. This model is generated by filtering the original model. Filter are selected by rules. In the examples, we use SPARQL for filtering and SWRL as rule language.

The presented framework strongly depends on the lightweight communication model given in section 3. So it is not intended to be a general access control framework for the Semantic Web. Instead it is designed to be a fast, reliable and easy to implement as part of an RDF triple store. In order to achieve this, we focused on a clear execution algorithm with explicit rules, but do not use any reasoning capabilities.

One important advantage when compared to other approaches is the possibility to create both simple and complex access control environments as necessary. Also, the minimal requirements to the underlying maintenance model are small, so that administrators can maximally reuse existing models within their rules.

References

1. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Proceedings of First International Workshop on Semantic

Web and Databases 2003. (2003) 131–150
2. Bizer, C.: RAP (RDF API for PHP). Website (2004) http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/.
3. Beckett, D.: The design and implementation of the Redland RDF application framework. Computer Networks **39** (2002) 577–588
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Horrocks, I., Hendler, J., eds.: The Semantic Web - ISWC 2002. First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings. Volume 2342 of Lecture Notes in Computer Science., Springer (2002) 54–68
5. Leuf, B., Cunningham, W.: The Wiki Way. Addison-Wesley Longman, Amsterdam (2001)
6. Auer, S., Dietzold, S., Riechert, T.: 3ba.se Semantic Wiki. Prototype (2006) http://3ba.se.
7. Bizer, C., Seaborne, A.: D2RQ -Treating Non-RDF Databases as Virtual RDF Graphs. Poster (2004) 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan.
8. Dietzold, S.: Generating RDF Models from LDAP Directories. In Auer, S., Bizer, C., Miller, L., eds.: Proceedings of the SFSW 05 Workshop on Scripting for the Semantic Web , Hersonissos, Crete, Greece, May 30, 2005. Volume 135 of CEUR Workshop Proceedings., CEUR-WS (2005)
9. Kagal, L., Finin, T., Hendler, J., eds.: Policy Management for the Web. (2005)
10. Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005)
11. Tonti, G., Bradshaw, J.M., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In Fensel, D., Sycara, K.P., Mylopoulos, J., eds.: The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings. Volume 2870 of Lecture Notes in Computer Science., Springer (2003) 419–437
12. Kagal, L.: Rei: A Policy Language for the Me-Centric Project. Technical report, HP Labs (2002)
13. Bizer, C., Cyganiak, R., Gauss, T., Maresch, O.: The TriQL.P Browser: Filtering Information using Context-, Content- and Rating-Based Trust Policies. In Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005) 12–20
14. Li, H., Zhang, X., Wu, H., Qu, Y.: Design and Application of Rule Based Access Control Policies. In Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005) 34–41
15. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (Working Draft). W3c working draft, World Wide Web Consortium (W3C) (2006)
16. Seaborne, A.: RDQL - A Query Language for RDF. W3c member submission, World Wide Web Consortium (W3C) (2004) http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
17. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the eleventh international conference on World Wide Web, ACM Press (2002) 592–603

18. Reddivari, P., Finin, T., Joshi, A.: Policy based access control for an RDF store. In Kagal, L., Finin, T., Hendler, J., eds.: Policy Management for the Web. (2005) 78–81
19. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, World Wide Web Consortium (W3C) (2004) http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.
20. Dietzold, S.: LACS: Lightweight Access Control Schema. OWL ontology (2006) http://purl.org/net/lacs.
21. Berners-Lee, T.: Notation 3 - An readable language for data on the Web. Website (1998) http://www.w3.org/DesignIssues/Notation3.html.