Combining Public and Private Linked Data through Graph-based Authorization Profiles in the semantic.works Framework

Tom De Nies, Aad Versteden, Erika Pauwels and Johan Delaure

redpencil.io – email: firstname.last.name@redpencil.io

Abstract

The semantic.works framework enables developers to build applications using Linked Data, eliminating the need to rely on conversions to disseminate it. However, in many real-world applications not all of this data is public, either temporarily or permanently. To mitigate this, the mu-authorization microservice allows the configuration of authorization groups and the types of data they have access to. In the background, this means that all data will be distributed across RDF graphs corresponding to these profiles. This process is completely transparent for other microservices in the stack, unburdening application developers from having to implement this logic. Additionally, this allows for timed distribution of data by other microservices, since all that needs to be done to provide access to a certain profile is to write data into the right graph. In this paper, we describe our approach to authorization for SPARQL, show that it has been used in real-world applications, and discuss the challenges we face.

1. Introduction

Semantic.works offers an open source microservice framework in which services communicate through a shared semantic model, resulting in a maintainable and extendable architecture for web applications. In this paper, we focus on the mu-authorization[1] service, which allows public and private data to be combined in a containerized triplestore, and the challenges this presents.

2. Our Approach

In many of our projects, access to certain data must be restricted. For example, confidential documents are only accessible to certain users, work-in-progress data must not be shared until it is finalized, digitally signed documents cannot be published whereas their unsigned counterparts can, etc. To achieve this using a triplestore, *mu-authorization* places a layer in front of the SPARQL endpoint. Data is organized into graphs, and the access to these graphs is restricted to a certain set of roles, each of which is associated with a group of users. Our service then rewrites queries based on the session information of the user and the access rights on the data. Paired with identification & authentication[2], this guarantees that other microservices can transparently query the triplestore without having to worry about access to restricted data.

QuWeDa 2023, 7th Workshop on Storing, Querying and Benchmarking Knowledge Graphs

[1]Documentation and code available at https://semantic.works/ & https://github.com/mu-semtech/mu-authorization
[2]See https://github.com/mu-semtech/mu-identifier and https://github.com/mu-semtech/login-service

To determine the access policy: i.e., which users have access to which data, group access rules are specified in a configuration file. Groups have **usage restrictions** that can be set to *read*, *write*, and/or *read for write*[3]. These restrictions are paired with **access rules** to determine whether or not the user is part of this group, as well as with **graph specifications** and **resource constraints** to know which data can be read from/written to which graphs. For example, a simple group specification for read-only, public data would be configured as follows:

```
%GroupSpec{
  name: "public",
  useage: [:read],
  access: %AlwaysAccessible{},
  graphs: [ %GraphSpec{ graph: "http://example.org/public" } ]
}
```

Two types of **access rules** are available: *AlwaysAccessible* and *AccessByQuery*. *AlwaysAccessible* is used for public data, as it simply allows access to the data for this group for all users regardless of their session information. *AccessByQuery* on the other hand, provides a flexible way to determine whether or not a user has access to the data, for example:

```
%AccessByQuery{
  vars: ["role"],
  query: "PREFIX ex: <http://example.org/>
    PREFIX org: <http://www.w3.org/ns/org#>
    SELECT ?role WHERE { <SESSION_ID> ex:user / org:role ex:Admin . }"
}
```

The only limit here is the expressiveness of a SPARQL query, so more complex criteria are also possible, e.g.: membership of an organization, a minimum age requirement, a valid subscription, etc. However, there are caveats to more complex authorization rules, discussed in Section 3.

Graph specifications define which graph is created and which triples are added to this graph by a group, based on two kinds of constraints on the data: *ResourceFormatConstraints* and *ResourceConstraints*. A *ResourceFormatConstraint* defines a constraint on the prefix of the subject URI of a triple. For example, this graph specification will send all triples with a subject that starts with `http://example.org/clients/` to the `http://example.org/sales` graph:

```
%GraphSpec{
  graph: "http://example.org/sales",
  constraint: %ResourceFormatConstraint{
    resource_prefix: "http://example.org/clients/"
  }
}
```

A *ResourceConstraint* allows to put a constraint on the resource type (`rdf:Class`) and/or the predicates. For example, the following graph specification makes sure no other personal information than the name and account name are read from/stored in the graph:

[3]The data can be read while doing update queries only, useful e.g., for writing login activities to a users graph

```
%GraphSpec{
  graph: "http://example.org/sales",
  constraint: %ResourceConstraint{
    resource_types: [ "http://xmlns.com/foaf/0.1/Person" ],
    predicates: %NoPredicates{
      except: [ "http://xmlns.com/foaf/0.1/name",
        "http://xmlns.com/foaf/0.1/accountName" ]
    }
  }
}
```

3. In Use & Challenges

At the time of writing, mu-authorization is used in projects[4] in the public sector, which have a strong focus on disseminating Linked Data as part of an effort towards openness of government. Most notably, these projects include Kaleidos, a software platform to support the decision making process of the Flemish government; and LBLOD, a platform to increase the quality of the data streams between local administration, the Flemish government, and the public. Both of these projects combine public and private data. In Kaleidos, decisions are drafted and internally discussed before they are released to governmental agencies & the general public. Additionally, there are currently 5 incremental access levels for documents and 8 different user roles, going from ministers & their supporting staff, to employees at various agencies, which makes the need for a flexible authorization service very clear. In LBLOD, most data is publicly readable, but only authorized organizations and mandataries can submit new data. In these projects, the following technical challenges emerged as a consequence of the authorization layer.

(Timed) data distribution: When a group must have read access to parts, but not all of the data that is written to a graph, resource constraints are not enough to ensure confidentiality when it is needed. Additionally, for governments, the release of information to the public is often mandatory at specified times. E.g., in the Kaleidos project, the decisions made during the Flemish council of ministers must be released the same day, while the documents are released at 2pm the next day. To address this, mu-authorization calculates *deltas*[5] for each update query it executes, and forwards this delta to interested clients. E.g., in Kaleidos, where different levels of government have different access levels to confidential documents, these deltas are consumed by a custom distribution service. This service copies the document-related triples to the appropriate graphs based on their access level. Additionally, this service interprets publication times for decisions of the council, automatically releasing these decisions to government agencies and eventually, the public at predetermined times.

Data duplication: Copying data to graphs to accommodate timed distribution introduces an additional challenge: when a portion of the data that is distributed across graphs is identical, this creates significant data duplication, scaling linearly with the number of groups that need access to the same data at different times. E.g, in Kaleidos, each public resource starts in the most

[4]https://redpencil.io/projects
[5]i.e. which triples have been added and which have been removed

restricted graph, and is then gradually made public across 3 other graphs, going from most to least restricted. Since the most restricted graph in this application has approx. 15 million triples, and the least restricted graph holds 13.1 million, we estimate 87.3% of the data is duplicated across all graphs. While this does reduce query execution overhead, as each user group only has one graph to select from, it is unclear whether the benefits in query performance outweigh the downside of added storage. To minimize the duplication needed for each group, we are currently exploring separating the 'active' data from the 'static' data in individual graphs.

Query execution overhead: As with any query rewriting approach, a certain increase in query execution time is to be expected. The actual performance depends highly on the strategy that is chosen when distributing data and specifying authorization groups. Storing disjoint data in separate graphs and allowing multiple groups to access these graphs might minimize data duplication, but could increase query execution time. This results in a delicate balancing act, for which we have yet to find an optimum. A full benchmark on added execution time would be out of scope for this paper, and is part of our future work.

Indexing: Finally, most web applications require some form of search engine, including our projects. While semantic.works supports integration of Elastic search[6], special attention must be given to the authorization groups. Originally, each group required its own search index, resulting in a quickly increasing indexing time. Recently, an update was made to the mu-search service, which allows the specification of *additive indexes*, enabling the indexer to re-use indexes for authorization groups that share graphs. In the case of Kaleidos, this optimization allowed us to reduce the indexing time from 36 hours to 16 hours.

4. Related Work

Since a complete literature overview is well beyond the scope of this short paper, we refer to Kirrane et al.[1], Zdraveski et al.[2], and more recently, da Silva[3] for a comprehensive overview of related work in this field. Our approach can be classified between what Kirrane et al. call *Role Based Access Control* and *Context Based Access Control*. While we do allow contextual access rules defined using SPARQL, in practice the complexity is often limited to querying for user roles to avoid the downsides we discussed in Section 3. Finally, we are looking for optimizations to rewrite queries in the field of federated querying, e.g., as in Werbrouck et al., where the use of `FROM NAMED` instead of `FROM` showed significant performance gains for disjoint graphs[4].

References

[1] S. Kirrane, A. Mileo, S. Decker, Access control and the resource description framework: A survey, Semantic Web 8 (2016) 1–42. doi:10.3233/SW-160236.

[2] V. Zdraveski, D. Trajanov, R. Stojanov, S. Stojanova, M. Jovanovik, Ranking semantic web authorization systems, Semantic Web (2017).

[3] T. G. da Silva, Access control in linked data archives (2023).

[4] J. Werbrouck, P. Pauwels, J. Beetz, R. Verborgh, E. Mannens, Consolid: A federated ecosystem for heterogeneous multi-stakeholder projects, Semantic Web (2022) 1–32.

[6]https://github.com/mu-semtech/mu-search