Fine-Grained Access Control for RDF data on Mobile Devices*

Owen Sacco[1], Matteo Collina[2], Gregor Schiele[1], Giovanni E. Corazza[2], John G. Breslin[1,3], and Manfred Hauswirth[1]

[1] Digital Enterprise Research Institute, Galway, Ireland
{owen.sacco, gregor.schiele, manfred.hauswirth}@deri.org
[2] University of Bologna, Italy
{matteo.collina, giovanni.corazza}@unibo.it
[3] National University of Ireland, Galway
{john.breslin}@nuigalway.ie

Abstract. Existing approaches for fine-grained access control for RDF data suffer from high overhead, making them ill-suited for mobile devices. This makes it difficult to develop mobile applications that manage personal RDF data in a privacy preserving manner. In this paper we propose a new approach to realise fine-grained access control for mobile devices. We show how fine-grained privacy settings for personal information stored in mobile devices can be described using the Privacy Preference Ontology (PPO) – a light-weight vocabulary for defining fine-grained privacy preferences. Moreover, we introduce a two stage privacy preservation approach for efficient filtering of personal information on mobile devices. Our approach combines (1) an efficient query-based analysis stage with (2) a result filtering stage based on the privacy preferences described using PPO.

Keywords: Semantic Web, Access Control, Mobile, Privacy, PPO, PPM, Gambas

1 Introduction

Web information systems are getting mobile. Due to more powerful mobile devices like smartphones and tablets, users increasingly let them manage and publish their personal data like social network information or sensor readings. This, in combination with the increasing popularity of Linked Data technologies like RDF and SPARQL, has led to the need to develop efficient data storage systems (i.e. RDF stores) for mobile devices. However, current storage systems such as RDF on the Go [8] do not offer efficient support for fine-grained access control for the data contained in them. This makes it difficult to develop mobile applications that manage personal RDF data in a privacy preserving manner. Existing

* This work is funded by the Science Foundation Ireland (grant SFI/08/CE/I1380 (Líon 2)), by an IRCSET scholarship co-funded by Cisco systems and by the European Commission (FP7, contract FPF-2011-7-287661, GAMBAS).

approaches for access control for RDF data either suffer from high overhead, making them ill-suited for mobile devices – or do not provide fine-grained control, i.e. the ability to control access at the level of individual triples (e.g. by filtering triples resulting from SPARQL queries). For instance, most rule-based access control systems focus on applying access control policies to actions rather than directly to the underlying data [5]. This requires creating different actions for defining how to interact with the data. Moreover, for each type of action, a rule must be created to define a particular action and another rule is created for defining an access control policy for that action. This results in a large amount of rules that need to be processed. In previous work [13, 11], we presented a Privacy Preference Ontology (PPO) – a light weight vocabulary for defining privacy preferences for RDF data – and a Privacy Preference Manager (PPM) [12, 15] enforcing them. However, this system induces large overhead on mobile devices.

Based on our previous work, in this paper we propose a new approach for fine-grained RDF data access control that is specifically designed for mobile devices. Our approach allows users to specify privacy preferences and enforces them when RDF data is accessed. It is co-located with the data and executed entirely on the user's mobile device. No external server support is needed, giving the user full control over his/her data at any time without trusting an external party. Our approach is based on RDF and SPARQL, modelling privacy preferences with RDF and checking them with SPARQL queries. This allows us to reuse the full power of RDF/SPARQL support in the existing RDF store on the mobile device without the need to add an additional reasoner or specific parser to process language specific rules. At the same time we retain the expressiveness of access control policies. Our approach does not assume any special support from the RDF store and can be used on top of any RDF store that offers support for SPARQL. To filter RDF triples we introduce a novel two stage approach that combines (1) an initial efficient query analysis stage that extracts the necessary metadata about the query and the (2) filtering phase that filters the result set without having to access the store for additional metadata (about the query). Our evaluation shows that this improved filtering algorithm results in a 10 times increase in system performance compared to our previous approach.

The paper is structured as follows: Section 2 gives a short overview on our target scenario and our assumptions as well as our privacy preference ontology PPO and privacy preference manager PPM. Based on this, Section 3 presents the current PPM filtering algorithm which was not published in our previous work. Then, we introduce our new improved filtering algorithm in Section 4. Section 5 presents evaluation results. Finally, Section 6 gives an overview of related work before we wrap up the paper with future work and a conclusion in Section 7.

2 Access Control for RDF Stores on Mobile Devices

In this work, we are focusing on how access to personal user data that is stored on mobile devices can be protected. To clarify our target scenario, consider two friends Alice and Bob who want to exchange personal data with their smartphone

devices. Each device by default, denies access unless otherwise instructed by its user. Alice uses her smartphone, contacts Bob's smartphone and asks for his location. Bob receives a notification on his smartphone that Alice has requested to access his location. Bob grants Alice access and this privacy preference is stored in his smartphone. Alice can now retrieve and view Bob's location on her smartphone. Other data is still not accessible. Next time Alice requests to view Bob's location, if the request matches Bob's stored privacy preference, then she is automatically granted (or denied) access. Otherwise, Bob is notified about Alice's new request and decides whether to grant her access or not.

To realise this example we propose an access control system for RDF stores on mobile devices. By storing the data directly on the users' mobile devices, users can have full control over their data without trusting any external server or provider. However, the access control algorithms must be executed on the mobile device, too and thus they must be very efficient to respond in a timely fashion and not waste battery life. We assume that an RDF store is installed on the mobile devices and that user data is modelled as RDF triples using vocabularies such as FOAF[4] for describing user profiles, SIOC[5] for describing microblog posts, OGP[6] for describing activities in Social Networks and the WGS84 Geo Positioning vocabulary[7] for defining data related to locations. While data from major websites is generally not modelled directly in RDF, mapping wrappers can easily be implemented through the websites' APIs. This is beyond the scope of this paper and we assume that necessary mappings have already been done. Our approach models access control policies for RDF data using the Privacy Preference Ontology (PPO). PPO is non-domain specific and can model privacy preferences for any RDF scenario. In this section, we provide an overview of PPO and we explain how we model privacy preferences using it. Subsequently, we describe how the Privacy Preference Manager (PPM) enforces such privacy preferences by filtering out RDF data based on them. The PPM is datastore independent and therefore can be easily customisable to provide fine-grained access control to any datastore.

2.1 Privacy Preference Ontology (PPO)

PPO[8] [11, 13] is a *light-weight* Attribute-based Access Control (ABAC) vocabulary that allows users to describe fine-grained privacy preferences for restricting or granting access to non-domain specific Linked Data elements, such as Social Semantic Data. Considering that PPO is described in RDF(S), it does not require a specific parser or reasoner but it retains the expressivity of fine-grained access control policies similar to rule-based approaches. Among other use-cases, PPO can be used to restrict part of FOAF profile records to users that have

[4] Friend-of-a-Friend (FOAF) – `http://www.foaf-project.org`
[5] Semantically-Interlinked Online Communities (SIOC) - http://sioc-project.org/
[6] `OpenGraphProtocol(OGP)\url{http://ogp.me/}`
[7] WGS84 – `http://www.w3.org/2003/01/geo/wgs84_pos#`
[8] PPO – `http://vocab.deri.ie/ppo#`

```
PREFIX ppo:   <http://vocab.deri.ie/ppo#> .
PREFIX bob:   <http://vmuss13.deri.ie/userprofiles/bob/> .
PREFIX alice: <http://vmuss13.deri.ie/userprofiles/alice/>
PREFIX wgs84: <http://www.w3.org/2003/01/geo/wgs84_pos#>
bob:PrivacyPreferences#1 a ppo:PrivacyPreference;
ppo:hasCondition [
  ppo:classAsSubject wgs84:SpatialThing];
ppo:assignAccess acl:Read;
ppo:hasAccessSpace [
  ppo:hasAccessAgent alice:UserProfiles#me].
[...]
```

Fig. 1. Bob's privacy preference to grant Alice his location

specific attributes. It provides a machine-readable way to define settings such as "Provide my location only to my family" or "Grant read access to my activity only to Alice".

As PPO deals with RDF(S)/OWL data, a privacy preference defines: (1) the resource, statement, named graph, dataset or context it must grant or restrict access to; (2) the conditions refining what to grant or restrict (for example defining which instance of a class as subject or object to grant); (3) the access control privileges (including `Create`, `Read`, `Write`, `Update`, `Delete` and `Append`); and (4) an `AccessSpace`, defined by either an agent or a SPARQL query that specifies a graph pattern that must be satisfied by the requesting user.

2.2 Modelling Privacy Preferences for RDF data using PPO

Figure 1 illustrates Bob's privacy preference that restricts his location only to Alice. The location is modelled as an instance of type `SpatialThing` which includes longitude and latitude. Hence the privacy preference is applied to any resource of this type – in our case, Bob's location. Although the PPO is designed as an attribute-based model (using SPARQL queries) to test whether requesters satisfy particular attributes, it also enables users to specify specific people (as agents) to whom to grant or deny access. In this example Alice is granted the `read` access to Bob's location.

2.3 Privacy Preference Manager (PPM)

The PPM [12, 15] is an access control manager that allows users to create privacy preferences for RDF data. The manager also filters the requested data by returning only a *subset* of the requested data containing only those triples that are granted access as specified by the privacy preferences. The PPM was developed as a Web application – either as a centralised Web application or in a federated Web environment. The privacy preferences are stored separately from the data and can only be accessed by the PPM.

```
Data: resultSet and privacyPreferencesList
Result: (1) protectedTriplesList; (2) unprotectedTriplesList;
        (3) accessAgentsList; and (4) accessPrivilegesList.
List<PrivacyPreference> pList ← privacyPreferencesList;
List<Triple> rs ← resultSet;
Triple t ← new Triple();
PrivacyPreference p ← new PrivacyPreference();
forall the t ∈ rs do
    forall the p ∈ pList do
        if p.Match(t) then
            pURI ← p.getPrivacyPreferenceURI();
            aURI ← p.getAgentURI();
            privilege ← getAccessPrivilege();
            protectedTriplesList.add(t, pURI);
            accessAgentsList.add(aURI, pURI);
            accessPrivilegesList.add(privilege, pURI);
        else
            unprotectedTriplesList.add(t);
        end
    end
end
```

Algorithm 1: Privacy Preferences and Triples Matching

Users can log into their PPM with WebID [16]. Once logged in, users can create privacy preferences for their RDF data. They can also log into another user's PPM and request data from it. The other user's PPM would return back only that data which the user is granted access – based on the privacy preferences.

The PPM contains various modules including: (1) an authentication module for users logging into the manager; (2) a user interface that allows users to interact with the manager; (3) an RDF retriever and parser module for extracting and parsing RDF data from RDF datasources; (4) a privacy preference creator module for creating privacy preferences; and (5) a privacy preference filtering module that ensures that the access control polices are enforced. The PPM also offers an API which could be integrated within other applications.

Although the PPM is suited for Web environments, it is not originally designed for operating on mobile devices due to their limited resources – such as processing power, memory resources and battery life. To port the PPM to mobile devices we modified the filtering module substantially to reduce the number of querying operations needed for filtering. First, we rewrote the module using a memorisation technique to avoid looking up the type of the same resource multiple times. Clearly, this is a time vs memory tradeoff. In addition we designed a new filtering algorithm that extends our previous one to further reduce the number of queries. In the subsequent sections we first explain the original filtering algorithm and outline the parts which are resource expensive. We then provide our extended algorithm and evaluate both of them.

```
Data: subject URI or object URI of the triple and restricted class
Result: boolean isInstance – i.e. whether the subject or object is an instance of
        the class
query ← "SELECT ?o WHERE <subject URI ∨ object URI of restricted
triple> rdf:type ?o";
result ← executeQuery(query);
if (result ≠ restrictedClass) then
    remote ← getEndpoint(subject ∨ object);
    remoteResult ← remote.executeQuery(query);
    if remoteResult ≠ restrictedClass then
        isInstance ← false;
    else
        isInstance ← true;
    end
else
    isInstance ← true;
end
```

Algorithm 2: Class Matching

3 PPM Access Control Filtering Algorithm (PPF-1)

The PPM access control filtering algorithm (called PPF-1 in this paper) consists of (1) a matching part which maps the triples in the requested result set to the specific privacy preferences that apply to the triple; and (2) a filtering part that filters the result set by checking which triples a requester is granted access. This algorithm was not published in our previous work and therefore in this section we provide a detailed overview.

Initially, PPF-1 expects a list of requested triples together with the named graph they reside in. Moreover, the set of privacy preferences related to the data in the store is also passed to the algorithm. With these, PPF-1 first matches the triples to their corresponding privacy preferences; then, it checks what the requester can access and grants the requester a filtered result set. The following sections describe the different parts of PPF-1 in more detail: Section 3.1 describes the matching part and Section 3.2 describes the filtering part.

3.1 Privacy Preferences and Triples Matching

Algorithm 1 illustrates the matching between triples and privacy preferences. This part iterates through every triple in the result set and for every triple it checks all the privacy preferences to match which ones apply to the triple. The algorithm checks whether each privacy preference applies to: (1) the named graph in which the triple resides; (2) a resource in the triple; and (3) a rectified statement – i.e. the triple's subject, predicate and object.

The algorithm checks whether each privacy preference has a condition that specifies: (1) the resource must be the subject of the triple; (2) the resource must be the object of the triple; (3) the subject of the triple must be an instance of a

```
Data: protectedTriplesList
Result: (1) accessTriplesList(triple, privilege) (2) noAccessTriplesList(triple)
Iterator<ProtectedTriple> pIterator = protectedTriplesList.Iterator();
while pIterator.hasNext() do
    pt ← pIterator.next();
    forall the agent ∈ accessAgentsList do
        if pt.privacyPreferenceURI = agent.privacyPreferenceURI then
            if ¬(pt.Triple ∈ accessTriplesList) then
                privilege ← accessPrivilegesList.Privilege;
                accessTriplesList.add(pt.Triple, privilege);
            end
        else
            noAccessTriplesList.add(pt.Triple);
        end
    end
end
```

Algorithm 3: Privacy Preferences Filtering

certain class; (4) the object of the triple must be an instance of a certain class; (5) contains a particular predicate; and (6) contains a particular literal.

For most of these checks, the values in both the requested triples and in the privacy preferences are tested to check whether they are both the same. However, for testing whether a subject or object of the triple are instances of a particular class, the algorithm queries the store each time a privacy preference (for each triple) is tested. This part is explained in Algorithm 2.

Algorithm 2 checks whether the subject or object of a requested triple are instances of a class specified in a privacy preference. This algorithm is called by algorithm 1 that passes the subject or object of the triple and the restricted class specified in the privacy preferences as parameters. The algorithm constructs a query that gets the class type of the subject or object. If the class type matches with the restricted class then the algorithm returns true to Algorithm 1. Otherwise it returns false. If the result of the query does not contain any result (i.e. `result = null`), then the algorithm fetches the endpoint URI of the datastore in which the class types for the subject or object are specified. The endpoint URIs are mapped to the subjects and objects. Once the class type is retrieved, the algorithm returns to Algorithm 1 whether they match (true) or not (false).

If any of the `p.Match(t)` conditions in Algorithm 1 are true, then the triple and the privacy preference's URI are added to the `protectedTriplesList`. Moreover the access privileges of each matched privacy preferences are added to the `accessPrivilegesList` together with the privacy preference URI – in order to map the triples to the access privileges by using the privacy preference URI as the lookup identifier. Similarly, the access agent in each matched privacy preference are added to the `accessAgentsList` together with the privacy preference URI. Once all the triples are iterated, the filtering part filters the protected triples as explained below.

3.2 Privacy Preferences Filtering

Algorithm 3 filters the triples to send back only the triples which the agent has access to. The algorithm checks that for each triple in the `protectedTriplesList`, the agent has been granted access by matching the privacy preference URI bound to the triple with the URI bound to the agent. If these match, then the triple is added to the `accessTriplesList`. If the privacy preference URI does not match to any of the URIs bound to the agent, then the triple is added to the `noAccessTriplesList`. Once completed, the filtering algorithm sends back the `accessTriplesList` that represents the filtered result set.

4 Extended Access Control Filtering Algorithm (PPF-2)

PPF-1 has a major performance bottleneck in the privacy preference matching phase: for each restricted triple and for every privacy preference PPF-1 executes a query on the RDF store to test whether the subject or object is of a particular class type. For instance if there are 100 requested triples and 100 privacy preferences that test different types of classes, then PPF-1 will initiate 10,000 queries – assuming that each privacy preference tests only one class type. This may result in a large overhead since executing a query can be expensive – specifically on mobile devices with restricted resources. To increase efficiency, the number of necessary store accesses for identifying the class of a resource must be reduced without losing PPF-1's fine-grained control over data access.

In this section we introduce an extended filtering algorithm (called PPF-2) that fulfils these requirements. The main idea of PPF-2 is to identify the class of a resource by analysing both the requested query and the ontologies used by the data. To reduce the effort of analysing the used ontologies, we perform an ahead-of-time indexing phase for the ontologies at the system start time. This index is later used to identify the given classes. With this ahead-of-time indexing in place, the actual filtering process becomes a two stage algorithm, as follows:

1. analysis of the query to derive the resources' classes (Stage 1);
2. filtering of the triples (Stage 2), using the knowledge derived in Stage 1.

In the following we describe how we realise Step 1. Stage 2 is similar to the filtering done in PPF-1 and thus not explained again.

4.1 Knowledge Extraction from the Ontology and Query

Our solution is based on a query analysis step that allows to identify the classes of each resource based on the attributes that are used in the query. The query analyser parses the SPARQL query and for each resource it extracts *inbound* and *outbound* properties. Inbound properties are extracted from the triples in which the resource is the object. Outbound properties are extracted from the triples in which the resource is the subject. Based on these properties it is possible to identify the classes of a resource by looking at the ontologies data. Our approach

uses a closed-world assumption, i.e. we assume that the filtering algorithm knows every ontology on which a privacy preference can be defined. This assumption is valid because: if an ontology is unknown when the privacy preference is defined, then the PPM can retrieve it before any actual query is run. The RDF Schema [9] standard defines two type of relationship for properties: `rdfs:domain` and `rdfs:range`. The first is used to state that any resource that has a given property is an instance of a class, while the second is used to state that the values of a property are instances of a class. Thus, both of them can be used to derive the actual class(es) of a resource.

4.2 Defining an Index to derive Classes from Properties

As mentioned before, it is possible to identify the class of a resource by looking at the query and leveraging the ontology. Similarly to accessing the store, querying the ontologies is a slow process. This can be improved by indexing the ontologies (once) *before* any actual query is run. Thus, it is possible to make the identification of a resource's class a memory-only operation.

```
PREFIX rdf: http://www.w3.
    org/2000/01/rdf-schema
SELECT ?class ?property
WHERE {
{?property <rdf:#domain>
    ?class}
UNION {
  ?property <rdf:#domain>
    ?parent .
  ?class <rdf:#subClassOf>+
      ?parent}};
```

Fig. 2. The SPARQL 1.1 query to build the index on the domain relationship.

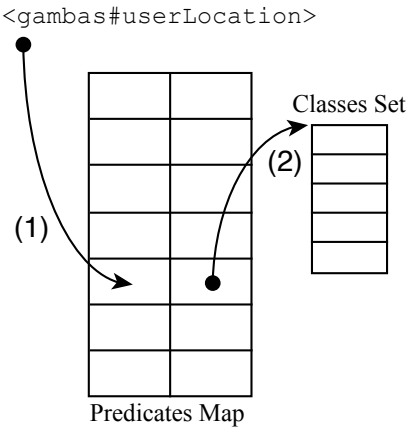

Fig. 3. The index data structure used by the class derivation algorithm. The map is accessed with the predicate (1) and then the set is processed (2).

Figure 2 shows a query that – when executed on a RDF store containing all the ontologies – extracts all the given properties of a specific class. Moreover, it uses the "new" path syntax introduced in SPARQL 1.1 to gather all the properties of its super classes. A similar query is then used to extrapolate the classes from the `rdfs:range` relationship. With this information two indexes

[9] RDF Schema – `http://www.w3.org/TR/rdf-schema/`

```
PREFIX gambas: http://www.gambas-ict.eu/ont/
PREFIX wgs84: http://www.w3.org/2003/01/geo/wgs84_pos#
SELECT ?lat ?long ?noise
WHERE {?user <gambas:userLocation> ?location .
  ?location <wgs84:lat> ?lat .
  ?location <wgs84:long> ?long .
  ?location <gambas:noiseLevel> ?noise}
```

Fig. 4. A SPARQL query example where the resources' class can be uniquely determined by the query analysis step.

are built, one for using the `rdfs:domain` and one for using the `rdfs:range` relationships. To guarantee fast access to the information in an index, we use a combination of Red-Black tree-based map and set implementations.

Figure 3 shows an example of how the `rdfs:domain` index is used. Given a resource linked through a predicate `userLocation`; we use the predicate as a key into the predicates map. The accompanying value in the map points to a set of classes, which we add to a result set. This procedure is then repeated for all predicates of the given resource. Then, all the resulting sets are intersected The resulting intersected set contains all the classes that the resource can be an instance of. This process is repeated for each index and the results are intersected.

Example Figure 4 shows a SPARQL query usable to extract the location (given as latitude and longitude) of a given user and the noise level at this location. The `?user` is modelled as a `gambas:User`, a subclass of `foaf:Agent`. The `?location` is a `gambas:Place`, a subclass of `dol:Location` [10], which has an attached `wgs84:lat` (latitude) and `wgs84:long` (longitude). In order to derive the classes of the variables in the query of Figure 4, the algorithm proceeds as follows for the `?user` resource:

1. extract the `<gambas:userLocation>` property;
2. access the index on `rdfs:domain` using the property as key;
3. access the linked classes set, which contains only the `gambas:User` class.

A similar approach can be applied to the `?location` resource. In the following section we will show a comparison of the performances of this modification versus the base case.

5 Evaluation

In order to evaluate the performance gain achieved by our extended filtering algorithm, we conducted a number of experiments on a Google Nexus 7 device

[10] DOLCE – `http://ontologydesignpatterns.org/wiki/Ontology:DOLCE\%2BDnS\_Ultralite`

running Android 4.2.2. Our system is implemented in Java. We compared two configurations with a PPM running on top of an RDF On the Go data store [8]. In the first configuration the PPM is using our previous filtering algorithm PPF-1. In the second one, the PPM is using our new filtering algorithm PPF-2.

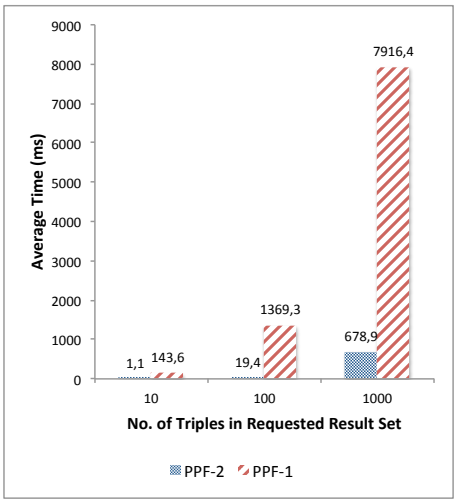

Fig. 5. Performance with varying size of result set

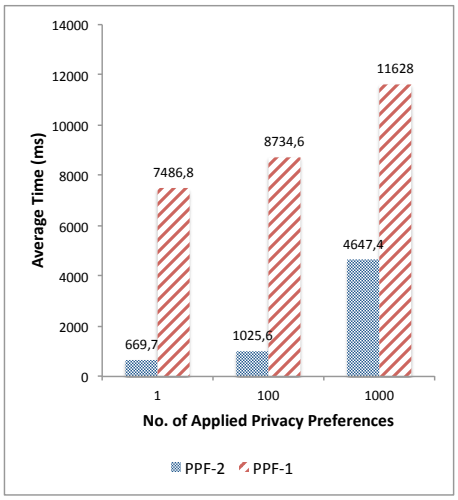

Fig. 6. Performance with varying number of privacy preferences

The evaluation dataset was composed of 15000 triples, containing data about seven real-world user profiles. Using this dataset we executed a sample query on a user's topic interests and filtered the intermediate results with both algorithms (PPF-1 and PPF-2). Since we are mainly interested in the overhead induced by access control instead of query execution, we measured the execution time for filtering, omitting the time needed to execute the sample query on the dataset. The latter time depends only on the underlying RDF store and thus is the same for both filtering algorithms. To characterise the filtering performance in scenarios with different complexity, we varied both the number of triples in the intermediate result and the number of checked privacy preferences. Each experiment was repeated ten times. We started measuring after an initial preheating phase consisting of ten filtering runs. This reduced the variance introduced by the Android Just-in-Time optimiser. Moreover, each experiment was executed independently in a separate Android App, with no other running App and with all synchronisation services disabled – further reducing variances.

Figure 5 shows the execution time for filtering an intermediate result set of varying size (10, 100, and 1000 triples) using a single privacy preference. As can be seen, PPF-2 clearly outperforms PPF-1 by at least a factor of 10, confirming the effectiveness of the predefined index technique (see Section 4). Even for an intermediate result set of 1000 triples (representing the result of a

query matching a comparatively large number of the 15000 triples in the RDF store), PPF-2 requires only approximately 0.7s to check access and filter the result set. In comparison, PPF-1 requires nearly 8s, making it unsuitable for many scenarios, e.g. interactive systems. The time required for filtering a mid size intermediate result set of 100 triples is around 0.02s for PPF-2 (compared to approximately 1.4s for PPF-1). Filtering a small intermediate result set of only 10 triples is nearly not measurable with both algorithms.

Figure 6 shows the execution time for filtering an intermediate result set of fixed size (1000 triples) using a varying number of privacy preferences (1, 100, and 1000 preferences). Again, PPF-2 clearly outperforms PPF-1 for all measurement points, reducing the absolute time for filtering triples with 100 privacy preferences to around 1s, down from 8.7s. Interestingly, the results for filtering with one privacy preference are quite similar (0.7s for PPF-2, down from 7.5s) due to fixed (i.e. size-independent) execution efforts. For 1000 privacy preferences, PPF-2 can still outperform PPF-1 by a factor of approximately 2.5 but both algorithms may still be too slow to be used in time critical scenarios (with PPF-1 requiring around 11.6s and PPF-2 around 4.6s).

Note that the presented results are only valid for situations in which the original query contains knowledge that can be used for filtering optimisation. This may not always be the case. Therefore we also conducted experiments with an unbound query that requested all triples in the RDF store. This query contains no knowledge for PPF-2. In this case PPF-2 is reduced to PPF-1. It must access the store for each triple check and thus cannot perform better than PPF-1. This is confirmed by our measurements, since the results for PPF-1 and PPF-2 are the same in this case.

6 Related Work

Access control and privacy for RDF data is not a new topic. In this section we discuss related approaches and explain how our work differs from earlier work.

Access control privileges for RDF data can be modelled using the Web Access Control (WAC) vocabulary[11]. However, this vocabulary is designed to specify access control to entire RDF documents rather than to specific data contained within the RDF document. *Privacy policies* can be modelled using the Platform for Privacy Preferences (P3P)[12]. It specifies a protocol that enables Web sites to share their privacy policies with Web users expressed in XML. P3P does not ensure that Web sites act according to their publicised policies and it does not enable end users to define their own privacy preferences. The authors in [7] propose a *privacy preference formal model* consisting of relationships between subjects and objects in Social Semantic Web applications. However, the proposed formal model does not provide fine-grain access control for RDF data. Similarly, the authors in [10] also propose an access control model for semantic networks. However, they do not cater for RDF data in mobile devices. `RelBac`

[11] WAC — `http://www.w3.org/ns/auth/acl`
[12] P3P — `http://www.w3.org/TR/P3P/`

[6] is a relational access control model that provides a formal model based on relationships amongst communities and resources. It is also not intended for RDF data stored in mobile devices.

The authors in [3] propose an *access control framework* for Social Networks by specifying privacy rules using the Semantic Web Rule Language (SWRL)[13]. However, this work does not support processing SWRL rules on mobile devices and requires a specific parser to process the SWRL syntax.

The authors in [5] compare 12 rule-based languages for enforcing access control. Most of them require defining a large amount of rules for defining access control policies. Moreover, these require specific reasoners and parsers; apart from a system to enforce them. Our system however is based on an RDF(S) vocabulary thus processable by RDF parsers without installing a specific parser. It is also light-weight and requires minimum amount of defining access control policies but keeping similar expressivity as rule-based approaches.

The authors in [9] present a role-based *access control model for RDF stores* called RAP that binds role permissions to RDF store actions, such as inserting a triple. This model does not support fine-grained access control for data stored in mobile devices. The authors in [1] also present an access control framework for RDF stores that consists of a pre-policy evaluation and query rewriting. The authors use Protune [2] for expressing the policies which requires a specific framework to process these policies.

Finally, the authors in [4] propose an access control vocabulary that is similar to our PPO and a manager similar to our PPM. However, their model applies only to named graphs, unlike our model which we apply to statements, resources and classes. Although they provide support for mobile devices, the access control policies are sent to a central server and processed on this server. Our approach supports access control filtering directly on mobile devices.

7 Conclusion and Future Work

Access to personal data on mobile devices must be controlled tightly and efficiently. In this paper we presented our approach for fine-grained access control for RDF data on mobile devices. It allows users to fully control access to their data directly on their mobile devices, increasing their trust in the system. This will increase their willingness to share such data with others in a privacy preserving manner and independently of any external provider. As we have shown, Linked Data technology like RDF and SPARQL can be used – even on mobile devices – to realise access control for RDF data. By using RDF to model our privacy preferences (with the same expressivity as rule-based approaches) and a SPARQL engine to check them, no special rule language and reasoner components are necessary. Instead, the store managing the user data can be used to realise the access control on this data. Our experiments show that to be efficient such a system should combine multiple techniques, e.g. pre-indexing, query analysis as well as result filtering. Our work can be extended in several directions.

[13] SWRL — `http://www.w3.org/Submission/SWRL/`

Firstly, an evaluation of the impact of the proposed index on a combination of different types of privacy preferences is needed. Secondly, access space queries remain problematic, as they need to be tested on the store. It should be possible to address this in a similar manner as PPF-2 by analysing and building indexes for access space queries prior to executing the filtering algorithm.

References

1. F. Abel, J. L. De Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *ISWC'07/ASWC'07*, 2007.
2. P. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *POLICY*, 2005.
3. B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. A Semantic Web Based Framework for Social Network Access Control. SACMAT'09, 2009.
4. L. Costabello, S. Villata, F. Gandon, et al. Context-aware access control for rdf graph stores. In *ECAI*, 2012.
5. O. D. De Coi J.L. A review of trust management, security and privacy policy languages. In *International Conference on Security and Cryptography*, SECRYPT'08, 2008.
6. F. Giunchiglia, R. Zhang, and B. Crispo. Ontology Driven Community Access Control. *Trust and Privacy on the Social and Semantic Web, SPOT'09*, 2009.
7. P. Kärger and W. Siberski. Guarding a Walled Garden Semantic Privacy Preferences for the Social Web. *The Semantic Web: Research and Applications*, 2010.
8. D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth. RDF On the Go: An RDF Storage and Query Processor for Mobile Devices. In *Posters and Demos of the ISWC 2010*, 2010.
9. P. Reddivari. Policy based access control for a rdf store. In *In Proceedings of the Policy Management for the Web Workshop, A WWW 2005 Workshop*, 2005.
10. T. Ryutov, T. Kichkaylo, and R. Neches. Access control policies for semantic networks. In *POLICY*, 2009.
11. O. Sacco and J. G. Breslin. PPO & PPM 2.0: Extending the privacy preference framework to provide finer-grained access control for the web of data. In *I-SEMANTICS '12*, 2012.
12. O. Sacco and A. Passant. A Privacy Preference Manager for the Social Semantic Web. In *SPIM Workshop*, 2011.
13. O. Sacco and A. Passant. A Privacy Preference Ontology (PPO) for Linked Data. In *Linked Data on the Web Workshop*, LDOW'11, 2011.
14. O. Sacco, A. Passant, and J. G. Breslin. User controlled privacy for filtering the web of data with a user-friendly manager. In *Poster Session at I-SEMANTICS '12*, 2012.
15. O. Sacco, A. Passant, and S. Decker. An Access Control Framework for the Web of Data. In *IEEE TrustCom-11*, 2011.
16. H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF + SSL : RESTful Authentication for the Social Web. *Semantic Web Conference*, 2009.