J Data Semant (2013) 2:21–36
DOI 10.1007/s13740-012-0014-9

ORIGINAL ARTICLE

# A Social Semantic Web Access Control Model

**Serena Villata · Luca Costabello · Nicolas Delaforge · Fabien Gandon**

Received: 31 December 2011 / Revised: 10 October 2012 / Accepted: 16 October 2012 / Published online: 13 November 2012


**Abstract** In the Social Web, the users are invited to publish a lot of personal information. These data can be easily retrieved, and sometimes reused, without providing the users with fine-grained access control mechanisms able to restrict the access to their profiles, and data. In this paper, we present an access control model for the Social Semantic Web. Our model is grounded on the Social Semantic SPARQL Security for Access Control vocabulary (`S4AC`). This vocabulary can be used by the users to define their own terms of access to the data. We define an algorithm, implemented in our Access Control Manager, which allows to check, after a client query, to which extent the data are available, depending on the user's profile. The evaluation of the access conditions is related to different features, such as the *social tags* associated with the data, and the user's contextual information, such as being part of a group, being located in a specific place. We provide an evaluation of the overhead introduced by our Access Control Manager, and we show that access control in the Social Semantic Web comes with a cost, but this is acceptable given the benefits of data protection.

**Keywords** Access control · Social Semantic Networks · RDF · SPARQL

## 1 Introduction

One of the key features of the Social Web is the ability to publish, and thus find a lot of personal and professional information about people. With the advent of the Social Semantic Web this is even more evident, as underlined by Breslin et al. [2]. The availability of personal and non-personal data of the users has both positive and negative sides. On the one hand, this allows people to share their data, e.g., photos, videos, posts, with their friends and the persons they know. On the other hand, semantic forms of the user's profiles and data can be reused elsewhere, e.g., what happened with FOAF search engines and aggregators as Plink, or FoaFSpace. This leads to the need for mechanisms where users can restrict the access to their data by specifying the attributes the accessors must satisfy to have the access granted. Access control is important to lead to a diffusion of Social Semantic Web platforms to make them able to guarantee the same kind of authorizations as in standard Social Web platforms such as Facebook, or Google+.

In this paper, we address the following research question:

- How to define an access control framework for the Social Semantic Web?

This research question breaks down into the following subquestions:

1. How to define a fine-grained context-aware access control model for the Social Semantic Web?
2. How to define a pluggable access control framework for the Social Semantic Web using Semantic Web languages only?

We answer these questions adopting exclusively Semantic Web languages and recycling, when possible, already existing vocabularies. The research question has to deal with different aspects that need to be taken into account when designing an access control model for the Social Semantic Web. First of all, we avoid the usual access control lists, often maintained by a sole authority, because we cannot specify the

S. Villata (✉)· L. Costabello · N. Delaforge · F. Gandon
INRIA Sophia Antipolis, Sophia Antipolis Cedex, France
e-mail: serena.villata@inria.fr

access restrictions to any particular user, in a context where the user information is so dynamic. Second, we rely on the social tags assigned by the users to the data and to the other users. Moreover, contextual information are also considered in this model to grant the access to the users by considering not only their personal information but also additional attributes, e.g., time and location constraints.

First, we describe the Social Semantic SPARQL Security for Access Control vocabulary (S4AC[1]), a lightweight ontology which allows to specify fine-grained access control policies for RDF data [27,28]. In particular, the S4AC vocabulary defines an access policy as a tuple composed by an *access privilege*, stating the kind of privilege the data consumer is granted to, an *evaluation context* which is requested to hold when the client query is performed, the *object*(s) to be protected by the policy, i.e., the named graphs [5], and a set of *access conditions* which have to be verified in order to grant the access to the consumer. This vocabulary relies on the SPARQL 1.1 language. In particular, the Access Conditions are expressed through `ASK` queries, returning `true` if access is granted to the data consumer, `false` otherwise. Access privileges are mapped to SPARQL primitives through the SPIN vocabulary.

Second, we show the integration of our access control model in the ISICIL Social Platform. The overall access control framework allows data providers to specify lightweight access policies to protect their data, at *named graph* granularity. We assume the data providers to perform a preliminary step to partition the dataset they want to protect in named graphs. We implemented a prototype of an Access Control Manager whose task is to restrict the consumption of data accessible via the associated SPARQL endpoint. The Access Control Manager verifies which named graphs are accessible by the data consumer, so that the consumer's query is run on those graphs only. Finally, the model supports a user friendly interface allowing both expert, and non-expert users to define their own terms of access. The system evaluation shows that access control comes with a cost, and that performance loss is acceptable when dealing with sensitive data.

This paper does not deal with access control for the Social Web in general, but we present a framework suitable for the Social Semantic Web. Our aim is not to provide a privacy manager or a cryptography system, but we are interested in formalizing, developing and evaluating an access control framework which authorize or not the access of the users to the data of the other users, without considering personal information only. For the time being, our lightweight framework assumes the trustworthiness of the information sent by the data consumer. Moreover, our approach focuses only on SPARQL endpoints. Other access strategies are out of the scope of this work.

[1] http://ns.inria.fr/s4ac/.

Despite the amount of proposals of access control models [1,4,13–17,19,20,25,26], none of them presents a Social Semantic access control model based on Semantic Web languages only, a pluggable and easy-to-integrate filter for generic SPARQL endpoints without modifying the endpoint itself, providing access conditions from triple granularity level up to dataset granularity level, and taking into account the social tags assigned by the users to their data and other users and the contextual information. Moreover, we rely on W3C recommendations only, as we do not introduce any new language or technology.

The reminder of the paper is organized as follows: Sect. 2 introduces the ISICIL social platform, Sect. 3 provides a description of the S4AC vocabulary and the kind of policies which can be defined using this vocabulary, Sect. 4 presents the access control model and describes the developed prototype, and Sect. 5 shows the experimental results. Related work and conclusions end the paper.

## 2 The ISICIL Platform

We introduce the ISICIL Social platform first to detail the needs of a Social Semantic Web platform, and second to show the system where our framework has been embedded and the user interface is specified. The ISICIL[2] initiative (Information Semantic Integration through Communities of Intelligence onLine) mixes web applications with formal representations and processes to integrate them into corporate practices for technological and scientific monitoring. In particular, ISICIL deals with corporate data, and the users interact with each others through a semantic social network. The platform has to manage the users' personal data, and this leads to the need of introducing an access control framework able to protect not only the personal data of the users but also the corporate data which require fine-grained access policies. This free open source platform proposes three functionalities:

1. a semantic social network bundled with social network analysis tools,
2. a system for semantic enrichment of folksonomies linked with corporate terminologies and
3. semantically augmented user interfaces, activity monitoring and reporting tools.

ISICIL integrates a social network and Semantic Network Analysis (SNA) tools. It uses popular ontologies to model a social graph in RDF (linking people, resources and tags), it designs an ontology for describing SNA indices, strategic positions and actors, communities, etc. and enriches the social graph with results from the analysis. The authors

[2] http://isicil.inria.fr.

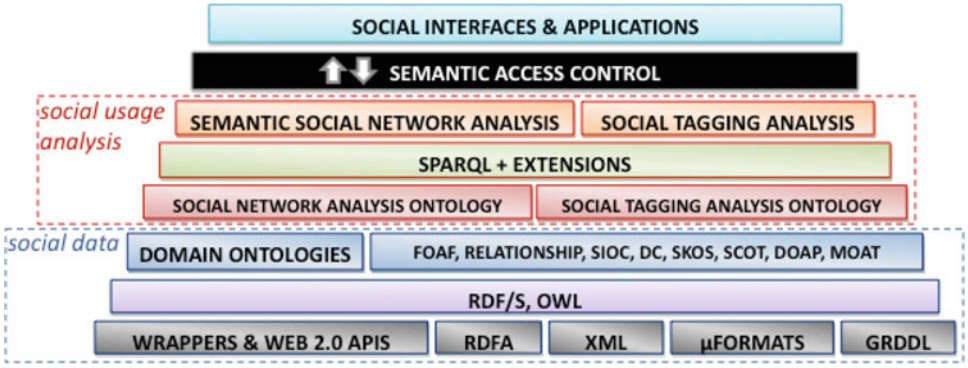


**Fig. 1** The Semantic Social Network analysis stack with the addition of the access control brick

implement the computation of the main SNA indices using SPARQL and post-processing in a few cases. See [11,12] for details.

Figure 1 visualizes the stack of tools designed to conduct a semantic social network analysis. The goal of this stack is to provide a framework that enables to consider not only the network structure embedded in the social data but also the schemas that are used to structure, link and exchange this information. The stack is composed of (1) tools for building, representing and exchanging social data, and (2) tools for extracting social network analysis metrics and leveraging social graphs with their characteristics. ISICIL represents the social graphs in RDF, which provides a directed typed graph structure. Then it leverages the typing of nodes and edges with the primitives of existing vocabularies together with specific domain ontologies when needed. With this typing, semantic engines are able to perform type inferences from data schemas for automatically enriching the graph and checking its consistency. For the analysis of the network, the authors design SemSNA[3] that defines different SNA metrics ranging from the annotation of strategic positions and strategic actors (like degrees or centralities), to the description of the structure of the network, like diameter. With this ontology, we can (1) abstract social network constructs from domain ontologies to apply the tools on existing schemas by extending our primitives, and we can (2) enrich social data with the SNA metrics that are computed on the network. These annotations enable us to manage the life cycle of an analysis more efficiently, by pre-calculating relevant SNA indices and updating them incrementally when the network changes over time. Moreover, they can be used during the querying of social data for ordering and filtering the results.

The ISICIL platform integrates these approaches into a web architecture deploying interconnected social semantic tools on an intranet to support business intelligence and technology monitoring activities including watch, search, notification and reporting. The ISICIL platform is a typical REST API architecture, built as a JavaEE web application, hosted in a classical servlet container. Because the ISICIL platform relies on the Semantic Web framework, it is vital to define an access control mechanism that relies only on Web standards. For practical reasons, the project has been split into three layers:

- The core layer embeds the CORESE/KGRAM library as the main triple store, conceptual graph engine and SPARQL 1.1 compliant interpreter[4] [6,7]. SPARQL 1.1 new features are extensively used, e.g., update, named graphs and paths. This layer also implements the read-/write mechanisms between CORESE/KGRAM and the ISICIL custom persistence system.
- The business layer is dedicated to the implementation of ISICIL models, publishing services and methods required for interacting with the semantic engine. Each business object has a dedicated service for inserting/updating/deleting annotations. We have also pre-wired the main queries to simplify client-side interaction and the parsing of the results, e.g., when accessing from mobile devices. For those who need complex queries, a SPARQL endpoint service is also provided.
- The RESTful API represents the HTTP translation of the business layer. Almost all of the business services have a corresponding web service. The authentication is managed using the Spring Security framework[5]. Because of the sensitive aspect, business intelligence data are protected by an Access Control Manager adopting the `S4AC` vocabulary.

In the rest of the paper, we will detail the social semantic access control mechanism we use in the ISICIL platform.

## 3 The Access Control Model

In this section, we present our access control model. The access control model is built over the notion of Named Graph [5], thus supporting fine-grained access control policies, including the triple level (enforcing permission

[3] http://ns.inria.fr/semsna/2009/06/21/voc.rdf.

[4] http://www-sop.inria.fr/edelweiss/software/corese/.

[5] http://static.springsource.org/spring-security/site/.

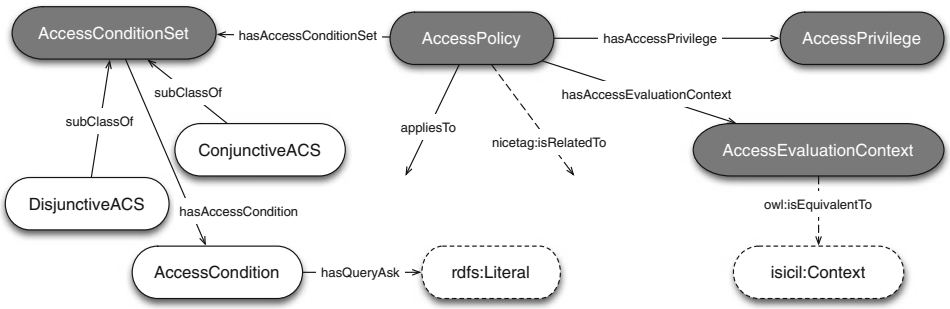


**Fig. 2** An overview of the S4AC ontology. Core classes are in *grey*

models is an envisioned use case for RDF named graphs[6]). We rely on named graphs to avoid depending on documents (one document can serialize several named graphs, one named graph can be split over several documents, and not all graphs come from documents[7]). At conceptual level, our policies can be considered as access control conditions over g-boxes[8] (according to W3C RDF graph terminology), with semantics mirrored in the SPARQL language.

The model is grounded on the Social Semantic SPARQL Security for Access Control Ontology (S4AC). An overview of S4AC lightweight vocabulary is provided in Fig. 2. Our access control model is integrated with the models adopted in the Social Semantic Web and the Web of Data. In particular, S4AC reuses concepts from SIOC[9], SKOS[10], WAC[11], NiceTag[12], SPIN[13], Dublin Core[14], and the access control model as a whole is grounded on further existing ontologies, such as FOAF[15] and RELATIONSHIP[16].

The main component of the S4AC model is the Access Policy, as presented in Definition 1. Roughly, an Access Policy defines the constraints that must be satisfied to access a given named graph or a set of named graphs. If the Access Policy is *satisfied* then the data consumer is allowed to access the data. Otherwise, the access is not granted. The constraints specified by the Access Policies may concern the data consumer, i.e., the user or the environment in which the user is querying the SPARQL endpoint, or any given combination of these dimensions.

**Definition 1** (Access Policy) An Access Policy ($P$) is a tuple of the form

$$P = \langle \text{ACS}, \text{AP}, T, R, \text{AEC} \rangle,$$

where (a) ACS is a set of Access Conditions to satisfy, (b) AP is an Access Privilege, (c) $T$ is the tag of the set of resources to be protected by $P$, (d) $R$ is the resource(s) to be protected by $P$, and (e) AEC is the Access Evaluation Context of $P$.

An Access Condition, as defined in Definition 2, expresses a constraint which needs to be verified in order to have the Access Policy satisfied. Notice that both $T$ and $R$ represent the data to protect. The difference is that $T$ refers to the set of named graphs associated with a certain tag, and $R$ refers to the URI(s) of the specific named graph(s).

**Definition 2** (Access Condition) An Access Condition (AC) is a condition which tests whether or not a query pattern has a solution.

In the S4AC model, we express an Access Condition as a SPARQL 1.1 ASK query[17].

**Definition 3** (Access Condition verification) If the query pattern has a solution (i.e., the ASK query returns *true*), then the Access Condition is said to be *verified*. If the query pattern has no solution (i.e., the ASK query returns *false*), then the Access Condition is said *not* to be *verified*.

*Example 1* An example of Access Condition, which is verified only if the data consumer is a *collaborator* of the provider of the resource to protect, is the following:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rel: <http://purl.org/vocab/relationship/>

ASK {?resource dcterms:creator ?provider.
     ?provider rel:collaboratesWith ?consumer.}
```

Each Access Policy $P$ is composed by a set of Access Conditions, as defined in Definition 4.

**Definition 4** (Access Condition Set) An Access Condition Set (ACS) is a set of access conditions of the form $\text{ACS} = \{\text{AC}_1, \text{AC}_2, \ldots, \text{AC}_n\}$.

Roughly, the verification of an Access Condition Set returns an answer of the kind *true*/*false*. We consider two standard ways to provide such an evaluation: conjunctively and disjunctively.

[6] http://bit.ly/w3rdfperm.

[7] The discussion about the use of named graphs in RDF 1.1 can be found at http://www.w3.org/TR/rdf11-concepts.

[8] http://bit.ly/graphterm.

[9] http://rdfs.org/sioc/spec/.

[10] http://www.w3.org/TR/skos-reference/.

[11] http://www.w3.org/wiki/WebAccessControl.

[12] http://ns.inria.fr/nicetag/2010/09/09/voc.html.

[13] http://spinrdf.org/.

[14] http://dublincore.org/documents/dcmi-terms/.

[15] http://xmlns.com/foaf/spec/.

[16] http://vocab.org/relationship/.

[17] http://www.w3.org/TR/sparql11-query/#ask.

**Definition 5** (Conjunctive Access Condition Set) A Conjunctive Access Condition Set (CACS) is a logical conjunction of Access Conditions of the form $\text{CACS} = \text{AC}_1 \wedge \text{AC}_2 \wedge \cdots \wedge \text{AC}_n$.

**Definition 6** (Conjunctive ACS evaluation) A CACS is verified if and only if every contained Access Condition is verified.

**Definition 7** (Disjunctive Access Condition Set) A Disjunctive Access Condition Set (DACS) is a logical disjunction of Access Conditions of the form $\text{DACS} = \text{AC}_1 \vee \text{AC}_2 \vee \cdots \vee \text{AC}_n$.

**Definition 8** (Disjunctive ACS evaluation) A DACS is verified if and only if at least one of the contained Access Conditions is verified.

The second component of the Access Policy is the Access Privilege. The privilege specifies the kind of operation the data consumer is allowed to perform on the resource protected by the Access Policy.

**Definition 9** (Access Privilege) An Access Privilege (AP) is a set of allowed operations on the protected resources of the form $\text{AP} = \{Create, Read, Update, Delete\}$.

We model the Access Privileges as four classes of operations in order to maintain a close relationship with CRUD-oriented access control systems. This relationship allows a finer-grained access control than simple read/write privileges as in WAC, and it suggests to the data providers how to specify the access privileges, following the example of CRUD-oriented systems, as we will discuss in relation to the user interface. The idea is that, in particular in the Social Semantic Web, there is a big difference in allowing the users who ask to access my data to update my data or to delete my data, or to simply read my data. We distinguish the Update, Create and Delete operations to let the provider specify with a deeper degree of detail what the consumers are allowed to perform on her data. Moreover, we relate the four privilege classes to the SPARQL 1.1 query and update language. This matching is realized with the `skos:related` property through the SPIN ontology. The latter models the primitives of the SPARQL query and update languages (e.g., `SELECT`, `INSERT DATA`, etc.) as SPIN classes. We show how this matching is actually used by our framework in Sect. 4.

As previously explained, policies protect data at named graph level. We offer two different ways of specifying the protected object: the provider may target one or more given named graphs, or it may target a set of named graphs with a common tag. The former is achieved by providing the URI(s) of the named graph(s) to protect using the `s4ac:appliesTo` property. The latter is accomplished by listing the tags of the named graphs to protect with the property `nicetag:isRelatedTo`. In this case, the assumption is that the named graphs have been annotated with such metadata.

Finally, the Access Policy is associated with an Access Evaluation Context. The latter provides an explicit link between the policy and the actual user data (modeled with FOAF in the case of ISICIL coupled with a number of existing contextual vocabularies concerning, e.g., time, location that will be used to evaluate the Access Policy). For instance, the data provider can define an access policy of the kind "The access to *named-graph*1 is granted to all the users that are friends of mine and that are currently located in Germany". In this way, the evaluation of the policy takes into account not only the profile of the consumer but also some contextual information, in this case consumer's location.

**Definition 10** (Access Evaluation Context) An Access Evaluation Context (AEC) is a list of predetermined bound variables of the form $\text{AEC} = (\langle \text{var}_1, \text{val}_1 \rangle, \langle \text{var}_2, \text{val}_2 \rangle, \ldots, \langle \text{var}_n, \text{val}_n \rangle)$.

In this paper, we focus on the ISICIL context, thus the Access Evaluation Context list is composed only by a couple $\text{AEC} = (\langle ctx, URI_{ctx} \rangle)$. We map therefore the variable *ctx*, used in the policy's Access Conditions, to the URI identifying the actual user context in which the SPARQL query has been performed. More specifically, the Access Evaluation Context is implemented as a SPARQL 1.1 `BINDINGS` Clause[18] to constrain the `ASK` evaluation, i.e., "`BINDINGS ?ctx {`$(URI_{ctx})$`}`".

Further details of the `S4AC` model include, among others: the specification of the creator of the policy (`sioc:hasCreator`) to keep track of this information in the Social Semantic platform, the creation date (`dcterms:created`), the specification of the variables used in the access conditions of the policies and their description in natural language adopted in the user interface of the framework to help the provider reusing others' policies, and a `skos:prefLabel` property associated with the Access Conditions to provide a sort of "explanation" to the consumer in case she cannot access the data (following the example of AIR [18]). Moreover, in the ISICIL platform, we are able to manage the fact that only a maximum number of accesses is granted, as in Giunchiglia et al. [15], by means of an Access Condition, and we can grant random access to a resource (e.g., `ASK{FILTER(rand()>0.5)}`.

The semantics of our Access Control Policies is mirrored in the semantics of the SPARQL language, in particular, concerning the `ASK` query and the `BINDINGS` clause. The result of the verification of each access condition is composed, in case of multiple conditions, conjunctively or disjunctively, and this combination is the overall result of the policy evaluation. The Access Privilege and the resource to protect are

[18] http://www.w3.org/TR/sparql11-federated-query/#update.

**Fig. 3** The Access Policy protecting `:alice_reviews`

```
:policy1 a s4ac:AccessPolicy;
      s4ac:appliesTo :alice_reviews;
      s4ac:hasAccessPrivilege [ a s4ac:Update];
      s4ac:hasAccessConditionSet :acs1.

:acs1 a s4ac:AccessConditionSet;
    s4ac:ConjunctiveAccessConditionSet;
    s4ac:hasAccessCondition :ac1,:ac2.

:ac1 a s4ac:AccessCondition;
    s4ac:hasQueryAsk
    """ASK {
              ?context a isicil:Context.
              ?context isicil:user ?u.
              ?u foaf:knows ex:alice#me.}""".

:ac2 a s4ac:AccessCondition;
    s4ac:hasQueryAsk
    """ASK {
               ?context a isicil:Context.
               ?context isicil:user ?u.
               ?u rel:hasFriend ?f.
               FILTER (!(?f=ex:ACME_boss#me))}""".
```

**Fig. 4** The content of the named graph `:alice_reviews` containing the reviews authored by Alice

```
ex:29900 a bibo:Article;
       dcterms:title "Great concert with Bob!";
       dcterms:date "2010";
       dcterms:creator example:alice#me;
       bibo:abstract "Really enjoyed Coldplay".

ex:29655 a bibo:Article;
       dcterms:title "Disappointed";
       dcterms:date "2010";
       dcterms:creator example:alice#me;
       bibo:abstract "Not up to the standards".
```

components of the policy which do not concur to its verification. All the semantics of our Access Policies relies on the semantics of the `ASK` queries combined with the contextual `BINDINGS`.

Conflicts among policies might occur if data provider adds Access Conditions with contrasting `FILTER` clauses. For instance, it is possible to define positive and negative statements such as `ASK{FILTER(?user=<http://example#bob>)}` and `ASK{FILTER(!(?user=<http://example#bob>))}`. If these two Access Conditions are applied to the same data, a logical conflict arises. This issue is handled in our framework by evaluating policies applied to a resource in a disjunctive way. This means that in the example above, if the consumer satisfies one of the two access conditions, then the access is granted to her. This is not satisfactory in many situations, thus we expect to add a mechanism, following the example of [10], to avoid the insertion of conflicting policies as a future work.

*Example 2* Consider the following scenario. Alice is attending a music festival and she uploads some content to the ISICIL social platform. She prefers to share these contents to all the people knowing her but not to those who are friends of her boss. We now present an example of Access Policy with a conjunctive Access Condition Set associated with an `Update` privilege (Fig. 3). The policy protects the named graph containing Alice's reviews of concerts (`:alice_reviews` visualized in Fig. 4) and allows the

*cond1*
```
ASK { ?resource dcterms:creator ?provider .
 ?provider rel:hasColleague ?user . }
```

*cond2*
```
ASK { ?resource dcterms:creator ?provider .
 ?provider rel:hasFriend ?user . }
```

*cond3*
```
ASK { ?resource dcterms:creator ?provider .
 ?provider dcterms:creator ?g .
 GRAPH ?g { ?user nicetag:hasCommunitySign ?tag }}
```

*cond4*
```
ASK { ?user a foaf:Person .
 FILTER(! (?user= <http://MyExample.net#bob>))}
```

*cond5*
```
ASK { FILTER(rand()>0.5) }
```

*cond6*
```
ASK { ?user a foaf:Person .
 FILTER(?user= <http://MyExample.net#bob>)}
```

*cond7*
```
ASK { ?resource dcterms:creator ?provider .
 ?provider sioc:member_of ?g .
 ?user sioc:member_of ?g . }
```

**Fig. 5** Examples of access conditions expressed through SPARQL ASK queries.

access to the named graph only if the consumer (a) knows Alice, and (b) is not a friend of Alice's boss.

Figure 5 presents some examples of `ASK` queries which may be associated with the access conditions. *Cond1* grants the access to those users who have a relationship of kind "colleagues" with the provider. *Cond2* grants the access to the friends of the provider *Cond3* is more complicated[19]. It grants the access to those users that are marked with a specified tag. To specify the tag, we use again the NiceTag ontology which allows to define the relationship among the resources and the tags for each tagging action. Negative access conditions are allowed, where we specify which user cannot access the data. This is expressed, as shown in *Cond4*, by means of the `FILTER` clause, and access is granted to every user except *bob*. *Cond5* expresses an access condition where the user can access the data only if he is a minimum lucky, e.g., one chance out of two. *Cond6* provides a positive exception where only a specific user can access the data, it is the contrary of *Cond4*. *Cond7* grants the access to users who are members of a particular group, to which the provides belongs too.

## 4 The Access Control Manager

The Access Control Manager (ACM), visualized in Fig. 6, is the core module which allows the data providers to define and check the Access Conditions.

First, the ACM provides a mean to the data provider to define its own access policies. The user accesses the ACM through the interface which deals with expert and non-expert users. Expert users are able to define their own Access Conditions directly writing the SPARQL 1.1 ASK queries. Non-expert users, instead, need to be guided by the interface during the policies definition, as we show in Fig. 7. They can use a number of templates both previously inserted by us and generated from the access policies defined by the other users, and they can edit these templates building the desired policies. The *hasDescription* property is used to explain meaning of the variables. The definition of the access policies includes, in particular, the definition of the Access Policies, such as (a) the set of Access Conditions, and the way they have to be evaluated, i.e., conjunctively or disjunctively, (b) the resources or the set of tags associated with the Access Policies, and (c) the binding to constrain the variables of the Access Conditions. After the definition of the policies, the user is allowed to see a preview of the result of the restrictions resulting from the application of the policies. In this way, she can verify whether the result is the expected one and she can decide to reformulate the policies. The user interface implemented in HTML 5 is visualized in Fig. 7.

The Access Control Manager we design is meant as a pluggable component to be inserted in front of a generic SPARQL endpoint, as shown in Fig. 6. Our access control framework is developed in the following way:

1. The data consumer queries the SPARQL endpoint to access the content, and at the same time, the ISICIL platform sends the user information coupled with the query. These data are sent as an `INSERT DATA` statement to build the named graph representing the user's data. Summarizing, the user sends two SPARQL queries to the endpoint, the first one for accessing the datastore, and the second one for providing her personal information. A caching mechanism can be introduced here to avoid sending the personal information every time a query is performed.
2. The query of the consumer is not directly processed by the SPARQL endpoint, but it is filtered by the Access Control Manager.
3. The Access Control Manager selects the policies concerning the consumer's query, and after their evaluation, it returns the set of named graphs the consumer is granted access to.
4. The query of the consumer is processed only on the accessible named graphs.
5. The result of the query is sent to the consumer.

The core of our framework is the Access Enforcement Module. The aim of this component is twofold: first, the module selects the Access Policies to assess, and second, it verifies the set of Access Conditions included in the selected policies to allow or not the access. In the following, we describe the two algorithms used to decide whether the consumer is allowed to access the data or not.

Algorithm 1 (Fig. 11) is the overall algorithm for the query execution with the access enforcement. The input of

[19] The GRAPH keyword is used to match patterns against named graphs.

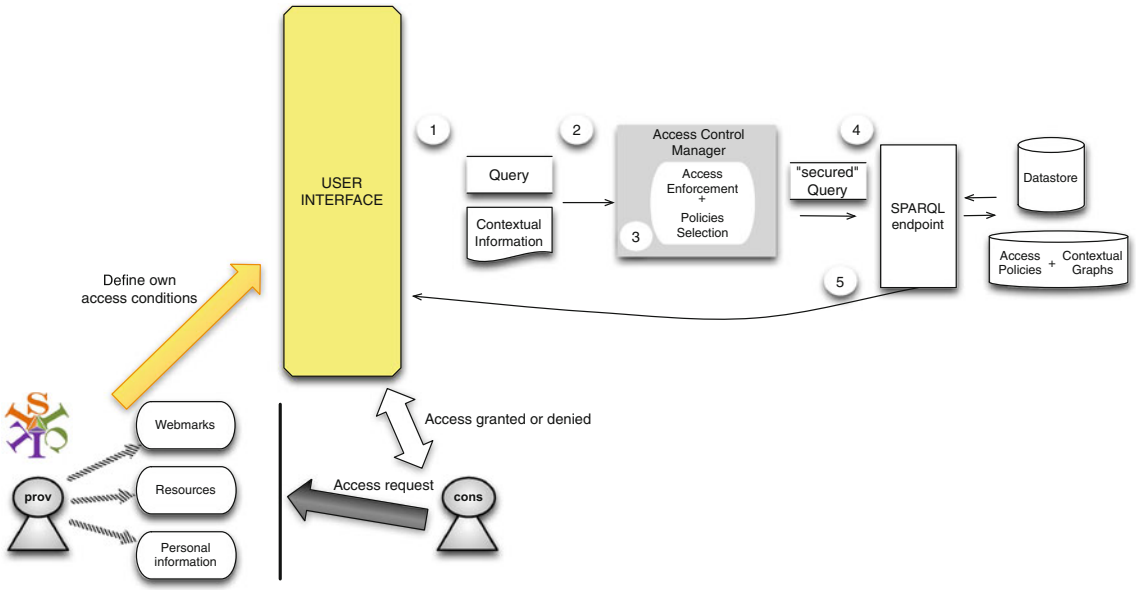


**Fig. 6** The Access Control Manager


**Fig. 7** The ISICIL non-expert user interface for creating the access policies

**Fig. 8** The SPARQL query issued by Bob

```
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX dcterms: <http://purl.org/dc/terms/>

DELETE {ex:article dcterms:subject <http://dbpedia.org/page/Category:
     Concert_tours>. }
INSERT {ex:article dcterms:subject <http://dbpedia.org/page/Category:
     Music_performance>. }
WHERE {ex:article a bibo:Article}
```

**Fig. 9** The Access Conditions bound to the actual user context with the BINDINGS clause

```
ASK{?context a isicil:Context.
       ?context isicil:user ?u.
       ?u foaf:knows ex:alice#me.}
       BINDINGS ?context {(example:actualCtx1)}

ASK {?context a isicil:Context.
       ?context isicil:user ?u.
       ?u rel:hasFriend ?f.
       FILTER (!(?f=ex:ACME_boss#me))}
       BINDINGS ?context {(example:actualCtx1)}
```

**Fig. 10** The constrained SPARQL query issued by Bob.

```
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX dcterms: <http://purl.org/dc/terms/>

DELETE {ex:article dcterms:subject <http://dbpedia.org/page/Category:
     Concert_tours>. }
INSERT {ex:article dcterms:subject <http://dbpedia.org/page/Category:
     Music_performance>. }
USING :peter_reviews
USING NAMED :peter_reviews
WHERE {ex:article a bibo:Article}
```

the algorithm is the consumer's query $Q$ and the RDF graph $G_{ctx}$ modeling the information of the consumer. We assume the existence of a repository of access policies APS. The algorithm starts by saving the consumer's graph in a local cache (line 1). The set of selected accessible named graph NGS is empty at the beginning of the algorithm execution (line 3). The selection of the Access Policies is addressed by the sub-routine Access Policies Selection (line 4), which returns the set of Access Policies the query is concerned by. Then, the algorithm runs all the Access Conditions composing the selected policies (lines 7–10). For each policy, depending on the kind of Access Conditions Set, i.e., conjunctive or disjunctive, if the policy is verified then the named graph to which the policy allows the access is added to the set of accessible named graphs (lines 11–12). Finally, after the execution of all the policies, the query of the consumer is sent to the protected SPARQL endpoint with the addition of the FROM and FROM NAMED clauses (line 16). This allows the enforcement module to execute the query only on those named graphs which are accessible, given the user information. Adding the FROM clause is not enough because, in case the client query includes a GRAPH clause, we need to specify the set of named graphs to be queried in a FROM NAMED clause, otherwise the query will be executed on all the named graphs of the store. The algorithm outputs the triples resulting from $Q$ (line 18).

*Example 3* An example of client query is shown in Fig. 8, where Bob wants to update the rock festival's reviews[20]. When the query is received by the Access Control Manager, it selects the Access Policies concerning this query (Fig. 3). The Access Conditions composing the policy are then coupled with a BINDINGS clause, as shown in Fig. 9, where the *?context* variable is bound to the actual Bob's information. Suppose Bob knows Alice, but it is also a friend of Alice's boss (note that these information are retrieved from Bob's FOAF profile): the Access Policy protecting Alice's named graph does not grant access to Bob. After the identification of the named graph(s) accessible by Bob (for instance, the named graph :peter_reviews), the Access Control Manager adds the USING and USING NAMED clauses[21] to constrain the execution of the client query only on the allowed named graphs. USING and USING NAMED describe a dataset in the same way as FROM and FROM NAMED clauses. The keyword USING instead of FROM in update requests has been chosen to avoid possible ambiguities which could arise from writing "DELETE FROM". The "secured" client query is shown in Fig. 10.

Algorithm 2 (Fig. 11) is the Access Policies Selection routine. The aim of this algorithm is to select, starting from the consumer's query, what are the Access Policies the query is concerned with. The input of the algorithm is the query $Q$ and the repository of the policies APS. The idea is that we do not want to verify all the Access Policies every time a query is run. Thus, we adopt a selection mechanism to obtain only a subset of Access Policies to execute. In particular, the algorithm maps the consumer's query to one of the four access privileges S4AC defines (line 1). Then, the algorithm selects all the Access Policies which have the identified Access Privilege (lines 3–7). The selected policies are returned to the main algorithm of access enforcement.

[20] Notice that the client query can be every kind of query defined by the SPARQL 1.1 Query and Update language, e.g., CONSTRUCT, SELECT.

[21] http://www.w3.org/TR/sparql11-update/#deleteInsert.

**Fig. 11** SPARQL query execution procedure

**Algorithm 1**: Query Execution with Access Enforcement

**Input**: a SPARQL query $Q$, an RDF graph $G_{ctx}$, Access Policy Set APS

**Output**: the SPARQL query result $R$

save $G_{ctx}$ in local contextual cache;
**if** $G_{ctx}$ *has changed* **then**
  $NGS = \emptyset$;
  $APS \leftarrow$ `APSelection`$(Q, APS)$;
  **forall** $AP_i \in APS$ **do**
    $ACcount_{false} = 0$;
    **forall** $AC_j \in ACS_i$ **do**
      append $G_{ctx}$ to $AC_j$ as `BINDINGS` clause;
      **if** $ASK_{AC_j}$ *execution returns false* **then**
        $ACcount_{false}++$;
    **if** ($ACS_{AP_i}$ *is DACS and* $ACcount_{false} < |ACS_{AP_i}|$)||($ACS_{AP_i}$ *is CACS and* $ACcount_{false} = 0$ **then**
      $NGS \leftarrow NGS \cup NG_{AP_i}$;
**else**
  $NGS \leftarrow NGS_{cached}$;
**forall** $NG_i \in NGS$ **do**
  append `FROM <`$NG_i$`>`, `FROM NAMED<`$NG_i$`>` to $Q$;
$R \leftarrow$ run $Q$;
**return** $R$;

**Algorithm 2**: Access Policies Selection

**Input**: SPARQL client query $Q$, APS

**Output**: a reduced set of Access Policies $APS_r$

$AccPrv_Q \leftarrow$ map $Q$ type to CRUD operation;
$APS_r = \emptyset$;
**forall** $AP_i \in APS$ **do**
  **if** $AccPrv_{AP_i} \equiv AccPrv_Q$ **then**
    $APS_r \leftarrow APS_r \cup AP_i$ ;
  **end**
**end**
**return** $APS_r$;

## 5 Evaluation

To assess the impact on response time, we implemented the Access Control Manager as a Java EE component and we plugged it to the Corese-KGRAM RDF store[22] and SPARQL 1.1 query engine[23] [7]. We evaluate the prototype on an Intel Xeon E5540, Quad Core 2.53 GHz machine with 48 GB of memory, using the Berlin SPARQL Benchmark (BSBM) dataset 3.1[24].

In Fig. 12, we show the execution of ten independent runs of a test query batch consisting in 50 identical queries of a simple `SELECT` over `bsbm:Review` instances (tests are preceded by a warmup run). We measure the response time with and without access control (referred to as secured

[22] Concerning accessing inferred statements, Corese-KGRAM allows to know where are the inferred triples. In this way, we can apply to these inferred triples the same access policies that regulate the access to the triples from which these triples have been inferred.

[23] http://www-sop.inria.fr/edelweiss/software/corese/.

[24] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/Dataset/.

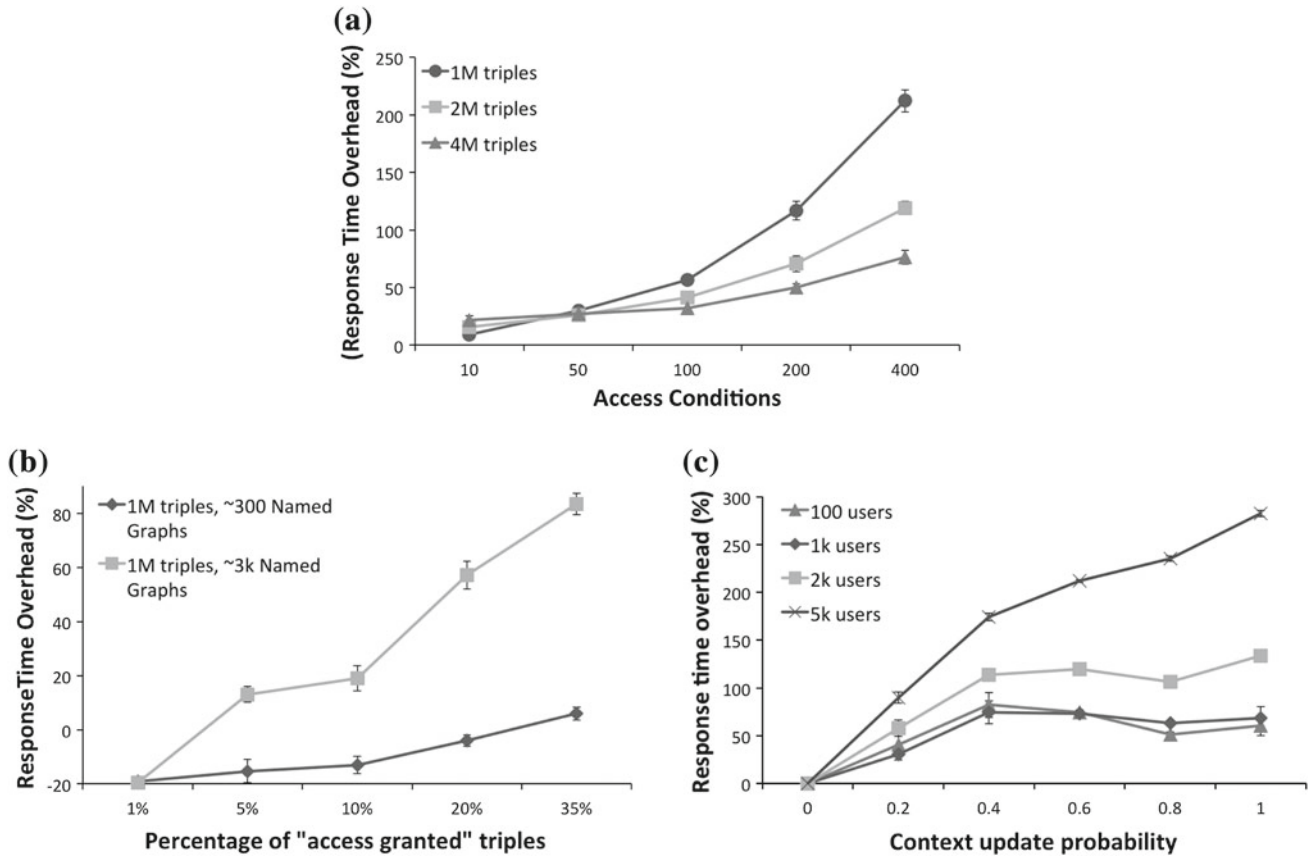


**Fig. 12** Response time overhead

and non-secured in the figures). When executed against the Access Control Manager, the test SPARQL query is associated with the static user context. Each Access Policy contains exactly one Access Condition. In Fig. 12a, to simulate a worst-case scenario, access is granted to all named graphs defined in the base (i.e., all Access Conditions return true), so that query execution does not benefit from cardinality reduction. Figure 13 shows the same results of Fig. 12a but with the actual values for (a) 1M triples, (b) 2M triples, and (c) 4M triples, respectively. The results show that larger datasets are less affected by the delay introduced by our prototype, as datastore size plays a predominant role in query execution time (e.g., for 4M triples and 100 always-true Access Policies we obtain a 32.6 % response time delay).

In a typical scenario, the Access Control Manager restricts the results of a query. In Fig. 12b, we assess the impact on performance for various levels of cardinality reduction, using modified versions of the BSBM dataset featuring a larger amount of named graphs (we define a higher number of `bsbm:RatingSites`, thus obtaining more named graphs). Figure 14 shows the same results of Fig. 12b but with the actual values. The results show that, when access is granted to a small fraction of named graphs, the query is executed faster than the case without access control (e.g., if access is granted to only 1 % of named graphs, the query is executed 19 % faster on the 1M triple test dataset). As more named graphs and triples are accessible, performance decreases. In particular, response time is affected by the construction of the active graph, determined by the merge of graphs in the `FROM` clauses. As shown in Fig. 12b, the cost of this operation grows with the number of named graphs returned by the evaluation of the Access Policies.

In Fig. 12c, we analyze the overhead introduced on response time by queries executed in dynamic user environments. Figure 15 shows the same results of Fig. 12c but with the actual values. We execute independent runs of 100 identical `SELECT` queries, dealing with a range of context change probabilities. In case of a context update, the query is coupled with a SPARQL 1.1 `DELETE/INSERT` (i.e., update) of the context graph. Not surprisingly, with higher chances of updating the context, the response time of the query grows, since more SPARQL queries need to be executed. The delay of `INSERT DATA` or `DELETE/INSERT` operations depends on the size of the triple store and on the number of named graphs (e.g., after a `DELETE` query, the adopted triple store refreshes internal structures to satisfy RDFS entailment). Performance is therefore affected by the

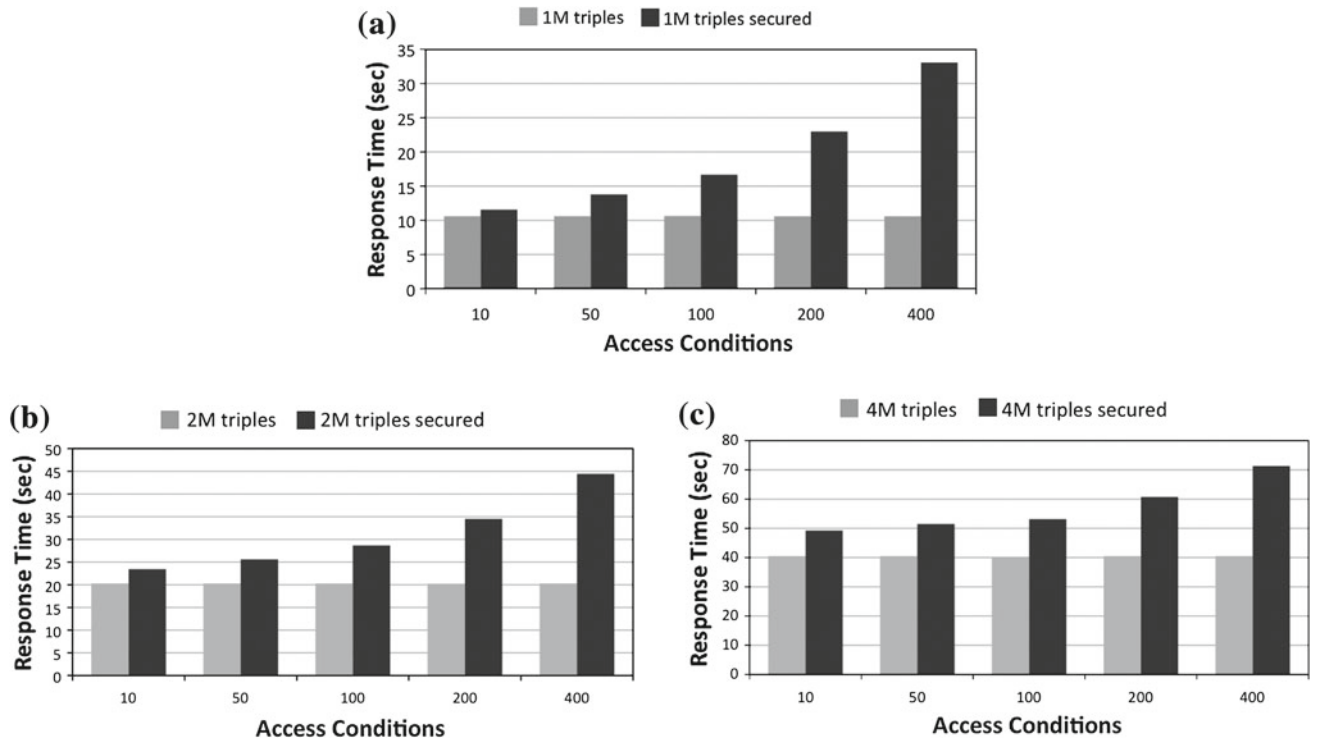


**Fig. 13** Response time on the number of access conditions (with and without access control)

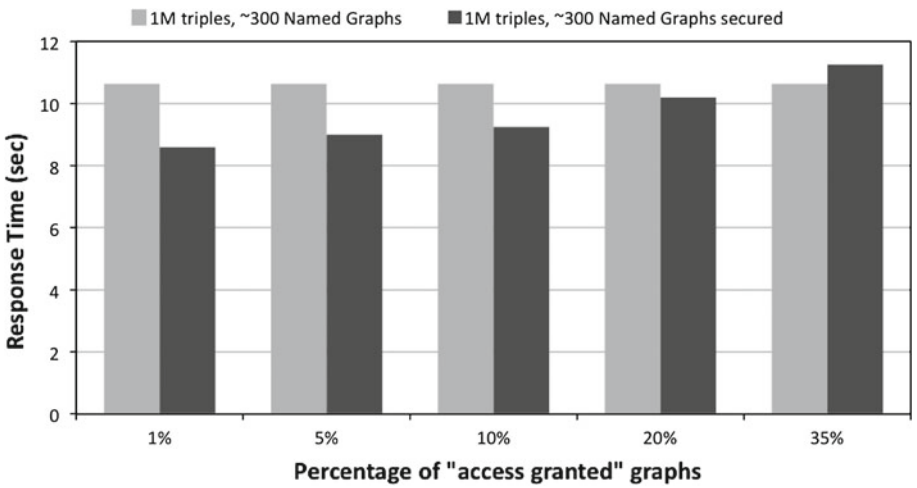


**Fig. 14** Response time on the percentage of access granted graphs (with and without access control)

number of active consumers, since each of them is associated with a user context graph.

## 6 Related Work

In this section, we summarize and compare the existing literature with the proposed approach. In particular, the comparison is addressed following the different features that are italicized in Table 1. In this table, we use N/A when the feature is not available in the model, YES when the feature is implemented in that model, and when necessary we specify the feature. The last row of the table is dedicated to the model we discuss in this paper (for brevity we call it *S4AC*). The table considers the following key features of the access control models: adopted languages (the kind of language used to express the policies), CRUD (the kind of access privileges the model is able to manage), context-awareness (if the context dimension is used to grant or not the access), granularity (what is protected by the model), performance evaluation (if an experimental evaluation of the model is presented), role-based (if the model is role-based), conflicts verification (if the model is able to detect possible inconsistencies among the policies). Following this table, in the rest of this section,

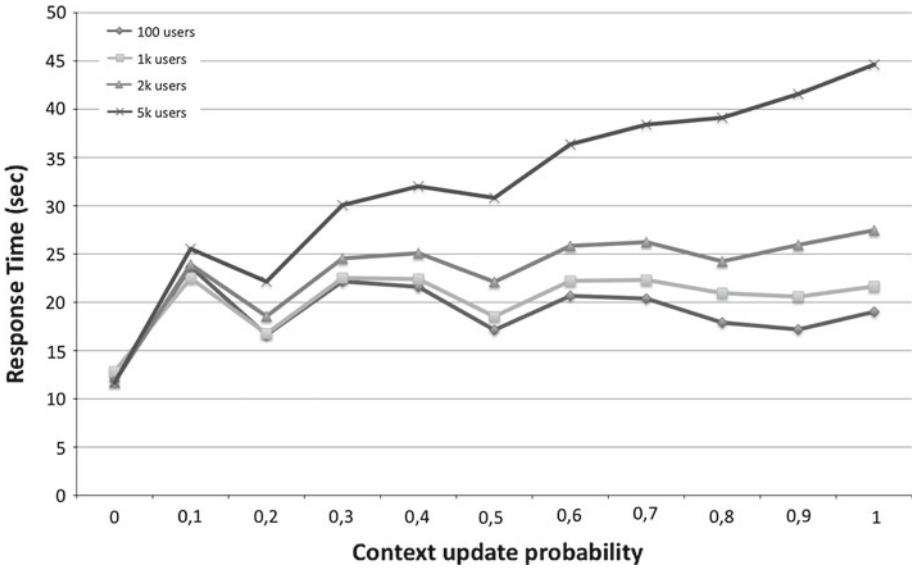


**Fig. 15** Response time with various context update probabilities (IM triples, 100 Access)

we compare the literature considering two different families of approaches, i.e., context-aware access control models, and context-unaware access control models.

### 6.1 Context-Unaware Access Control Models

The definition of access control policies for the Web has been firstly addressed by the Web Access Control vocabulary (WAC), which allows data providers to specify access control lists (ACL). This vocabulary grants access to a whole RDF document. In this paper, we provide a fine-grained access control model which grants access to specific RDF data, i.e., the data provider may want to restrict the access to a few named graphs. Moreover, we let consumers submit any SPARQL query, and the user information takes part into the evaluation.

Sacco and Passant [20,21] present a Privacy Preference Ontology (PPO[25]), built on top of WAC, to express fine-grained access control policies to an RDF file. In their approach, the consumer asks to access a particular RDF file, e.g., a FOAF profile. Their access control manager selects the part of the file the consumer can access, and returns it to the consumer. They do not propose an access control filter for generic SPARQL endpoints, mapping the queries with the access policies. They also specify access queries with SPARQL `ASK`. They rely entirely on the WAC vocabulary without distinguishing different kinds of `Write` actions, and they cannot specify conjunctive and disjunctive sets of privacy preferences.

Muhleisen et al. [19] present a policy-enabled server for Linked Data called PeLDS, where the access policies are expressed using a descriptive language called PsSF, based on SWRL[26]. They distinguish only `Read` and `Update` actions. The system is based on an ontology of the actions that can be performed on the datasets, but no further description is provided.

Giunchiglia et al. [15] propose a Relation-Based Access Control model (*RelBAC*), a formal model of permissions based on description logic. They require to specify who can access the data, while in our framework and in [21] the provider specifies the attributes the consumer must satisfy.

Finin et al. [13] study the relationship between OWL and Role-Based Access Control (RBAC) [22]. They briefly discuss possible ways of going beyond RBAC, and, in particular, they consider "attribute-based access control", where, similar to the proposal of this paper, the access constraints are based on general attributes of an action.

Hollenbach et al. [16] present a system where the providers can control the accesses to RDF documents, and these controls are expressed using the WAC. Our model extends WAC for allowing the construction of more fine-grained access control policies.

The Access Management Ontology (AMO) [3] defines a role-based access control model. The AMO ontology consists of a set of classes and properties dedicated to the annotation of the resources, and a base of inference rules modeling the access strategy to carry out. This model again needs to specify who can access the data.

Carminati et al. [4] propose a fine-grained on-line social network access control model based on semantic web technologies. Their main idea is to encode social network-related information by means of an ontology. By constructing such

[25] http://vocab.deri.ie/ppo.

[26] http://www.w3.org/Submission/SWRL/.

**Table 1** A qualitative evaluation of the proposed model with respect to the related work

| | Adopted languages | CRUD | Context awareness | Granularity | Performance evaluation | Role-based | Conflicts verification |
|---|---|---|---|---|---|---|---|
| WAC | RDF | Read/write | N/A | RDF document | N/A | N/A | N/A |
| Sacco and Passant | RDF/SPARQL | Read/write | N/A | Part of PDF document | N/A | N/A | N/A |
| Muhleisen et al. | SWRL | Read/write | N/A | RDF document | YES | YES | N/A |
| Giunchiglia et al. | DL | N/A | N/A | Resources | N/A | N/A | N/A |
| Finin et al. | OWL/RDF | N/A | N/A | Resources | N/A | YES | N/A |
| Hollenbach et al. | RDF | Read/write/control | N/A | RDF document | YES | N/A | N/A |
| Abel et al. | High level syntax | Read | YES | Triples | YES | N/A | N/A |
| Cuppens and Cuppens-Boulahia | Datalog | Read/write | YES but temporal and spatial only | Resources | N/A | N/A | YES |
| Corradi et al. | RDF | N/A | YES | Resources | YES | N/A | YES |
| Toninelli et al. | DL | N/A | YES | Resources | YES | N/A | YES |
| Flouris et al. | High level syntax | Read | N/A | Triples | YES | YES | YES |
| Carminati et al. | SWRL | Read | N/A | Resources | YES | N/A | N/A |
| Stroka et al. | RDF | N/A | N/A | Resources | N/A | N/A | N/A |
| **S4AC** | **RDF/SPARQL** | **Create/read/update/delete** | **YES** | **Named graphs** | **YES** | **N/A** | **N/A** |

an ontology, the authors model the Social Network Knowledge Base. They assume that a centralized reference monitor hosted by the social network manager will enforce the required policies. The access control policies are encoded as SWRL[27] rules. This approach is also based on the specification of who can access the resources, i.e., the access request is a triple $(u, p, URI)$, where the user $u$ requests to execute privilege $p$ on the resource located at *URI*.

Stroka et al. [23] present a preliminary proposal about securing the collaborative content on the platform KiWi. They consider global permissions, individual content item permissions, and RDF type-based permission management. They do not specify the kind of access polices they can define.

Flouris et al. [14] present a fine-grained access control framework on top of RDF repositories. As in our approach, the authors underline the need of a fine-grained access control framework while being repository independent. Differently from our framework, they propose a high-level specification language which has to be translated into a SPARQL/SerQL/SQL query to enforce the policy, while we use directly SPARQL 1.1 to specify the policies. Moreover, they focus only on read operations.

### 6.2 Context-Aware Access Control Models

Abel et al. [1] present a model of context-dependent access control at triple level, where contextual predicates are also allowed, e.g., related to time, location and credentials. The policies are not expressed using Semantic Web languages, but they introduce a high-level syntax then mapped to existing policy languages. They enforce access control as a layer on top of RDF stores. They pre-evaluate the contextual conditions, then the queries are expanded, and sent to the database.

Hulsebosch et al. [17] propose a number of context-sensitive verification methods with the aim to check the authenticity of the user's information. Their work shows that context-sensitive access control augments the usual access control models based only on the user's attributes by making them adaptable to contextual changes.

Cuppens and Cuppens-Boulahia [9] propose an Organization-Based Access Control (OrBAC) model where contextual conditions are also expressed. Contextual conditions are considered as extra conditions to be satisfied to activate a security rule. Their conditions are based on Datalog rules, and a context algebra is introduced. The main difference is that we entirely rely on Semantic Web languages. Moreover, we consider additional contexts to the temporal and spatial ones.

Corradi et al. [8] present UbiCOSM, a security middleware adopting the context as a basic concept for policy specification and enforcement. As we do, the authors consider

[27] http://www.w3.org/Submission/SWRL/.

context as a first-class design principle to rule the access to the resources. They distinguish among physical (i.e., physical spaces) and logical contexts (e.g., temporal conditions, user activities). The policies are expressed at a high level of abstraction in terms of RDF meta-data.

Toninelli et al. [25,26] follow two design guidelines, which guide also the construction of the framework proposed in this paper: context-awareness to control resource access, and semantic technologies for context and policy specification. What differs from our approach is, on one hand, the application scenario, spontaneous coalitions in their case and the Social Semantic Web in our approach, and, on the other hand, the semantic technology adopted, i.e., a rule-based approach with description logic in their case and the SPARQL 1.1 language in our case. Finally, their model is not seen as a pluggable framework on top of generic SPARQL endpoints. Toninelli et al. [24], instead, present a quality of context aware approach to access control in the Proteus policy framework. In this paper, we do not take into account the problem of the quality of the contextual information, and this is left for future work.

## 7 Conclusions

In this paper, we have introduced a fine-grained access control model for the Social Semantic Web. This model is grounded on the `S4AC` vocabulary which allows the users of social networks to define Access Conditions for their data. In particular, these Access Conditions are implemented as SPARQL 1.1 `ASK` queries, and they can be either conjunctively or disjunctively evaluated. Moreover, Access Policies can be constrained w.r.t. the set of tags the resources are tagged with. The Access Evaluation Context provides the mapping, implemented as a `BINDINGS` clause, between the information about the consumer and the Access Conditions. We have presented our Access Control Manager, realized in the context of the ISICIL social platform. The manager grants or denies access to the users. Through an interface which allows also non-experts to interact with the system, users can specify the Access Policies to protect their data. The manager looks for the policies which apply to the resource, and after checking the contextual constraints, if present, and the features of the consumer, it states whether the access is granted or not.

There are several lines to follow for future work. First of all, in this paper, we assume that the user's information is trustworthy. The trustworthiness of the information sent by consumers should not be taken for granted. The `User`'s identity needs to be certified: this is an open research area in the Web, and initiatives such as WebID[28] specifically deal with this issue. Hulsebosch et al. [17] provide a survey of context verification techniques (e.g., heuristics relying on context history, collaborative authenticity checks). At the same time, also the other contextual information like the time of the query or the location of the consumer when she is querying the SPARQL endpoint have to be checked. We plan to follow the example of Toninelli et al. [24].

Second, a user evaluation campaign is needed to evaluate our access control framework. The campaign aims at establishing the understandability of the framework, the definition of the access policies from non-expert users, the explanation of the result after the attempt to access the data, and many other features of our framework. Third, as ubiquitous connectivity grows, access control in the Social Semantic Web must not ignore the mobile context in which data consumption takes place. We are currently working on a mobile access control framework which will introduce also the *device* dimension to our user's information.

**Acknowledgments** The first author acknowledges the support of the DataLift Project ANR-10-CORD-09

[28] http://www.w3.org/2005/Incubator/webid/spec/.

## References

1. Abel F, Coi JLD, Henze N, Koesling AW, Krause D, Olmedilla D (2007) Enabling advanced and context-dependent access control in rdf stores. In: Proceedings of the 6th international semantic web conference (ISWC-2007), LNCS vol 4825, pp 1–14
2. Breslin J, Passant A, Decker S (2009) The social semantic web. Springer, Heidelberg
3. Buffa M, Faron-Zucker C, Kolomoyskaya A (2010) Gestion sémantique des droits d'accès au contenu : l'ontologie AMO. In: Yahia SB, Petit JM (eds) EGC, Revue des Nouvelles Technologies de l'Information, vol RNTI-E-19. Cépaduès-Éditions, pp 471–482
4. Carminati B, Ferrari E, Heatherly R, Kantarcioglu M, Thuraisingham BM (2011) Semantic web-based social network access control. Comput Secur 30(2–3):108–115
5. Carroll JJ, Bizer C, Hayes PJ, Stickler P (2005) Named graphs. J Web Sem 3(4):247–267
6. Corby O, Dieng-Kuntz R, Faron-Zucker C (2004) Querying the semantic web with Corese search engine. In: Proceedings of the 16th European conference on artificial intelligence (ECAI-2004). IOS Press, pp 705–709
7. Corby O, Faron-Zucker C (2010) The KGRAM abstract machine for knowledge graph querying. In: Web Intelligence, IEEE, pp 338–341
8. Corradi A, Montanari R, Tibaldi D (2004) Context-based access control management in ubiquitous environments. In: Proceedings of the 3rd IEEE international symposium on network computing and applications (NCA-2004), pp 253–260
9. Cuppens F, Cuppens-Boulahia N (2008) Modeling contextual security policies. Int J Inf Sec 7(4):285–305
10. Cuppens F, Cuppens-Boulahia N, Ghorbel MB (2007) High level conflict management strategies in advanced access control models. Electr Notes Theor Comput Sci 186:3–26
11. Erétéo G, Buffa M, Gandon F (2011) Semtagp: semantic community detection in folksonomies. In: IEEE/WIC/ACM international conference on Web Intelligence
12. Erétéo G, Buffa M, Gandon F, Corby O (2009) Analysis of a real online social network using semantic web frameworks. In:

Proceedings of the 8th international semantic web conference (ISWC-2009), LNCS, vol 5823. Springer, Berlin, pp 180–195
13. Finin TW, Joshi A, Kagal L, Niu J, Sandhu RS, Winsborough WH, Thuraisingham BM (2008) ROWLBAC: representing role based access control in OWL. In: Ray I, Li N (eds) SACMAT. ACM, pp 73–82
14. Flouris G, Fundulaki I, Michou M, Antoniou G (2010) Controlling access to RDF graphs. In: Proceedings of the 3rd future internet symposium (FIS-2010), LNCS, vol 6369, pp 107–117
15. Giunchiglia F, Zhang R, Crispo B (2009) Ontology driven community access control. In: Proceedings of the 1st workshop on trust and privacy on the social and, semantic web (SPOT-2009)
16. Hollenbach J, Presbrey J, Berners-Lee T (2009) Using RDF metadata to enable access control on the social semantic web. In: Proceedings of the workshop on collaborative construction, management and linking of structured knowledge (CK-2009)
17. Hulsebosch RJ, Salden AH, Bargh MS, Ebben PWG, Reitsma J (2005) Context sensitive access control. In: Proceedings of the 10th ACM symposium on access control models and technologies (SACMAT-2005), pp 111–119
18. Khandelwal A, Bao J, Kagal L, Jacobi I, Ding L, Hendler JA (2010) Analyzing the air language: a semantic web (production) rule language. In: Proceedings of the 4th international conference on web reasoning and rule systems (RR-2010), LNCS, vol 6333. Springer, Berlin, pp 58–72
19. Muhleisen H, Kost M, Freytag JC (2010) SWRL-based access policies for linked data. In: Proceedings of the 2nd workshop on trust and privacy on the social and, semantic web (SPOT2010)
20. Sacco O, Passant A (2011) A privacy preference manager for the social semantic web. In: Proceedings of the 2nd workshop on semantic personalized information management: retrieval and recommendation (SPIM-2011)
21. Sacco O, Passant A (2011) A privacy preference ontology (PPO) for linked data. In: Proceedings of the 4th workshop about linked data on the web (LDOW-2011)
22. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. IEEE Comput 29(2):38–47
23. Stroka S, Schaffert S, Burger T (2010) Access control in the social semantic web—extending the idea of FOAF+SSL in KiWi. In: Proceedings of the 2nd workshop on trust and privacy on the social and, semantic web (SPOT2010)
24. Toninelli A, Corradi A, Montanari R (2009) A quality of context-aware approach to access control in pervasive environments. In: Proceedings of the 2nd international conference on mobile wireless middleware, operating systems, and applications (MOBILWARE-2009), LNICST, vol 7. Springer, Berlin, pp 236–251
25. Toninelli A, Montanari R, Kagal L, Lassila O (2006) A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: Proceedings of the 5th international semantic web conference (ISWC-2006), LNCS, vol 4273. Springer, Berlin, pp 473–486
26. Toninelli A, Montanari R, Kagal L, Lassila O (2007) Proteus: a semantic context-aware adaptive policy model. In: Proceedings of the 8th IEEE international workshop on policies for distributed systems and, networks (POLICY-2007), pp 129–140
27. Villata S, Delaforge N, Gandon F, Gyrard A (2011) An access control model for linked data. In: Proceedings of the 7th international IFIP workshop on semantic web & web semantics (SWWS-2011), LNCS, vol 7046. Springer, Berlin, pp 454–463
28. Villata S, Delaforge N, Gandon F, Gyrard A (2011) Social semantic web access control. In: Proceedings of the 4th international workshop social data on the web (SDoW-2011), pp 48–59