A Social Semantic Web Access Control Model

Serena Villata, Nicolas Delaforge, Fabien Gandon, Amelie Gyrard

Social Semantic Web Access Control*

Serena Villata, Nicolas Delaforge, Fabien Gandon, Amelie Gyrard

INRIA Sophia Antipolis {firstname.lastname}@inria.fr

Abstract. In the Social Web, the users are invited to publish a lot of personal information. These information 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 resources. 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 Ontology. This ontology can be used by the users to define, thanks to an Access Control Manager, their own terms of access to the data. Moreover, the Access Control Manager allows to check, after a query, to which extent the data is available, depending on the user's profile. The evaluation of the access conditions is related to different features, such as *social tags*, contextual information, being part of a group, and relationships with the data provider.

1 Introduction

One of the key features of Social Web is the ability to publish, and thus find a lot of personal and professional information about people. With the advent of Social Semantic Web, this is more evident, as underlined by Breslin et al. [2]. This availability of 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 users' profiles can be reused elsewhere, e.g., what happened with FOAF search engines and aggregators as Plink, or FoaFSpace. This leads to the need of mechanisms whereby users can restrict the access to their data.

In this paper, we address the research question: *How to define an access control model for the Social Semantic Web?* This question has to deal with different aspects that need to be taken into account when designing a model of access control for the Social Web. First of all, we avoid the usual access control lists, often maintained by a sole authority, because we cannot specify the access restrictions to any particular user, in a context where the user information is so dynamic as in the Social Web. Second, we rely on the social tags assigned to the users and their data. Moreover, the contextual information are considered in this model, i.e., time constraints, geo-localization information, maximum of accesses to a resource. Finally, the model supports a user friendly interface allowing both expert, and non expert users to define their own terms of access.

* The authors acknowledge support of the projects ISICIL ANR-08-CORD-011 and DataLift ANR-10-CORD-09 funded by the French National Research Agency.

We define the Social Semantic SPARQL Security for Access Control vocabulary (S4AC), a lightweight ontology which allows the users to specify fine-grained access control policies for their RDF data (Figure 1). At the core of S4AC is the Access Condition which is a SPARQL 1.1. `ASK` clause that specifies the condition to be satisfied in order to grant the access to a resource. Moreover, the users can define Access Conditions based on *tags* which restrain the conditions to the resources tagged with such tags, e.g., resources tagged "friends", "amici", "ami". The conditions can be bound to specific values to provide an Access Evaluation Context, e.g., `<''?user'', <http://myExample.net#sery>>` where the URI of the user is bound to `<http://myExample.net#sery>`. Finally, the Access Condition is associated with a temporal and spatial validity. The Access Privilege, instead, defines which kind of privilege is granted to the user satisfying the Access Conditions and the contextual constraints, e.g., `s4ac:Read` grants the user the privilege to read the requested data. Moreover, we introduce the Access Control Manager letting the users in the Social Semantic Web to (i) define the access conditions for their RDF data, e.g., their FOAF profile, and (ii) filter the RDF data depending on the access conditions the user who wants to access satisfies.

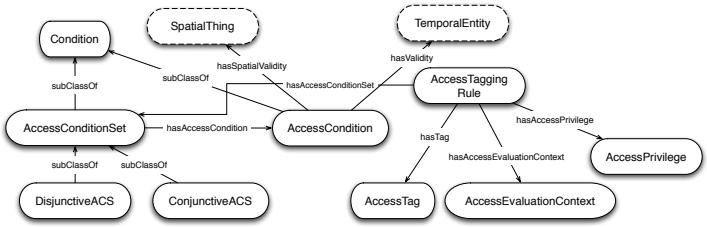

Fig. 1. An overview of the S4AC Ontology.

A key feature of our approach is to rely only on Semantic Web languages. As a consequence, our access control model is platform independent, and can be used by any kind of system based on those languages. In particular, the semantics of our policies is grounded in SPARQL 1.1[1] `ASK` queries. Relying on SPARQL semantics, our model allows the user to submit arbitrary queries while enforcing fine-grained access rules on the results he will receive. If the result of the `ASK` query is *true*, then the user is provided with the information he requires. If the result is $false$, then the model returns to the user a denial coupled with one or more rule labels explaining the reasons of the denial.

The reminder of the paper is organized as follows: Section 2 provides a description of the S4AC ontology and the kind of policies which can be defined using such ontology, and Section 3 presents the access control model and describes the developed prototype. Related work and conclusions end the paper.

[1] http://www.w3.org/TR/sparql11-query/

2 Social Semantic SPARQL Security for Access Control Ontology

The Social Semantic SPARQL Security for Access Control Ontology (S4AC), online at `http://ns.inria.fr/s4ac/v1#`, is detailed in Figure 2. One of the key features of our access control approach is to be integrated with the models adopted in the fields of the Social Web, and of the Web of Data. In particular, S4AC reuses concepts from SIOC[2], SCOT[3], NiceTag[4], WAC, TIME[5], GEO[6], and the access control model as a whole is grounded on further existing ontologies, as FOAF[7], Dublin Core[8], and RELATIONSHIPS[9].

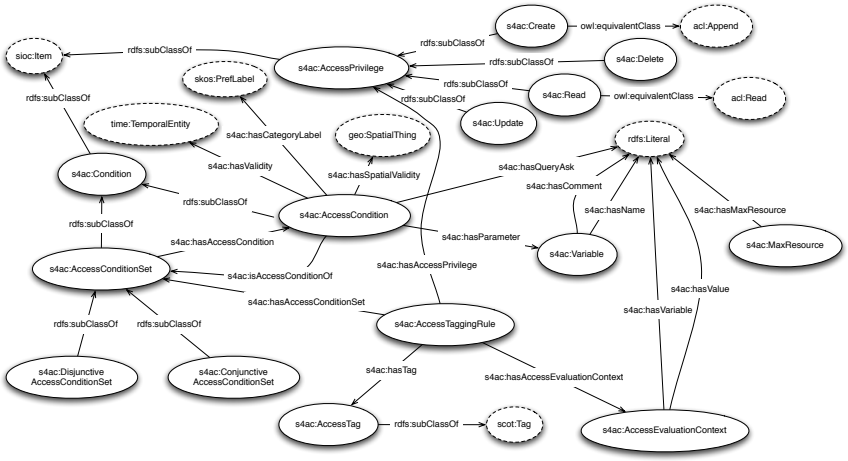

Fig. 2. The S4AC Ontology.

The main class of the S4AC ontology is the class *AccessCondition*, which is a subclass of the class *Condition*, itself a subclass of `sioc:Item`.

Definition 1. *An Access Condition (AC) is a SPARQL 1.1* `ASK` *query. If a solution exists, the* `ASK` *query returns true, and the Access Condition is said to be verified. If no solution exists the* `ASK` *query returns false, and the Access Condition is said not to be verified.*

[2] http://rdfs.org/sioc/spec/
[3] http://scot-project.net/
[4] http://ns.inria.fr/nicetag/2010/09/09/voc.html
[5] http://www.w3.org/TR/2006/WD-owl-time-20060927/
[6] http://www.w3.org/2003/01/geo/wgs84_pos
[7] http://xmlns.com/foaf/spec/
[8] http://dublincore.org/documents/dcmi-terms/
[9] http://vocab.org/relationship/.html

The *Access Condition* grants or restricts the access to the data. If the `ASK` returns *true*, the access is granted to the user. In order to return the user a more informative answer if the access is denied, we introduce the property *hasCategoryLabel*. This property allows to associate to each AC one or more natural language labels which "identify" the access condition, and they are returned to the user to provide him the reasons of the denial. We cannot return the user all the access conditions, because this would make him aware of the policies of the provider. If it is the case that only some results are filtered, it is a matter of the access control model whether to communicate or not, thanks to the *hasCategoryLabel* property, that an access restriction has been applied. The *AccessCondition* defines two properties of the access polices: *hasValidity*, and *hasSpatialValidity*. They allow to define the validity of an Access Condition. Thanks to the use of the concept `time:TemporalEntity`, the validity can be expressed in various ways: valid from/through a specific date/time, or valid in a specific time interval. *hasSpatialValidity*, instead, deals with the spatial localization of the user at the moment of trying to access the data. We use the concept `geo:SpatialThing` in order to express the spatial constraints. These properties are used to express policies in which not only the identity of the user requesting the data is checked, but also the contextual information related to the time and place in which the request is performed. A further class is *MaxResource* which defines the number of times the user can access all or a specified resource. We introduce also the property *hasParameter* which provides for each variable used in the ACs, a comment in natural language explaining the meaning of the variable. This is introduced with the aim to explain to the user how the variables are used in the access policies he is adopting, e.g., "?date" has the associated comment "the date of creation of the resource".

Definition 2. *An Access Evaluation Context (AEC) is a list L of predetermined bound variables of the form* $L = (\langle var_1, val_1\rangle, \langle var_2, val_2\rangle, \ldots, \langle var_n, val_n\rangle)$ *that is turned into a SPARQL 1.1 Binding Clause to constrain the* `ASK` *query evaluation when verifying the Access Conditions.*

The *AEC* is represented in the ontology as the class *AccessEvaluationContext* which has two properties, *hasVariable* and *hasValue*, which are respectively the variable, and the value to which the variable is bound. It is used to provide a standard evaluation context to the access conditions, e.g., requesting user, resource provider. Consider the following example: `L=(<''?resource'', ''<http://MyExample.net#doc>''>,<''?user'',''<http://MyExample.net#sery>''>)`. This list can be used to generate an additional SPARQL 1.1 Binding Clause for the access conditions of the form: `BINDINGS ?resource ?user {(<http://MyExample.net#doc>, <http://MyExample.net#sery>)}`.

Definition 3. *An Access Condition Set (ACS) is a set of Access Conditions.*

The *AccessConditionSet* class has a property *hasAccessCondition* which identifies which Access Conditions form the ACS. Two subclasses of *AccessConditionSet* are introduced: conjunctive, and disjunctive ACS.

Definition 4. *A Conjunctive Access Condition Set (CACS) is a logical conjunction of Access Conditions of the form* $CACS = AC_1 \wedge AC_2 \wedge \ldots \wedge AC_n$. *A CACS is verified if and only if every access conditions it contains is verified.*

Definition 5. *A Disjunctive Access Condition Set (DACS) is a logical disjunction of Access Conditions of the form* $DACS = AC_1 \vee AC_2 \vee \ldots \vee AC_n$. *A DACS is verified if and only if at least one of the access conditions it contains is verified.*

Definition 6. *An Access Tagging Rule (ATR) is a triple* $R = \langle ACS, TagSet, Bindings \rangle$ *where ACS is an Access Condition Set, TagSet is a set of tags* $\{tag_1, tag_2, \ldots, tag_m\}$, *and Bindings is an Access Evaluation Context. An ATR is verified for a resource tagged with one or more tags from TagSet if and only if the ACS is verified for that resource. The ACS may be reduced to a single access condition. In this case, the ATR is said to be verified if and only if the single access condition is verified. The TagSet may be empty, in which case the ATR applies to any named graph.*

An ATR declares that the access conditions in the ACS applies to any RDF graph tagged with one or more tags from *TagSet*. The class *AccessTaggingRule* has three properties: *hasAccessConditionSet*, associating an ACS to the ATR, *hasTag*, providing a set of tags to the ATR, and *hasAccessEvaluationContext*, associating to the ATR the AEC, i.e., the bindings applied to the rule. Moreover, it has the property *hasAccessPrivilege* which defines the access privilege the user is granted to: *Read*, *Create*, *Update*, *Delete*. We expand the `acl:Write` class, which is used for every kind of modification on the content, and we allow fine-grained access control privileges. The class *AccessTag*, used to define the set of tags, is a sub-class of `scot:Tag`.

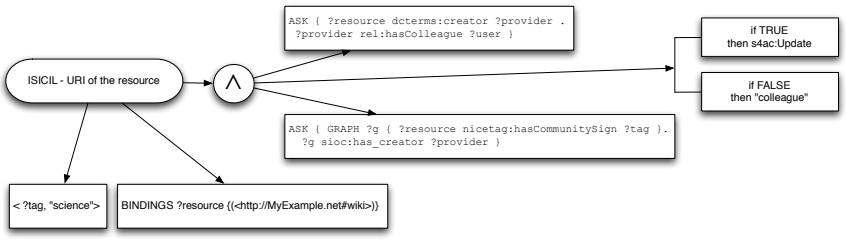

Fig. 3. An example of access policy.

We show now in detail which kind of access control policies are enabled by the proposed access control model. Consider the policy defined below: the data provider defines an access policy such that only his resources tagged with tag "family" are constrained by the access condition which grants the access to those users which have a *hasParent* relationship with the data provider, i.e.,

the parents of the provider. The Access Condition Set is composed only by one access condition, thus this is the only one which needs to be evaluated. The access privilege is `Read`. Thus, given a `SELECT` query of the user, if he is granted with the access, then he is allowed to `Read` the requested data. The use can access the data from December 31th at 23:59. If the user is not granted with the access then the label the system returns him together with the failure message is "parents", to explain that the reasons of the failure have to be associated to the fact that the user is not a parent of the provider; we choose not to send any message if some results are filtered.

```
<http://MyExample.net/expolicies>
 a s4ac:AccessTaggingRule;
  s4ac:hasAccessConditionSet [
   s4ac:hasAccessCondition [
     s4ac:hasValidity [
       time:hasBeginning [
         time:inXSDDateTime 2011-12-31T23:59:00
       ];
     ];
     s4ac:hasCategoryLabel ''parents''@en;
     s4ac:hasQueryAsk ''
       ASK { ?resource dcterms:creator ?provider .
       ?provider rel:hasParent ?user }''
   ];
  ];
  s4ac:hasAccessPrivilege s4ac:Read;
  s4ac:hasTag ''family''@en.
```

The table below presents some examples of the `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, and *Cond3* extends this access condition also to the friends of friends. *Cond4* is more complicated[10]. It grants the access to those users that are marked with a specified tag. For specifying the tag, we use the NiceTag ontology which allows to specify the relationship among the resources and the tags for each tagging action. Also negative access conditions are allowed, where we specify which specific user cannot access the data. This is expressed, as shown in *Cond5*, by means of the `FILTER` clause, and the access is granted to every user except *sery*. *Cond6* 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. *Cond7* provides a positive exception where only a specific user can access the data, it is the contrary of *Cond5*. *Cond8* grants the access to those users who are members of a particular group, to which the provides belongs too. Finally, *Cond9* ensures the access to all the resources tagged with *?tag*.

An example of conjunctive ACS is as follows: $CACS_{friends-but-sery} = Cond_2 \wedge Cond_5$, where the access is granted to the users who are friends of the provider, but the user `<http:MyExample.net#sery>`, even if she is a friend of the provider, cannot access the data. An example of disjunctive ACS is

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

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 rel:hasFriend{1,2} ?user }
cond4	ASK { ?resource dcterms:creator ?provider . ?provider dcterms:creator ?g . GRAPH ?g { ?user nicetag:hasCommunitySign ?tag }}
cond5	ASK { FILTER(! (?user= <http://MyExample.net#sery>))}
cond6	ASK { FILTER(random()>0.5) }
cond7	ASK { FILTER(?user= <http://MyExample.net#sery>)}
cond8	ASK { ?resource dcterms:creator ?provider . ?provider sioc:member_of ?g . ?user sioc:member_of ?g }
cond9	ASK { GRAPH ?g { ?resource nicetag:hasCommunitySign ?tag } ?g sioc:has_creator ?provider }

$DACS_{colleagues-or-friends} = Cond_1 \vee Cond_2$, where it is ensured that the users who are colleagues or friends of the provider are allowed to access the data.

The ATR detailed above can be constrained to a wider set of tags such as $ATR_{parents} = \langle Cond, \{''parent'','' parents'','' family'','' relatives''\}, \emptyset\rangle$ where no AEC is provided. Further examples of ATRs are: (i) $ATR_{friends} = \langle Cond_2, \{''friends'','' amici'','' ami''\}, \emptyset\rangle$ where the access condition constrains the access to friends, and three tags are provided without an AEC; (ii) $ATR_{group} = \langle Cond_7, \{''common'','' group'','' close''\}, \emptyset\rangle$ is the same for the belonging to the group of the provider; (iii) $ATR_{hiking} = \langle Cond_4, \emptyset, \langle ''?tag'','' hiking''\rangle\rangle$ where the user can access the data if he is tagged with tag "hiking" in the graph created by the provider; (iv) $ATR_{fun} = \langle DACS_{colleagues-or-friends}, \{''fun'','' funny'','' :-)''\}, \emptyset\rangle$ where the user can access the data if the disjunctive ACS above is satisfied on the named graphs tagged with these three tags.

3 Access control for the Social Semantic Web

3.1 The ISICIL use case

The challenge of the ISICIL[11] project is to reconcile new web applications with formal representations and processes to integrate them into corporate practices for technological, and scientific monitoring. More specifically, ISICIL proposes to study and to experiment with the usage of new tools for assisting corporate intelligence tasks. These tools rely on Web 2.0 advanced interfaces, e.g., blog, wiki, social bookmarking, for interactions, and on semantic web technologies for interoperability and information processing.

[11] http://isicil.inria.fr/v2/index.php

In this context, the users can create webmarks, resources, e.g., wiki pages, and personal information, e.g., social relationships represented through an activity stream. All these data cannot be fully accessible by any other user on the Web. The idea is that the users should be allowed to define their own policies in order to grant the access to their data only to those users who have the features they require. In particular, the access control model has to consider the social dimension in which it is inserted. This leads to the need of defining a model where the users can easily define their access policies, e.g., by using tags, and the relation among them. The access control model has to rely on a vocabulary like S4AC able to define the fine-grained properties the user must satisfy to access the data. For instance, the WAC vocabulary[12] allows the user to specify access control lists (ACL). The ACL are of the form `[acl:accessTo <card.rdf>; acl:mode acl:Read, acl:Write; acl:agentClass <groups/fam#group>]`, which means that anyone in the group `<http://example.net/groups/fam#group>` may read and write `card.rdf`, but a drawback of this vocabulary is that it grants the access to a whole RDF document, e.g., `card.rdf`.

3.2 The Access Control Manager

The Access Control Manager (ACM), visualized in Figure 4, is the core module which allows the user to define, and check the access conditions.

First, the ACM provides a mean to the user (userA in Figure 4) to define its own access policies. The user accesses the ACM through the user interface which considers two kinds of users: the expert users, and the non expert ones. The expert users we consider are those users who are able to define their own access conditions directly writing the SPARQL 1.1 ASK queries. Non expert users, instead, are those users who need to be guided through the interface during the policies definition, as shown in Figure 5, and they can reuse and edit the policies defined by other users clarified by using the *hasParameter* property to explain the variables. The definition of the access policies includes in particular the definition of the Access Tagging Rules, such as (i) the set of access conditions, and the way they have to be evaluated, i.e., conjunctively or disjunctively, (ii) the set of tags the resources have to be associated with in order to apply these access conditions, and (iii) the binding to constrain the variables of the access conditions. After the definition of the policies, the user is allowed to see through the interface a preview of the result of the restrictions resulting from the application of the policies. In this way, the user can verify whether the result is the expected one, or not, and he can decide eventually to reformulate the policies.

Second, consider another user, (userB in Figure 4), who wants to access the data of userA. The request to access the data is first filtered by the ACM which will allow userB to access only the data he is granted access to. The ACM receives the query of the userB. Once the request of the userB is received, the ACM selects, by means of the module called Access Control Policies Selector

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

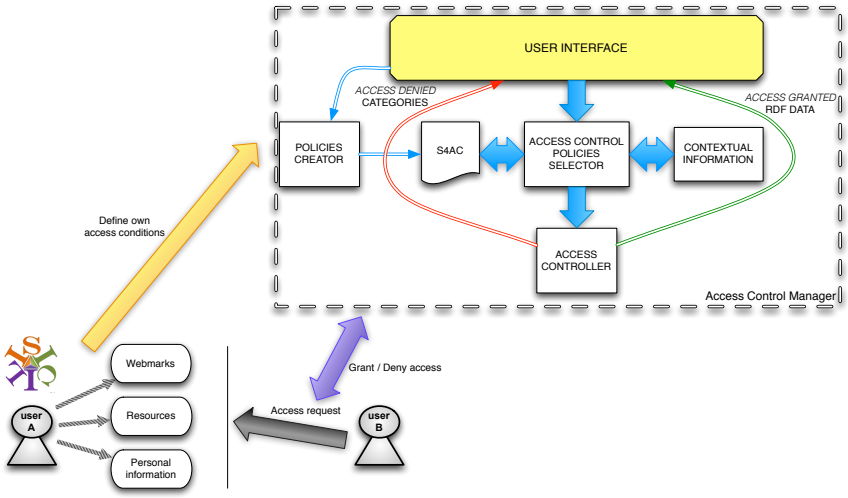

Fig. 4. The Access Control Manager.

(ACPS), which policy applies, depending on the requested operation. For instance, if the user uses a `SELECT` query, then the ACPS identifies all the policies which apply, and concern a `Read` access privilege. The ACPS performs two kinds of operations: (i) it checks the S4AC module which contains all the access conditions provided by userA to protect his data, in order to identify which access conditions apply, and (ii) it checks whether the contextual information, e.g., the temporal or spatial validity of the selected policies is satisfied. Note that we check whether the contextual constraints hold before checking the reminder of the policy. If the contextual constraints are not satisfied, then we already know that the access will not be granted. After the identification of the policies, and a positive checking of the contextual constraints, the Access Controller module matches the policies according to the userB's profile to identify what he can access. The Access Controller addresses a SPARQL `ASK` query which returns *true* if the access to the data is granted to userB. Note that userB will receive only the data he can access, and he does not know that there may be other data to which his query was addressed and that he cannot access. If the answer is *false*, then the Access Controller returns a failure, coupled with the categories causing the failure. The categories are natural language labels that are used to explain to the the user the reasons behind the failure of his query. These categories are provided to the Access Controller by the ACPS when it checks the ontology. An example of access policy composed by two access conditions that have to be conjunctively evaluated, a Bindings clause, and a Tag Set is visualized in Figure 3. The two ACs constrain the access to all the users who are colleagues of the data provider, and to all the resources tagged with ?*tag*, respectively. Moreover, the ACs are applied only to those named graphs tagged with "science",

and to the resource identified by the URI `<http://MyExample.net#wiki>`. If the conjunction is positively evaluated, then the access is granted with the privilege `s4ac:Update`. Otherwise, the access is denied, and the label "colleagues" is returned.

The developed prototype provides a user interface implemented in HTML 5, as visualized in Figure 5. It relies on the SPARQL query engine KGRAM/CORESE[13]. Briefly, the system uses the Binding SPARQL 1.1 to substitute the variable *?resource* with the URI of the resource to be accessed. The query is executed to obtain all the *ATR*s associated with the resource, and the data provider. CORESE returns these *ATR*s which contain the ACS. The `ASK` queries inside the single AC are executed on CORESE, and the returned booleans are conjunctively or disjunctively evaluated to grant or deny the access.

Fig. 5. The non-expert user interface for creating the access policies.

4 Related work

Sacco and Passant [8] present a Privacy Preference Ontology (PPO), built on top of WAC, in order to express fine-grained access control policies to an RDF file. They also specify the access queries with a SPARQL `ASK`, but their vocabulary does not consider the temporal and spatial validity of the privacy preferences, and the maximum number of accesses allowed. They rely entirely on the WAC vocabulary without distinguishing the `Write` actions. Their model does not allow to specify set of tags to limit the application of the policies to the resources

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

marked with those tags, and to specify conjunctive and disjunctive sets of privacy preferences.

Giunchiglia et al. [6] propose a Relation Based Access Control model (*RelBAC*), providing a formal model of permissions based on description logics. They require to specify who can access the data, while in our model and in [8] the provider can specify the attributes the user must satisfy.

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.

Abel et al. [1] present a model of context-dependent access control at triple level, where also contextual predicates are allowed, e.g., related to time, location, credentials. The policies are not expressed using Web languages, but they introduce an high level syntax then mapped to existing policy languages.

Hollenbach and Presbrey [7] present a system where the users can define access control on RDF documents, and these access controls are expressed using the WAC. Our model extends WAC for allowing the construction of more fine-grained access control policies.

Carminati et al. [4] propose a fine-grained on-line social network access control model based on semantic web technologies. The access control policies are encoded as SWRL[14] 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. [9] 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.

Finin at al. [5] study how to represent RBAC using the OWL language. The authors show also the representation of policies based on general attributes of an action, similarly to what we present in this paper. The difference is that we specify the policies using SPARQL 1.1 ASK queries, where the Bindings clause is used to specify the values of the variables, and temporal and spatial constraints may be expressed too.

5 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 ontology which allows the users of the social networks to define the access conditions for their data. In particular, these access conditions have the form of SPARQL 1.1 ASK queries, and they can be either conjunctively or disjunctively evaluated. Moreover, the access policies can be constrained w.r.t. the set of tags the resources are tagged

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

with, and an access evaluation context providing the bindings can be specified too. We have presented our Access Control Manager, in the context of the ISICIL platform. The manager has the aim to grant or deny the access to the users. Through a user interface which allows also non-expert users to interact with the system, the 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 and the features of the user trying to access, 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 are trustworthy. Since this assumption is not always verified, we will investigate the adoption of methodologies able to assess the trustworthiness of the users. Second, a prototype of the Manager has been developed in the ISICIL platform. We aim at providing a more efficient implementation of the Manager, in order to fully integrate it into the platform.

References

1. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *Proceedings of the 6th International Semantic Web Conference (ISWC-2007), LNCS 4825*, pages 1–14, 2007.
2. John Breslin, Alexandre Passant, and Stefan Decker. *The Social Semantic Web*. Springer-Verlag, Heidelberg, 2009.
3. Michel Buffa, Catherine Faron-Zucker, and Anna Kolomoyskaya. Gestion sémantique des droits d'accès au contenu : l'ontologie AMO. In Sadok Ben Yahia and Jean-Marc Petit, editors, *EGC*, volume RNTI-E-19 of *Revue des Nouvelles Technologies de l'Information*, pages 471–482. Cépaduès-Éditions, 2010.
4. Barbara Carminati, Elena Ferrari, Raymond Heatherly, Murat Kantarcioglu, and Bhavani M. Thuraisingham. Semantic web-based social network access control. *Computers & Security*, 30(2-3):108–115, 2011.
5. Timothy W. Finin, Anupam Joshi, Lalana Kagal, Jianwei Niu, Ravi S. Sandhu, William H. Winsborough, and Bhavani M. Thuraisingham. ROWLBAC: representing role based access control in OWL. In Indrakshi Ray and Ninghui Li, editors, *SACMAT*, pages 73–82. ACM, 2008.
6. Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Ontology driven community access control. In *Proceedings of the 1st Workshop on Trust and Privacy on the Social and Semantic Web (SPOT-2009)*, 2009.
7. James Hollenbach, Joe Presbrey, and Tim Berners-Lee. 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)*, 2009.
8. Owen Sacco and Alexandre Passant. A Privacy Preference Ontology (PPO) for Linked Data. In *Proceedings of the 4th Workshop about Linked Data on the Web (LDOW-2011)*, 2011.
9. Stephanie Stroka, Sebastian Schaffert, and Tobias Burger. 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)*, 2010.