Ontology-based access control for social network systems

Amirreza Masoumzadeh* and James Joshi

School of Information Sciences,
University of Pittsburgh,
135 North Bellefield Avenue,
Pittsburgh, PA 15260, USA
Fax: (412)-624-2788
E-mail: amirreza@sis.pitt.edu
E-mail: jjoshi@sis.pitt.edu
*Corresponding author

Abstract: As the information flowing around in social network systems is mainly related or can be attributed to their users, controlling access to such information by individual users becomes a crucial requirement. The intricate semantic relations among data objects, different users, and between data objects and users further add to the complexity of access control needs. In this paper, we propose an access control model based on semantic web technologies that takes into account the above mentioned complex relations. The proposed model enables expressing much more fine-grained access control policies on a social network knowledge base than the existing models. We demonstrate the applicability of our approach by implementing a proof-of-concept prototype of the proposed access control framework and evaluating its performance.

Keywords: social network systems; SNSs; privacy; access control; semantic web.

Reference to this paper should be made as follows: Masoumzadeh, A. and Joshi, J. (2011) 'Ontology-based access control for social network systems', *Int. J. Information Privacy, Security and Integrity*, Vol. 1, No. 1, pp.59–78.

Biographical notes: Amirreza Masoumzadeh is currently working toward his PhD at the School of Information Sciences, University of Pittsburgh. He is a member of the Laboratory of Education and Research on Security Assured Information Systems (LERSAIS). He received his Master and Bachelor in Computer Engineering (Software) from Sharif University of Technology, Iran and Ferdowsi University, Iran in 2007 and 2004, respectively. His research interests include information security, mainly access control models, privacy, and trust in modern information systems. He is a student member of IEEE and ACM.

James Joshi is an Associate Professor and the Director of the Laboratory for Education and Research on Security Assured Information Systems (LERSAIS) in the School of Information Sciences at the University of Pittsburgh. He received his MS in Computer Science and PhD in Computer Engineering from Purdue University in 1998 and 2003, respectively. His research interests include role-based access control, trust management, and secure interoperability. He is a member of IEEE and ACM.

1 Introduction

Social network(ing) systems (SNSs) are increasingly becoming a major type of online applications that facilitate online social interactions and information sharing among a large number of users. The scale of active entities, interactions, and digital content in these complex environments brings about new security and privacy challenges. Users constantly provide contents and information to these systems, either explicitly, such as by uploading a photo, or implicitly by leaving behind interaction traces, such as by responding to an invitation. Because they are related to the users, such contents may include privacy-sensitive information. Besides data protection challenges for such contents from a system perspective, protecting users privacy from other users of the system is a unique requirement in SNSs. As per a general goal of SNSs, users are motivated to expand their social connectivity and awareness through interactions and content sharing with each other. However, as the social connections of a user grows, so does the complexity of privacy implications for him. The increased variety of social connections requires more fine-grained control on privacy-sensitive information.

Current major SNSs such as Facebook and MySpace provide some privacy control settings to their users. However, the access and privacy control features provided by these systems are usually limited, and not so flexible and robust. Moreover, they seem to be implemented incrementally without detailed formal modelling, which is not appropriate for such systems with huge user base and high volume of privacy-sensitive content. Several desirable control features are missing and there exists no basis of verifying consistency in policy enforcement. For instance in Facebook, a user can choose to hide her status of relationship with a second party. But one can learn about that relationship if the second party happens to not hide it. In other words, users cannot control disclosure of some intuitively privacy-sensitive information. As an example of inconsistency in policy enforcement, even if a user chooses not to be publicly listed in Facebook she will be still listed in the public listings of the groups she has joined.

Early access control models for social networks focus on computing trust values for users based on which they make access decisions (Kruk, 2004; Kruk et al., 2006; Carminati et al., 2006; Villegas et al., 2008). However, they do not consider complexities of the protected resources in SNSs. Digital resources in SNSs are comprised of various data types. Also, different annotation methods such as tagging and commenting are common in these systems. These all introduce a variety of semantic relations among objects. In particular, it is important to ensure the protection of not only the basic data entities and values, but also their relations. For instance, a person tagged in a photo might not be only concerned about being tagged, but also about who else has been tagged in the same photo, and who actually owns the photo. In order to truly capture the fine-grained protection requirements in SNSs, it is important to have an appropriate data model. We rely on ontology modelling of knowledge using Semantic Web technologies. Some recent work also propose to use ontologies (Carminati et al., 2009; Ryutov et al., 2009), but fail to provide protection for relations, which is central to our approach. Moreover, unlike in traditional systems where security administrators are in charge of access control policy, in an SNS, users should be recognised as the main authority over access control policies regarding the information related to them. A flexible authority model is required to determine each user's authority over different resources. This feature has not been addressed in existing work.

In this paper, we propose an access control model that takes into account the intricate semantics of the privacy-sensitive knowledge base, and also respects the individual users' right to have a flexible control over access control policy on contents related to them. The access control model is designed to be as close as possible to knowledge ontology level and neutral in terms of application-level semantics in order to be suitable for generic social information systems. We leverage the semantic web technologies, i.e., OWL, SWRL, and SPARQL in particular, to model SNS knowledge and express and enforce access control policies, which can ensure practicality of our approach. Our contributions in this work can be summarised as follows.

- We propose social network systems ontology (SNO) to capture the information semantics in an SNS. We elaborate and discuss various scenarios regarding our proposed access control model based on SNO.
- We propose an ontology-based social network access control (OSNAC) model which addresses the protection of semantics-rich information in a knowledge base ontology such as SNO by using an access control ontology (ACO) and access control policy rules. It supports both user-defined authorisation rules and a system-level policy. The model also supports advanced user-level rules to provide more flexible control, including delegation of authority, dependent authorisations, and the ability to enable multiple authorities to enforce a composite policy regarding a protected resource. The model can also provide support for negative authorisation for an SNS.
- We provide an architecture and the prototype implementation of an OSNAC engine that automatically enforces access control policies on queries submitted to an SNS knowledge base. Our results show acceptable performance of our prototype and demonstrates the applicability of our approach.

The rest of the paper is organised as follows. In Section 2, we provide a brief introduction to the standards used, and propose an ontology for representing knowledge in an SNS. In Section 3, we present our proposed access control model, including the ACO, various supported policies, the enforcement model, and support for negative authorisation. We provide details about our prototype implementation and results in Section 4. In Section 5, we review related literature, and subsequently conclude the paper in Section 6.

2 Preliminaries

2.1 Semantic web languages, notations, and terminology

We leverage several semantic web languages in this work. In particular, we use web ontology language (OWL) to express the protected knowledge in an SNS and some access control decision information, semantic web rule language (SWRL) to specify access control policy rules, and SPARQL protocol and RDF query language (SPARQL) for access control enforcement.

OWL (W3C, 2004a) is a W3C recommendation to express meanings and semantics, which builds on RDF/RDFS. There are three main concepts in OWL. A class is a collection of objects, which are also called individuals/instances of the class. A *property* is a directed binary relation (predicate). An *object property* relates instances of two classes, and a *datatype property* relates individuals to data values (e.g., string values). A

class or a property can be defined as subclass or subproperty of another. OWL also supports various operators on classes such as union, intersection, and complement and restrictions such as cardinality constraints on properties. Depending on support for different operators/concepts, OWL has three sublanguages. For the purpose of this work, we have chosen OWL DL as it provides semantic features adequate for expressing knowledge in an SNS, does not have intractability issues, and there exist various tools and packages that support it. As customary to XML-based languages, we use namespace prefixes to distinguish different ontologies. For instance, owl:Class represents the type *Class* in the OWL namespace. We use a function-like notation for representing OWL property instances. For example, rdf:type(Person, owl:Class) defines Person as a new OWL class using property rdf:type in the context of RDF ontology. Instances of a property are called triples in the context of RDF. In a triple such as owns(Alice, book1), owns is the property, Alice is the subject of the property, and book1 is the *object* of the property. We use this terminology for describing reification of ontology properties.

SWRL (W3C, 2004b) allows combining horn-like rules with an OWL knowledge base, thereby enabling new knowledge reasoning tools. We encode access control policy rules using SWRL to reason on top of access decision information stored in an OWL-based knowledge base and infer access decision. SWRL rules have a very detailed syntax (W3C, 2004b). For the purpose of our work, we represent them simply as *antecedent* $\Rightarrow$ *consequent*, where *antecedent* (body) is a conjunction of multiple predicates and *consequent* (head) is a single predicate. Predicates can be either unary or binary, representing either a class or a property, respectively. A notation such as ?x is used to declare variable x in the body/head of a rule, which can be bound to class instances. For example, the following rule expresses that if someone is a tenure-track faculty he/she has a PhD degree.

$$\mathsf{TenureTrackFaculty}(?x) \Rightarrow \mathsf{hasDegree}(?x, \text{"PhD"})$$

SPARQL (W3C, 2008) is a syntactically SQL-like language for querying RDF graphs via pattern matching. We augment SPARQL queries with access control predicates to automatically enforce access control policies when a query is evaluated.

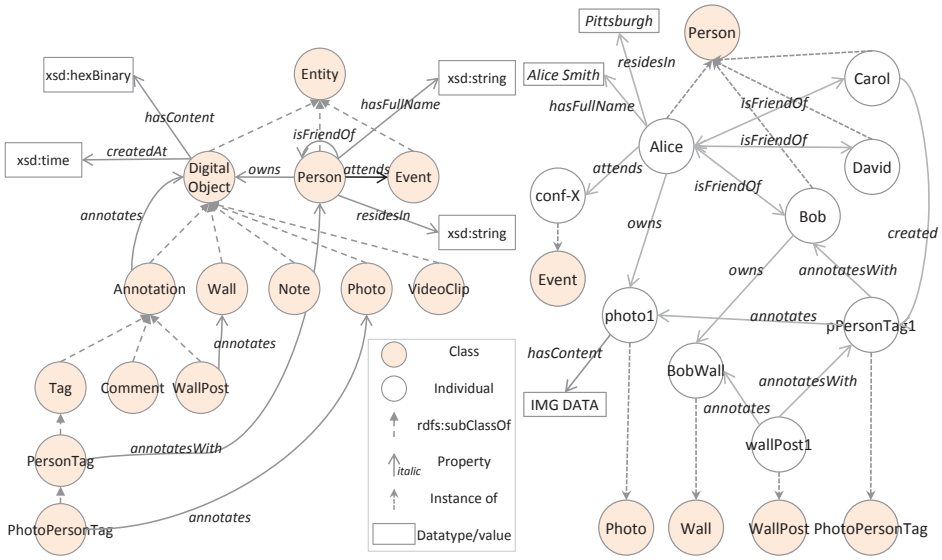
2.2 Social network system ontology

We propose SNO that models key entities and their relationships typically found in SNSs. This is partly because we could not find an appropriate ontology in the literature that can capture the details of objects in an SNS. Based on this, we elaborate and discuss various scenarios regarding our proposed access control model. Note, however, that our access control model is not tied or limited to this specific ontology. The current version of the ontology comprises of 14 concepts and ten object properties. Figure 1(a) depicts an overview of SNO.

The Entity concept is the root to all concepts in SNO, with three immediate descendants: DigitalObject, Person, and Event. The DigitalObject concept models any object with digital, usually presentable content. The Person concept models human users in the context of SNSs. The DigitalObject concept is specialised by subconcepts such as Note, Photo, Wall, and Annotation. The Note concept represents a textual content. The

actual content is linked to a Note object using hasContent datatype property. The Wall concept models the posting board on the homepage of a person in an SNS, such as the one Facebook provides. The Annotation concept represents special digital objects that instead of directly representing a content, annotate one object (e.g., a wall, a photo, etc.) using another object (e.g., a note, a person, etc.). The two objects are related to an annotation object, using properties Annotates and AnnotatesWith, respectively. Annotation itself is specialised by Comment, Tag, and WallPost. Comment annotates an object with a note. PhotoPersonTag is a specialised tag that annotates a photo with a person. WallPost annotates a wall with an object, e.g., a photo. We choose to represent annotation as a concept, rather than a relation, in order to be able to capture more semantics regarding it. For instance, it is usually important to know who has tagged a person in a photo; that might be different from the owner and the tagged person.

Figure 1 SNO, (a) SNO ontology (b) sample instantiation of SNO (see online version for colours)



(a) (b)

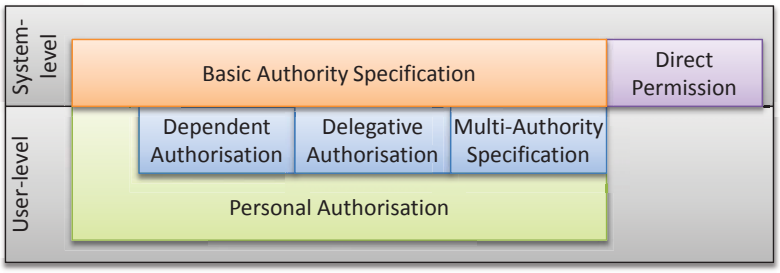
Figure 1(b) shows a small, sample instantiation of SNO. The knowledge describes Alice's name, where she resides, her friendship with Bob, Carol, and David, and an event she attends. Alice also owns photo photo1, in which Bob is tagged (pPersonTag1 annotates photo1 with Bob). This tag has been created by Carol. Moreover, this photo tag has been posted on Bob's wall, i.e., wallPost1 annotates Bob's wall, BobWall, with pPersonTag1. Using SNO concepts and relations, more complex semantics can be represented, which is not shown in the simple scenario described above. For instance, David may make a comment about Bob's wall post mentioned earlier. This can be represented by comment comment1 which annotates wallPost1 (i.e., annotates(comment1, wallPost1)) with a note. Throughout the paper, we use namespace prefix *sn* to refer to SNO concepts and relations.

3 Ontology-based social network access control model

We propose OSNAC, a rule-based access control policy model for SNSs based on semantic web standards. OSNAC is a fine-grained semantics-aware model that captures relations between ontological concepts of knowledge as protected resources. For this purpose, the model relies on an ontology such as SNO (introduced in Section 2.2) that models the SNS knowledge. It also uses an ACO (described in Section 3.1) to model the policies. We assume a closed-world policy model where an access is denied unless it is allowed according to access control policy rules.

Figure 2 shows the overall OSNAC policy framework. Access control rules are specified at two levels: user and system. At the user level, every user can express *personal authorisation* rules regarding protected resources. For more flexible authorisations, users can leverage *dependent authorisation*, *delegative authorisation* as well as *multi-authority specification* rules. At the system level, the rules govern the overall privacy policy of the system. *Basic authority specification* rules determine which users have authority over which protected resources. They empower users by recognising the authorisations defined at user-level as permissions. In other words, they aggregate user-level authorisations by determining the appropriate authority for protected resources. In contrast, *direct permission* rules indicate permissions that are valid independently of users' policies. System- level policies are specified by administrators according to application semantics of a particular SNS, which are naturally less frequently updated than the user-level policies. Note that a higher-level policy component in Figure 2 can be considered performing aggregation of its lower-level components. We elaborate on various components of the framework in the rest of the section.

Figure 2 OSNAC policy framework (see online version for colours)



3.1 Policy expression at ontology level

Since knowledge resources are captured in an ontology, such as SNO, the access control policies need to express them using ontology concepts. In order to facilitate an efficient and semantics-rich access control decision, we choose to capture information related to access control policy in a separate ontology, which we call the ACO. We use namespace prefix ac to refer to ACO concepts and relations, which are depicted in Figure 3. ACO is used to model and store any knowledge solely needed for access control purpose including inferences based on access control policy rules. We categorise the concepts and relations in ACO as follows.

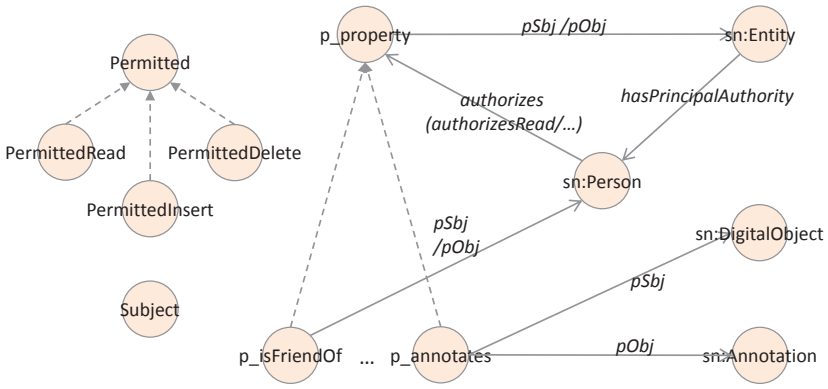
- *Access subject:* class ac:Subject is used to specify the access subject (an instance of sn:Person) of a given access request that is evaluated.
- *Reified properties:* we consider instances of SNO properties as the protected resources. However, since current semantic web languages such as OWL do not support expressions about property instances, which is needed for expressing authorisations, we reify SNO properties in ACO as follows. Class ac:p_property serves as an abstract reification of an SNO property. Properties ac:pSbj and ac:pObj relate class ac:p_property to its corresponding subject and object of the property in SNO, respectively. Corresponding to each property sn:*x*, there exists class ac:p_*x*, which is a subclass of ac:p_property. Thus, a relation such as sn:isFriendOf(Alice, Bob) in SNO is correspondingly represented in ACO using an instance of class ac:p isFriendOf; its subject and object are related using relations ac:pSbj(Alice) and ac:pObj(Bob), respectively.
- *Authorisations:* an authorisation is issued by a user to authorise the subject to access a property instance. Property ac:authorizes abstractly relates the user who issues the authorisation (an instance of sn:Person) to the reified property instance that is to be accessed (an instance of a ac:p_property descendant). Depending on the mode of access, one of the descendants of ac:authorizes will be used, which include ac:authorizesRead, ac:authorizesDelete, and ac:authorizesInsert.
- *Permissions:* a permission specifies an access granted by the SNS to a subject. Note that this is usually inferred based on user-level authorisations. Class ac:Permitted represents an abstract notion of such a permission. A reified property instance becomes also an instance of ac:Permitted if access to it is inferred to be granted by the SNS. Depending on the mode of access, one of the descendants of ac:Permitted will be used, which include ac:PermittedRead, ac:PermittedDelete, and ac:PermittedInsert.
- *Principal authority:* We assume a unique principal authority is assigned for every SNO class individual using property ac:hasPrincipalAuthority. The principal authority is most probably the originator of the object, and is determined by the system. In practice, principal authorities can be inferred based on other properties captured in SNO such as sn:owns or sn:created, that may be defined between an sn:Person instance and an sn:Entity instance. We elaborate more on how this concept is used in our model in Section 3.2.2.

The access control policy rules in our model follow the syntax mentioned in Section 2.1. In order to efficiently express reified properties as protected resource in the rules we introduce the following shorthand.

Definition 1: (*Reified property shorthand*) Expression [?*rsc* ← sn:*p*(*s*, *o*)] represents ?*rsc* as the protected reified property instance of type sn:*p* that relates property subject *s* to property object o, i.e., [?*rsc* ← sn:*p*(*s*, *o*)] ≡ ac:p_*p*(?*x*) ∧ ac:pSbj(?*x*, *s*) ∧ ac:pObj(?*x*, *o*).

To define rules, we only use abstract authorisation and permission predicates. For instance, we use ac:authorizes in the format of personal authorisations. However, an actual rule needs to use one of its descendants as mentioned in Section 3.1.

Figure 3 Access control ontology (see online version for colours)



3.2 System-level policy rules

System-level access control policy rules are specified by SNS administrators. Table 1 shows the format of system-level policy rules. In these rules, $[?rsc \leftarrow \mathsf{sn}{:}p(s, o)]$ specifies the protected resource according to the reified property shorthand. $\mathcal{P}$ is a conjunction of zero or more of either SNO predicates, ac:Subject, or ac:hasPrincipalAuthority, which is used to express more specifically where a rule applies. We call $\mathcal{P}$ a *rule extension sentence*.

Table 1 System-level access control policy rules

Direct permission:
$\mathcal{P} \wedge [?rsc \leftarrow \mathsf{sn} : p(s,o)] \Rightarrow \mathsf{ac} : \mathsf{Permitted}(?rsc)$
Basic authority specification:
$\mathcal{P} \wedge [?rsc \leftarrow \mathsf{sn} : p(s,o)] \wedge \bigwedge_{i=1}^{n\geq 1} \mathsf{ac} : \mathsf{authorizes}(u_i, ?rsc) \Rightarrow \mathsf{ac} : \mathsf{Permitted}(?rsc)$

3.2.1 Direct permission

Direct permissions allow the system to grant permissions to users without the involvement of user authorities. As shown in Table 1, a direct permission rule includes a rule extension sentence and a property resource specification in the antecedent, and an ac:Permitted descendant as the consequent.

Example 1: The following two direct permission rules entitle everyone to read the relations that are defined about the objects for which he/she is the principal authority.

$$\mathsf{ac} : \mathsf{Subject}(?\mathrm{sbj}) \wedge \mathsf{ac} : \mathsf{hasPrincipalAuthority}(?s, ?sbj)$$
$$\wedge \left[?rsc \leftarrow \mathsf{sn} : \mathsf{property}(?s, ?o)\right] \Rightarrow \mathsf{ac} : \mathsf{PermittedRead}(?rsc).$$
$$\mathsf{ac} : \mathsf{Subject}(?\mathrm{sbj}) \wedge \mathsf{ac} : \mathsf{hasPrincipalAuthority}(?o, ?sbj)$$
$$\wedge \left[?rsc \leftarrow \mathsf{sn} : \mathsf{property}(?s, ?o)\right] \Rightarrow \mathsf{ac} : \mathsf{PermittedRead}(?rsc).$$

In the first rule in the above example, the first two predicates in the body constitute the rule extension sentence, and the third predicate specifies the protected property instance. Note that ac:p_property is considered as the superclass of any reified SNO property; therefore, the reified property shorthand applies to any property with bound variables *s* and *o* for its subject and object. The first predicate of the rule extension sentence indicates *sbj* to be the access subject, and the second one indicates *sbj* to be the principal authority for a protected property's subject. Finally, the consequent predicate allows read access to such a property for the access subject. The second rule is only different in considering access subject to be the principal authority for a protected property's object.

3.2.2 Basic authority specification

Most of the access control decisions in a SNS are desirable to be mediated using policies determined by the relevant users, instead of direct permissions by the SNS itself. In this respect, the role of system-level rules is to determine policy authorities for resources. According to Table 1, authority specification rules follow a format similar to that of direct permissions, i.e., having a rule extension sentence $\mathcal{P}$ and reified resource specification in antecedent and ac:Permitted descendent in consequent. But they also include a conjunction of ac:authorizes predicates, which indicates users whose authorisations matter in granting permissions on the specific protected resources. This means that permissions are granted based on user-level authorisations. Although an SNS may define a set of authority specification rules customised for its own application semantics, here, we propose a generic, basic authority model for an ontology-based knowledge base system. Assuming there is a principal authority for every SNO individual, as described in Section 3.1, it is safe to consider the same authority to be effective for any property instances associated with that individual. Hence, the authority over an object property instance can be determined based on the principal authorities of the related individuals. For instance, access to sn:isFriendOf(Alice,Bob) is under the authority of both Alice and Bob. The authority over a datatype property instance is also the principal authority of the only related object. The basic authority model for read access can be expressed using the following rule.

$$\begin{aligned}&\mathsf{ac:hasPrincipalAuthority}(?s,?u_1)\big]\wedge\mathsf{ac:hasPrincipalAuthority}(?o,?u_2)\big]\\&\wedge\ [?rsc\leftarrow\mathsf{sn:property}(?s,?o)]\\&\wedge\ \mathsf{ac:authorizesRead}(?rsc)\wedge\mathsf{ac:authorizesRead}(?u_2,?rsc)\\&\Rightarrow\mathsf{ac:PermittedRead}(?rsc)\end{aligned}\tag{1}$$

The above rule grants a read permission on a property instance only if both the principal authorities of the individuals associated with the property instance authorise that access.

3.3 User-level policy rules

User-level access control policy rules are specified by SNS users regarding the protected resources that they have authority over. The actual effectiveness of such a rule is determined according to system authority policies. Table 2 shows the format of user-level policy rules. Similar to system-level policy rules, user-level policy rules include a *rule extension sentence* $\mathcal{R}$, which is a conjunction of zero or more of either SNO predicates or

ac:Subject, and a reified property shorthand for specifying protected resource in antecedent. Also, all user-level policy rules have an ac:authorizes descendant in consequent. The first argument of this predicate has to be the user who specifies the authorisation. Otherwise, user authorities may be misused. We assume that SNSs enforce this requirement.

Table 2 User-level access control policy rules

Personal authorisation:
$\mathcal{R} \wedge [?rsc \leftarrow \mathsf{sn}: p(s,o)] \Rightarrow \mathsf{ac}:\mathsf{authorizes}(u,?rsc)$
Dependent authorisation:
$\mathcal{R} \wedge \left[?rsc_1 \leftarrow \mathsf{sn}: p_1(s_1,o_1)\right] \wedge \left[?rsc_2 \leftarrow \mathsf{sn}: p_2(s_2,o_2)\right] \wedge \mathsf{ac}:\mathsf{authorizes}(u,?rsc_1)$ $\Rightarrow \mathsf{ac}:\mathsf{authorizes}(u,?rsc_2)$
Delegative authorisation:
$\mathcal{R} \wedge [?rsc \leftarrow \mathsf{sn}: p(s,o)] \Rightarrow \mathsf{ac}:\mathsf{authorizes}(u_2,?rsc) \Rightarrow \mathsf{ac}:\mathsf{authorizes}(u_1,?rsc)$
Disjunctive multi-authority specification:
$R = \{R_i\}$
$R_i = \mathcal{R}_i \wedge [?rsc \leftarrow \mathsf{sn}: p(s,o)] \wedge \mathsf{ac}:\mathsf{authorizes}(u_i,?rsc) \Rightarrow \mathsf{ac}:\mathsf{authorizes}(pa,?rsc)$
Conjunctive multi-authority specification:
$\mathcal{R} \wedge [?rsc \leftarrow \mathsf{sn}: p(s,o)] \wedge \bigwedge_{i=1}^{n} \mathsf{ac}:\mathsf{authorizes}(u_i,?rsc) \Rightarrow \mathsf{ac}:\mathsf{authorizes}(pa,?rsc)$

3.3.1 Personal authorisation

A personal authorisation rule expresses a permission granted by an individual user to others. According to the rule format shown in Table 2, a personal authorisation rule uses a rule extension sentence and a reified property shorthand in antecedent and an ac:authorizes descendant in consequent.

Example 2: The following rule, expressed by Alice, authorises her friends to read the photo-tags she has been marked with

$$\mathsf{ac}:\mathsf{Subject}(?sbj, \mathsf{sn}:\mathsf{isFriendOf}(?sbj, \mathsf{Alice}) \wedge \mathsf{sn}:\mathsf{PhotoPersonTag}(?pTag)$$
$$\wedge\ [?rsc \leftarrow \mathsf{sn}:\mathsf{anonatesWith}(?pTag, \mathsf{Alice})] \Rightarrow \mathsf{ac}:\mathsf{authorizesRead}(\mathsf{Alice}, ?rsc)$$

In the above example, the first three antecedent predicates compose the rule extension sentence. The first and second predicates specify the access subject to be Alice's friend, and the third predicate declares variable *pTag* to be of type sn:PhotoPersonTag. The forth predicate specifies the protected resource to be an sn:annotatesWith relation that relates PhotoPersonTag *ptag* to Alice. The consequent predicate indicates that Alice authorise such an access request.

3.3.2 Dependent authorisation

Dependent authorisation rules allow one authorisation to be inferred based on another authorisation. This is often useful in scenarios where authorisations need to be derived for related protected resources. As the format shown in Table 2, dependent authorisation rules include a rule extension sentence, two protected property specifications, and an

ac:authorizes descendent for the first protected property in antecedent, and an ac:authorizes descendent for the second protected property in consequent. In other words, dependent authorisation allows for authorisation propagation from protected resource rsc_1 to protected resource rsc_2 under specific condition determined by rule extension sentence $\mathcal{R}$.

Example 3: In the scenario depicted in Figure 1(b), suppose Alice has specified detailed authorisation rules for different types of annotations on her photo. Then she can make sure whoever gets to access the annotations can also access the photo content using the following rule, without redefining all the restrictions.

$$\left[?rsc_1 \leftarrow \mathsf{sn:anonates}(?x, \mathsf{photo1})\right] \wedge \left[?rsc_2 \leftarrow \mathsf{sn:hasContent}(\mathsf{photo1}, ?c)\right]$$
$$\wedge\ \mathsf{ac:authorizesRead}(\mathsf{Alice}, ?rsc_1) \Rightarrow \mathsf{ac:authorizesRead}(\mathsf{Alice}, ?rsc_2)$$

3.3.3 Delegative authorisation

Delegation has been shown to be useful in conjunction with access control models (Barka and Sandhu, 2000). In the context of the proposed model, we observe that delegating authority improves the flexibility of policies. Based on a delegative authorisation, a user delegates its authority over a specific resource to another user. According to the rule format shown in Table 2 user u_1 delegates authorisation on a specific resource to user u_2. In other words, user u_1 respects the authorisations made by user u_2 on that resource. Delagative authorisations may be used to relax authority on the protected relations. The basic authority specification rule stated in Section 3.2.2 [rule (1)] requires both the end authorities of a relation to authorise a permission, in order for it to be granted. However, such a mutual agreement might be too restrictive for some users and resources. The two end authorities can use delegative authorisations to respect one another's decisions on the specific permission(s) of choice, without a change in system-level rules.

Delegative authorisations are very flexible and secure in terms of delegation power. First, an authority can flexibly customise the permission. For instance, it can restrict the target subjects to have certain characteristics, or the resource/operation to be of certain type. Second, subsequent updates to the delegative authorisation rule will be applied seamlessly, without a need to worry about grant/revoke propagation issues that delegation models usually deal with. This is because unlike traditional delegation models, the permissions are not explicitly transferred; the authorisation rule is the sole means of delegation. Third, since delagative authorisations are at the user level, there is no need to assure that the delegator actually has the authority on the permission. Only the valid delegations will be effective based on the system-level authority specification rules.

3.3.4 Multi-authority specification

There are scenarios in SNSs where more authorities are desired to weigh in on an access control decision than just the directly related authorities. We support multi-authority specification in two ways. A principal authority may use multiple delegative authorisation rules to enable a *disjunctive multi-authority*. Such a multi-authority is disjunctive in the sense that a permission authorisation by any corresponding authority in the set is a sufficient condition for that permission to be considered authorised by the principal authority. Alternatively, a principal authority may create a *conjunctive*

multi-authority, in which every involved authority is required to authorise a permission in order that it would be considered authorised by the principal authority. Table 2 shows the formats of a rule set and a single rule that establish disjunctive and conjunctive multi-authority, respectively, where principal authority user *pa* shares the authority with users u_1, u_2, …, and u_n.

3.4 Access control enforcement

A basic access request is a triple $\langle sbj, rsc, opr \rangle$, where sbj is the user who requests the access (instance of sn:Person), $rsc = p(s, o)$ refers to the property instance to be accessed (instance of ac:p_property), and *opr* is the mode of access requested (read/delete/insert).

Definition 2: (*Access authorisation*) Given an access request $\langle sbj, p(s, o), opr \rangle$, the access is granted if and only if the following sentence is satisfied in the knowledge base, given the fact Subject(sbj):

$$[?rsc \leftarrow p(s,o)] \wedge \mathsf{ac:Permitted}(?rsc)$$

where predicate ac:Permitted is substituted with its proper descendant corresponding to *opr*. The access is denied otherwise.

We note that in the case of information retrieval from an SNS knowledge base multiple relations may be queried and evaluated simultaneously in order to retrieve a result set of interest. Conceptually, for each valid variable assignment in a query, every bound relation needs to be considered as one basic access query. However, an access authorisation per such relations is not efficient, and needs modification of the retrieval engine. Alternatively, we augment a query with access check primitives and evaluate that in order to retrieve only the authorised results.

Definition 3: (*Query access authorisation*) Let $\langle sbj, Q \rangle$ be a query access request by subject *sbj*, where $Q_W = \bigwedge_{i=1}^{n} \mathsf{sn}: p_i(s_i, o_i)$ represents the conjunctive WHERE clause of query *Q*. A retrieval engine automatically enforces the access control policy and retrieves the authorised result by evaluating:

$$Q'_W = \bigwedge_{i=1}^{n} \left\{ \mathsf{sn}: p_i(s_i, o_i) \wedge \left[?rsc_i \leftarrow \mathsf{sn}: p_i(s_i, o_i) \right] \wedge \mathsf{ac:PermittedRead}(?rsc_i) \right\}$$

given the fact Subject(sbj).

Each relation predicate in the original query is followed by two predicates for access control purpose: the first predicate bounds the relation to a resource variable, and the second predicate checks if the subject has permission to access the resource. A query that is augmented with access primitives can be directly processed by a query retrieval engine on the ontology, while access control policy rules are enforced seamlessly using an ontology reasoner.

Example 4: Suppose Bob requests access to the list of Alice's friends who reside in Pittsburgh. This is a complex query that involves accessing the list of Alice's friends, where they live, and their names. The following is a SPARQL-like syntax for this query.

SELECT $?x$?fname
WHERE { sn:friendOf(Alice, $?x$)
$\wedge$ sn:residesIn($?x$, Pittsburgh)
$\wedge$ sn:hasFullname($?x$, ?fname)}

Its access-augmented WHERE clause will be as follows.

sn:friendOf(Alice, $?x$)
$\wedge$ [$?rsc1 \leftarrow$ sn:friendOf(Alice, $?x$)] $\wedge$ ac:PermittedRead($?rsc1$)
$\wedge$ sn:residesIn($?x$, Pittsburgh)
$\wedge$ [$?rsc2 \leftarrow$ sn:residesIn($?x$, Pittsburgh)] $\wedge$ ac:PermittedRead($?rsc2$)
$\wedge$ sn:hasFullname($?x$, ?fname)
$\wedge$ [$?rsc3 \leftarrow$ sn:hasFullname($?x$, ?fname)] $\wedge$ ac:PermittedRead($?rsc3$)

When executing the augmented query in Example 4, if Bob does not have access to even one of the relations in the query corresponding to a specific Alice's friend, that person's information will not be retrieved. Thus, the result set reflects the authorised information according to the access control policies.

3.5 Supporting negative authorisation

The OSNAC policy model relies on positive authorisations. If the system cannot resolve a corresponding positive permission (i.e., descendants of ac:Permitted) for an access request, then the request is denied. Although positive authorisation can be used to express security policies in general, it is sometimes desirable to express intended policies using a mixture of positive and negative authorisations. Unfortunately, OWL and SWRL do not support negation-as-failure due to open-world assumption of the Semantic Web. This prevents us from reasoning collectively on positive and negative authorisations. To be more specific, if an authorisation cannot be inferred its negation cannot be inferred either. Our proposed workaround to support negative authorisation is to introduce separate predicates for negative authorisations and resolve the conflicts at the query processing time using SPARQL, once the inference is done. For this purpose we extend ACO, that was described in Section 3.1, as follows. Analogous to property ac:Permitted and its descendants, we define property ac:Prohibitted and its corresponding descendants (i.e., ac:ProhibittedRead, ac:ProhibittedInsert, and ac:ProhibittedDelete), that represent negative permissions. Also, we define property ac:deny and its corresponding descendants for negative user-level authorisation. In order to fully enable negative authorisations, we need to also specify corresponding basic authority specification rules for negative authorisations, in a manner similar to what is described in Section 3.2.2 but for negative permissions.

This approach satisfies the need for the common use of negative authorisations. Users will be able to express negative exceptions to positive authorisations, and any conflict are resolved at the retrieval time. We follow a *denial-takes-precedence* approach, when both ac:Permitted and ac:Prohibitted predicates are resolved for a specific access.

Definition 4: (*Query access authorisation with negation*) Let $\langle sbj, Q \rangle$ be a query access request by subject *sbj*, where $Q_W = \bigwedge_{i=1}^{n} \mathsf{sn} : p_i(s_i, o_i)$ represents the conjunctive WHERE clause of query *Q*. A retrieval engine automatically enforces the access control policy with negative authorisations and retrieves the authorised result by evaluating the following, given the fact Subject(sbj):

$$Q'_W = \bigwedge_{i=1}^{n} \begin{pmatrix} \mathsf{sn} : p_i(s_i, o_i) \\ \wedge \left[?rsc_i \leftarrow \mathsf{sn} : p_i(s_i, o_i) \right] \wedge \mathsf{ac:PermittedRead}(?rsc_i) \\ \wedge \ \mathsf{OPTIONAL} \left\{ \mathsf{ac:ProhibittedRead}(?rsc_j) \right\} \\ \wedge \ \mathsf{FILTER} \left(!\mathsf{bound}(?rsc_j) \right) \end{pmatrix}$$

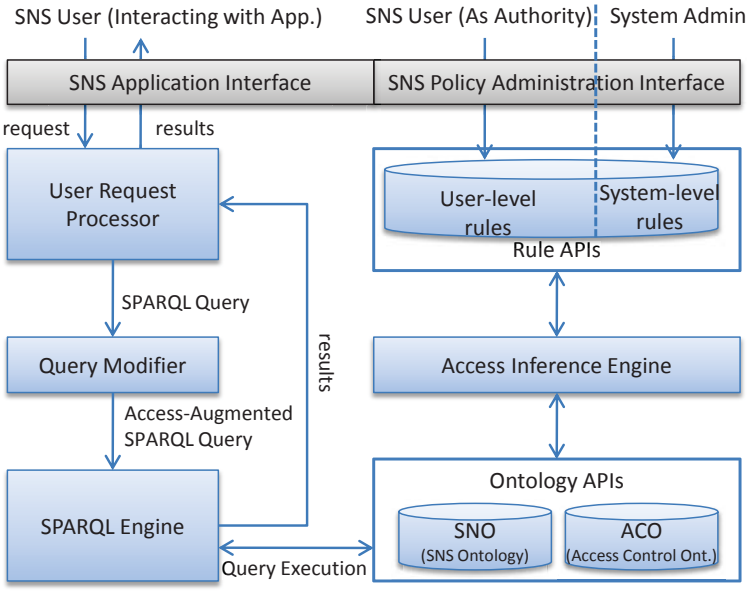
4 Implementation

4.1 Design and architecture

We have developed a prototype implementation of an SNS knowledge-base that is protected based on the proposed OSNAC model. The implementation has been done in Java language based on the Jena semantic web framework (http://jena. sourceforge.net).We leverage Jena's TDB for persistent storage of SNO and ACO. In an initialisation phase, based on the SNS knowledge captured in SNO, ACO is populated with the corresponding reified properties as described in Section 3.1.

Figure 4 illustrates the architecture of the prototype implementation. Access control policy rules are provided by users and system administrators, using separate interfaces, and are stored in the policy rule-base. Rules are expressed using SWRL as explained in Section 3. However, since SWRL is not directly supported in Jena, we programmatically convert rules to Jena's own rule language in a policy compilation phase. Note that there is no loss of expressiveness in this process. At run time, the *user request processor* accepts the requests from a user (in fact, from the SNS on behalf of a user), and passes it to the *query modifier* module, where it is augmented with access control primitives (refer to Section 3.4). The modified query is then sent to the *SPARQL engine*. Before execution of the query by engine, a fact is inserted in the knowledge base asserting the user to be the access subject. The SPARQL engine then interacts with the SNO and ACO to retrieve the query results. In the retrieval process, the *access inference engine* employs Jena general purpose rule engine to infer access primitive predicates (i.e., ac:authorizes and ac:Permitted) based on the knowledge stored in the ontologies and according to the access control policy rules. The access subject assertion is removed from knowledge base after query has been executed. Finally, the authorised query results are returned to the *user request processor*.

Figure 4 Architecture of the prototype implementation (see online version for colours)



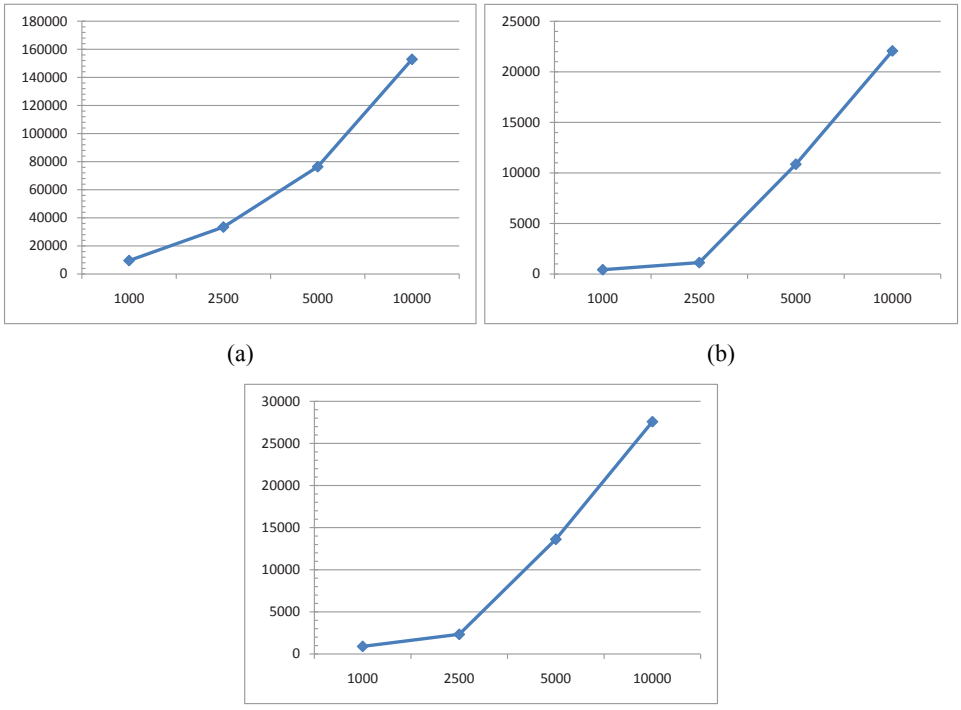
4.2 Access control enforcement

We have conducted tests on the access control engine by submitting SPARQL queries on a sample populated SNO [extended version of Figure 1(b)]. The engine successfully returns only the authorised information that is expected according to the sample access control policy rules. We also developed a data generator that randomly populates an SNO ontology with users, friendship links, photos, and photo tags. For the purpose of performance evaluation, we decided to focus on a limited scenario where users want to control their friendship links. It is much more easier to evaluate the effects of change in the number of users and their protected resources (i.e., friendship links here) on access control performance, without complicating the problem. We run the prototype on a standard desktop PC, and each measurement is the average result of ten runs.

We measure three performance parameters of the prototype access control engine. *Initialisation time* is the time it takes to load the ontologies and populate ACO with reified properties. *Inference time* is the time it takes to load access policy rules and build an inference model in Jena for reasoning on the ontologies. Finally, *access check time* is the time it takes for the engine to answer a simple predicate query involving an access check. Figure 5 shows the performance results of the access control engine for different data sizes. In each data point, we consider on average 60 friends per user. The first two steps, i.e., initialisation and inference, need to be performed once when the system starts. Therefore, it can be acceptable to spend about 36 seconds for completing these two steps with 2,500 users. The most important parameter here is the access check time, which is about two seconds for 2,500 users in our experiment. Although, this seems a little expensive, the performance can be improved by employing better software and hardware tools. Moreover, the access check time seems to be linear with the number of users (note that the scale is not linear in the horizontal axis). Since the ratio of friends per user is an

important parameter in the generated dataset and performance of the system, we also evaluated our model under different ratios of friends per user. Note that this ratio is currently about 130 for Facebook. The results are shown in Figure 6. All the performance increase at a linear rate, which is lower up to 60 and gets higher after that.

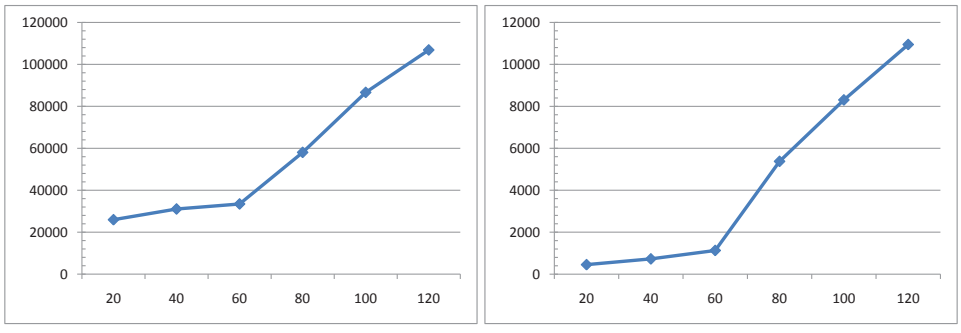
Figure 5 Performance of the prototype engine for different number of users (with avg. 60 friends), (a) initialisation time (ms) (b) inference time (ms) (c) access check time (ms) (see online version for colours)



(a) (b)

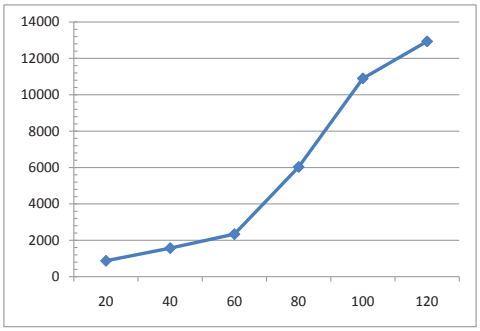
(c)

Figure 6 Performance of the prototype engine for different number of friends per user (2,500 users), (a) initialisation time (ms) (b) inference time (ms) (c) access check time (ms) (see online version for colours)



(a) (b)

Figure 6 Performance of the prototype engine for different number of friends per user (2,500 users), (a) initialisation time (ms) (b) inference time (ms) (c) access check time (ms) (continued) (see online version for colours)



(c)

5 Related work

Access control research in social network area is still in its early stages. Initial access control solutions for SNSs propose trust-based access control policies that are inspired by research developments in trust and reputation computation in social networks. Friend of a friend (FOAF)-realm (Kruk, 2004; Kruk et al., 2006) is one of the earliest approaches that quantifies the *knows relations* in the context of FOAF ontology as a trust metric, and support rules that control accesses of friends to resources in a social network by stating the maximum distance and minimal friendship level. Carminati et al. (2006) propose a conceptually-similar but more complete trust-based access control model. Villegas et al. (2008) propose to use a slightly different trust measure by automatically classifying nodes in zones. A general drawback of trust-based access control models is the usability issues, as it could be very hard to comprehend and specify appropriate trust thresholds, and hence be left with even less protection than simple, conventional access control approaches. While these approaches focus mainly on subject specification based on distance and trust measures, we take a more abstract approach and focus instead on accurately capturing the information semantics using an ontology-based access control policy. Trust information can be straightforwardly used in our approach if captured in the ontology, independently from underlying trust computation mechanism.

The closest work to this paper is probably the semantic web-based access control framework (Carminati et al., 2009), which also leverages OWL and SWRL. The authors define three type of policies, namely, access control policy, filtering policy, and admin policy. Access control policies are positive authorisation rules; filtering policies can limit someone's access to information by herself (not conceptually a security issue); and admin policies can express who are authorised to define those policies. Although they outline an access control framework, lack of formal descriptions and implementation leaves behind many ambiguities. In comparison, we propose a more detailed and semi-formal semantics for our model, and show the applicability by implementing a proof-of-concept framework. Also, our model captures the notion of individual authorities, and provide access control policies to protect the relations in the knowledge ontology as a more expressive and flexible alternative to entity protection. Ryutov et al. (2009) propose a

rule-based access control model for semantic networks, based on a constrained first order logic. The authors have implemented this model in a RDF-like framework. While the model is based on logic rules similar to our approach, notions such as *attaching policies* and separating policy at subject and object level are introduced but inadequately elaborated and justified in their work. Also, relations are mainly used in the access control rules, but not as of the protection objects; it seems that their approach only protects the objects at entity level. There are also other access control approaches for social networks that go beyond our focus in this work, such as protection against third-party applications (Shehab et al., 2008).

In the area of semantic web, Rei (Kagal et al., 2004) is a prominent policy language based on RDFS. Although Rei leverages semantic web languages it mainly provides a generic framework to support different deontic concepts in the policy (i.e., permission, prohibition, obligation, and dispensation), and distributed policy management. However, in terms of specifying subjects and protection objects it uses generic conditions, not specific to semantic web. In contrast, our model is more focused on how to specify fine-grained policy rules on a knowledge base that is specified using OWL. More closer to our work, there exist access control solutions for RDF stores, although not in the context of SNSs. Reddivari et al. (2005) propose RAP, a rule-based model and architecture. In RAP, access control policy is written using Jena framework rules, and supports both permit and prohibit predicates, similar to OSNAC features. Although no experimental results are reported, RAP does not seem to be a very efficient access control method. For a given query to the RDF store, the result set is retrieved first. Then the access control inference is performed separately for every triple in the result in order to decide to include it in the final result. Our query augmentation approach performs more efficiently. The use of access primitive predicates in the query avoids excessive overhead of access checks leveraging the query engine itself. There are other approaches to access control on RDF stores that are comparatively less grounded (Dietzold and Auer, 2006; Dersingh et al., n.d.; Liu, Xie, Li et al., 2009), or support a specific policy such as multi-level security (Jain and Farkas, 2006).

6 Conclusions

In this paper, we proposed OSNAC, an ontology-based access control model based on semantic web standards that empowers the individual users of a SNS to express fine-grained access control policies on their related information. We proposed an ontology for SNSs to further demonstrate our approach. The key idea in OSNAC is to express the policies on the relations among concepts in the social network ontology. We also provide policy means for the system to define an authority model, that decides which users' policies are effective on what protected resources. Moreover, the advanced policy rules provide more flexibility to the users, in delegating their power, and sharing the authority over specific objects, i.e., enabling multi-authority specification. We also implemented a framework prototype of the proposed model, evaluated its performance, and showed the applicability of our approach.

OSNAC provides powerful access control features for the users of SNSs. Existence of privacy options such as "share a tagged post with friends of the friend I tag" in Facebook shows the need for such expressive policies. However, even savvy users of SNSs do not have to be able to compose access control policy rules manually. An SNS employing

OSNAC may simply provide a user interface similar to the current practices, but with more flexible options to its user; then, provide the access control engine with policy rules corresponding to the user choices. Investigating user-friendly interfaces to enable users to fully benefit from OSNAC features will be our future work.

Acknowledgements

This research has been supported by the US National Science Foundation award IIS-0545912.

References

Barka, E. and Sandhu, R.S. (2000) 'Framework for role-based delegation models', *Proc. 16th Annual Computer Security Applications Conference*, IEEE Computer Society, pp.168–176, available at http://dx.doi.org/10.1109/ACSAC.2000.898870.

Carminati, B., Ferrari, E. and Perego, A. (2006) 'Rule-based access control for social networks', in R. Meersman, Z. Tari and P. Herrero (Eds.): *Proc. OTM 2006 Workshops (On the Move to Meaningful Internet Systems)*, pp.1734–1744, Vol. 4278 of LNCS, Springer, available at http://dx.doi.org/10.1007/11915072_80

Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M. and Thuraisingham, B. (2009) 'A semantic web based framework for social network access control', *Proc. 14th ACM Symposium on Access Control Models and Technologies*, ACM, pp.177–186, available at http://dx.doi.org/10.1145/1542207.1542237

Dersingh, A., Liscano, R., Jost, A., Finnson, J. and Senthilnathan, R. (n.d.) 'Utilizing semantic knowledge for access control in pervasive and ubiquitous systems, mobile networks and applications', available at http://dx.doi.org/10.1007/s11036-009-0180-7.

Dietzold, S. and Auer, S. (2006) 'Access control on RDF triple stores from a semantic wiki perspective', *Scripting for the Semantic Web Workshop at 3rd European Semantic Web Conference (ESWC)*, available at http://semanticscripting.org/SFSW2006/Paper3.pdf.

Jain, A. and Farkas, C. (2006) 'Secure resource description framework: an access control model', *SACMAT '06: Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, ACM, New York, NY, USA, pp.121–129, available at http://dx.doi.org/10.1145/1133058.1133076

Kagal, L., Finin, T., Paolucci, M., Srinivasan, N., Sycara, K. and Denker, G. (2004) 'Authorization and privacy for semantic web services', *IEEE Intelligent Systems*, Vol. 19, No. 4, pp.50–56, available at http://dx.doi.org/10.1109/MIS.2004.23.

Kruk, S.R. (2004) 'FOAF-realm: control your friends access to the resource', *Proc. FOAF Workshop*.

Kruk, S.R., Grzonkowski, S., Gzella, A., Woroniecki, T. and Choi, H.C. (2006) 'D-FOAF: distributed identity management with access rights delegation', *Proc. 1st Asian Semantic Web Conference*, Springer, pp.140–154, available at http://dx.doi.org/10.1007/11836025_15.

Liu, M., Xie, D., Li, P., Zhang, X. and Tang, C. (2009) 'Semantic access control for web services', Vol. 2, pp.55–58, available at http://dx.doi.org/10.1109/NSWCTC.2009.389.

Reddivari, P., Finin, T. and Joshi, A. (2005) 'Policy-based access control for an RDF store', *Workshop on Policy Management for the Web*, pp.78–81, available at http://ebiquity.umbc.edu/get/a/publication/323.pdf.

Ryutov, T., Kichkaylo, T. and Neches, R. (2009) 'Access control policies for semantic networks', pp.150–157, available at http://dx.doi.org/10.1109/POLICY.2009.11.

Shehab, M., Squicciarini, A. and Ahn, G-J. (2008) 'Beyond user-to-user access control for online social networks', *ICICS '08: Proceedings of the 10th International Conference on Information and Communications Security*, Springer-Verlag, Berlin, Heidelberg, pp.174–189, available at http://dx.doi.org/10.1007/978-3-540-88625-9_12.

Villegas, W., Ali, B. and Maheswaran, M. (2008) 'An access control scheme for protecting personal data', *Proc. 6th Annual Conference on Privacy, Security and Trust*, pp.24–35.

W3C (2004a) *OWL Web Ontology Language – Overview*, Recommendation ed.,. available at http://www.w3.org/TR/owl-ref/.

W3C (2004b) *SWRL: A Semantic Web Rule Language – Combining OWL and RuleML*, Submission ed., available at http://www.w3.org/Submission/SWRL/.

W3C (2008) *SPARQL Query Language for RDF*, Recommendation ed., available at http://www.w3.org/TR/rdf-sparql-query/.