OSNAC: An Ontology-Based Access Control Model for Social Networking Systems

Amirreza Masoumzadeh
School of Information Sciences
University of Pittsburgh
amirreza@sis.pitt.edu

James Joshi
School of Information Sciences
University of Pittsburgh
jjoshi@sis.pitt.edu

***Abstract*—As the information flowing around in social networking systems is mainly related or can be attributed to their users, controlling access to such information by individual users becomes a natural 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 few existing models. We demonstrate the applicability of our approach by implementing a proof-of-concept prototype of the proposed access control framework.**

I. Introduction

Social network(ing) systems (SNSs) are increasingly becoming a major type of online applications that facilitate digital social interaction 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, preserving users' privacy against 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. 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 modeling, 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 exist no basis of verifying consistency in policy specification and enforcement. For instance in Facebook, a user can choose to hide her relationship status 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 disables being 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 [1][2][3][4]. 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 modeling of knowledge using Semantic Web technologies. Some recent work also propose to use ontologies [5][6], 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 recognized 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 summarized as follows.

- We propose Social Networking 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 Ontology-based Social Network Access Control (OSNAC) model which addresses the protection of semantic-rich information in a knowledge base ontology such as SNO, using an access control ontology and access control policy rules. It supports both user-defined authorization rules and system-level authority policy. The model also supports advanced policy rules that provide flexible controls for users in SNSs in addition to the basic policies. These include delegation of authority, various definable interdependencies among user authorizations, and the ability to enable multiple authorities to enforce a composite policy regarding a protected resource.
- 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.

The rest of the paper is organized as follows. In Section II, we provide a brief introduction to the standards used, and propose an ontology for representing knowledge in an SNS. In section III, we present our proposed access control model, including the access control ontology, various supported policies, the enforcement model, and support for negative authorization. We provide details about our prototype implementation and results in Section IV. In Section V, we review related literature, and subsequently conclude the paper in Section VI.

II. Preliminaries

A. Semantic Web Standards

Web Ontology Language[1] (OWL) is a W3C recommendation to express meanings and semantics, which builds on RDF/RDFS. It consists of three main constructors: owl:Class defines an abstract notion for a class of objects, owl:ObjectProperty relates objects (class instances) to objects, and owl:DataTypeProperty relates objects to datatype values (e.g., strings or binary values). We have chosen OWL DL sublanguage 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. Semantic Web Rule Language[2] (SWRL) allows combining Horn-like rules with an OWL knowledge base, thereby enabling new knowledge reasoning tools. We use SWRL to express our access control policy rules. SPARQL[3] (SPARQL Protocol and RDF Query Language) is a syntactically-SQL-like language for querying RDF graphs via pattern matching. We use SPARQL for access control enforcement purpose.

B. SNS Ontology

We propose Social Networking systems Ontology (SNO) that models key entities and their relationships typically found in SNSs; partly because we could not find an appropriate ontology in the literature. 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 10 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, typically visualizable content. The Person concept models human users in the context of SNSs. The DigitalObject concept is specialized by subconcepts such as Note, Photo, Wall, and Annotation. The Note concept represents a textual content. 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 textual comment, a person, etc.). The two objects are related to an annotation object, using properties Annotates and AnnotatesWith, respectively. Annotation itself is specialized by Comment, Tag, and WallPost. Comment annotates an object with a note. PhotoPersonTag is a specialized 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.b shows a piece of an instantiated knowledge based on proposed SNO. The knowledge describes Alice's name, where she resides, her friendship with Bob, Carol, and David, and events she attends. Alice also owns photo photo1, in which Bob is tagged by Carol. This PhotoPersonTag is represented by pPersonTag1. Using SNO concepts and relations, more complex semantics can be represented, which is not shown in the sample instantiation. For instance, the tag mentioned earlier may need to be posted on Alice's wall. For this purpose, a WallPost instance should be created, e.g., wallPost1, having relations Annotates(wallPost1, aliceWall) and

[1] http://www.w3.org/TR/owl-ref/

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

[3] http://www.w3.org/TR/rdf-sparql-query/

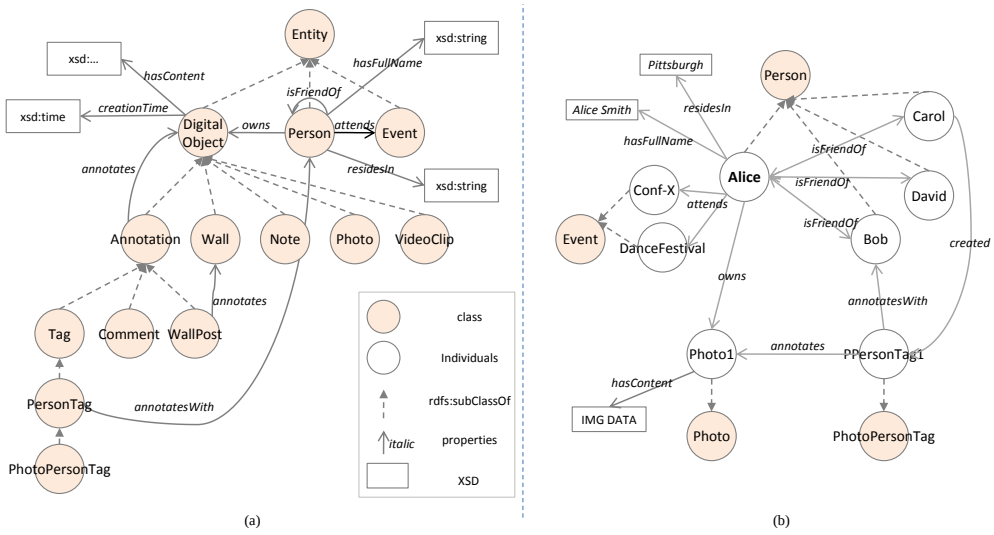

Figure 1. Social Networking systems Ontology (SNO) (a), and its instantiated knowledge (b)

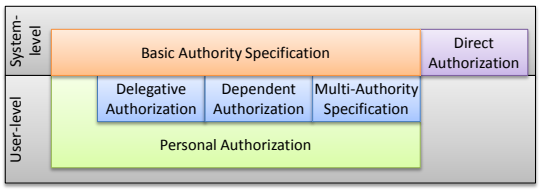

Figure 2. OSNAC Policy Framework

AnnotatesWith(wallPost1, pPersonTag1). Throughout the paper, we use namespace *sn* to refer to SNO concepts and relations.

III. OSNAC: Ontolgy-based Social Network Access Control Model

We propose Ontolgy-based Social Network Access Control (OSNAC), a rule-based access control policy model for SNSs based on Semantic Web standards. The model relies on an ontology such as SNO (introduced in Section II-B) that models the SNS knowledge. OSNAC also uses an access control ontology (ACO, described in Section III-A) to model the policies. We assume a closed-world policy model where authorization rules define valid accesses.

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 authorization* rules regarding protected resources. For more flexible authorizations, users can leverage *delegative* and *dependent* authorization 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 recognizing the authorizations defined at user-level as permissions. In other words, they aggregate user-level authorizations by determining the appropriate authority for protected resources. In contrast, *direct authorization* 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 the various components of the framework in the rest of the section.

A. Policy Expression at Ontology Level

Since knowledge resources are captured in an ontology, the access control policies need to semantically express them using ontology concepts. Our approach is to capture the other components of policies also using ontology. This way, access control policies and protected knowledge resources can naturally be integrated to facilitate an efficient and semantics-rich access control decision. For this purpose, we propose the Access Control Ontology (ACO). We use namespace *ac* to refer to ACO concepts and relations. We consider the relations to be the main protection objects in an ontology-based knowledge base. However, current semantic languages such as OWL do not support statements about the relation instances, which is necessary for specifying authorizations on them. In order to support this, we use a reification approach in ACO. Figure 3 shows the concepts and relations in ACO, that can be categorized as follows.

- ***reified properties*:** A property relates a subject to an object. Class ac:p_property serves as an abstract reification of an SNO property. Properties ac:hasSbj and ac:hasObj relate class ac:p_property to its corresponding subject and object of the property in SNO, respectively. Corre-

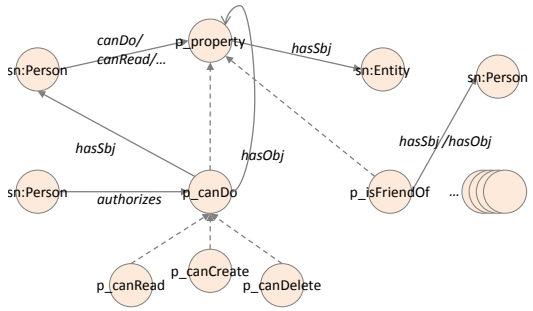

Figure 3. Access Control Ontology (ACO)

sponding to each property sn:x, there exists class ac:p_x, which is a subclass of ac:p_property. Thus, a relation in SNO such as sn:isFriendOf(Alice, Bob) is correspondingly represented in ACO using an instance of class ac:p_isFriendOf; its subject and object are related using relations ac:hasSbj(Alice) and ac:hasObj(Bob), respectively.

- ***permissions*:** A permission specifies an authorized action by a subject on a resource. This is expressed abstractly using property ac:canDo between classes sn:Person and ac:p_property. Actual permissions are defined using ac:canDo descendants, i.e., ac:canRead, ac:canCreate, and ac:canDelete.
- ***permission authorizations*:** A permission can be authorized by different authorities. As in the case of SNO properties, a permission property such as ac:canRead is reified as a class such as ac:p_canRead. In this case, properties ac:hasSbj and ac:hasObj relate the reified permission class to its access subject and resource, respectively. Reified permission classes are used to specify authorizations by individual authorities; property ac:authorizes relates an instance of sn:Person to a reified permission instance such as ac:p_canRead.

Access control policies use the above-mentioned concepts and relations, and follow standard SWRL syntax. However, we deviate slightly from strict syntax for readability purpose. We also introduce a shorthand for expressing reified properties as follows.

$$[?x \leftarrow p(s,o)] \equiv [\text{ac:p\_}p(?x) \wedge \text{ac:hasSbj}(?x,s) \wedge \text{ac:hasObj}(?x,o)]$$

The shorthand expression, on the left-hand side of the equation, asserts variable $?x$ is an instance of the reified property p that relates s and o. The right-hand side of the equation shows this in standard SWRL syntax. In defining rule formats, we refrain from stating the full grammar due to space considerations and mainly provide the rule syntax only for *read* permission, as the syntax for *create* and *delete* authorization rules is similar.

B. Basic Policy Rules

Among the different categories of access control policy rules depicted in Figure 2, personal authorization, direct authorization, and basic authority specification rules constitute the basic functionality of the framework.

1) Personal Authorization: A personal authorization rule expresses a permission granted by an individual user to others. The actual effectiveness of such a rule is determined according to system authority policies. Table I shows the format of a personal authorization rule. The first expression in the antecedent of the rule defines the protection resource $?rsc$ to be of an instance of property p between s and o. The second expression specifies the permission for subject sbj as read operation on the resource. The consequence of the rule states that personal authority u authorizes such a permission. A rule may be extended with more predicates in the antecedent for the purpose of specification of the resource and subject, as shown in the following example.

Example 1: Suppose Alice has specified the following personal authorization rule.

$$\begin{aligned}&\text{sn:isFriendOf}(?sbj,\text{Alice}) \wedge \text{sn:photoTag}(?pTag)\\ &\wedge [?rsc \leftarrow \text{sn:anonatesWith}(?pTag,\text{Alice})]\\ &\wedge [?prm \leftarrow \text{ac:canRead}(?sbj,?rsc)]\\ &\Rightarrow \text{ac:authorizes}(\text{Alice}, ?prm)\end{aligned}$$

Based on the above rule, Alice authorizes her friends to read the photo-tags she has been marked with. The resource is any sn:annotatesWith relation that relates to Alice. The first predicate provides specification for subjects using a relation, i.e., being Alice's friend. The second predicate provide more specification regarding the resource by stating that its subject is a photo-tag. The rest is similar to the format of the rule described above.

2) Direct Authorization: Direct authorization rules allow the system to grant permissions to users without involvement of user authorities. Similar to a personal authorization rule, the antecedent specifies the protection resource. However, ac:canDo descendants are used in the consequence. The format of a direct authorization rule is shown in Table I.

Example 2: The following direct authorizations entitle everyone to read the relations that are defined about them.

$$\begin{aligned}&[?rsc \leftarrow \text{sn:property}(?sbj,?o)] \Rightarrow \text{ac:canRead}(?sbj, ?rsc)\\ &[?rsc \leftarrow \text{sn:property}(?s,?sbj)] \Rightarrow \text{ac:canRead}(?sbj, ?rsc)\end{aligned}$$

The above two rules state that a subject can read a property instance of which she is either the subject or the object, respectively.

3) Basic Authority Specification: For most of the access requests in an SNS, user-level authorizations would be mainly applicable instead of the direct authorizations. In this respect, the role of system-level rules is to determine policy authorities for resources. Authority specification rules use ac:authorizes predicates in the antecedent and ac:canDo descendants in the consequence (Table I). This means that permissions are granted based on personal authorizations. Although an SNS may define a set of authority specification rules customized for its own application semantics, here, we propose a generic, basic authority model for an

Table I
BASIC ACCESS CONTROL POLICY RULES

Type	Rule Format
Personal Authorization	$\ldots \wedge [?rsc \leftarrow$ sn:$p(s,o)] \wedge [?prm \leftarrow$ ac:canRead$(sbj,?rsc)] \Rightarrow$ ac:authorizes$(u, ?prm)$
Direct Authorization	$\ldots \wedge [?rsc \leftarrow$ sn:$p(s,o)] \Rightarrow$ ac:canRead$(?sbj, ?rsc)$
Basic Authority Spec.	$\ldots \wedge [?rsc \leftarrow$ sn:$p(s,o)] \wedge [?prm \leftarrow$ ac:canRead$(?sbj,?rsc)] \wedge$ ac:authorizes$(u, ?prm) \Rightarrow$ ac:canRead$(?sbj, ?rsc)$

ontology-based knowledge base system. We assume there is a unique principal authority for every object (class instance), assigned 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:SNEntity instance. Assuming there is a principal authority for every object, it is safe to consider the same authority to be effective for any property instances associated with that object. Hence, the authority over an object property instance can be determined based on the principal authorities of the related objects. For instance, access to sn:isFriendOf(Alice,Bob) is under the authority of both Alice and Bob. This is truly aligned with potential privacy requirements of both authorities. The authority over a datatype property instance is also the principal authority of the only related object. The basic authority model can be expressed using the following rule.

$$[?rsc \leftarrow \text{sn:property}(?s,?o)] \wedge [?prm \leftarrow \text{ac:canRead}(?sbj,?rsc)] \wedge \text{ac:hasPrincipalAuthority}(?s,?u_1)] \wedge \text{ac:hasPrincipalAuthority}(?o,?u_2)] \wedge \text{ac:authorizes}(?u_1, ?prm) \wedge \text{ac:authorizes}(?u_2, ?prm) \Rightarrow \text{ac:canRead}(?sbj, ?rsc) \quad (1)$$

The above rule grants a (read) permission on a property only if both the principal authorities of the objects associated with the property authorize that permission.

C. Advanced Policy Rules

Advanced policy rules extend user-level policies by allowing distribution of user authorities. This is achieved by allowing authorization predicates both in the antecedent and consequence of rules.

1) Delegative Authorization: Delegation has been shown to be useful in conjunction with access control models (e.g., [7]). In the context of the proposed model, we observe that delegating authority improves the flexibility of policies. Based on a delegative authorization, a user delegates its authority over a specific resource to another user. According to the rule format depicted in Table II user u_1 delegates authorization on a specific permission to authority u_2. In other words, user u_1 respects the authorizations made by user u_2 on the specific permission. One usage of delagative authorizations could be to relax authority on the protected relations. The basic authority specification rule stated in Section III-B3 (rule (1)) requires both the end authorities of a relation to authorize 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 authorizations to respect one another's decisions on the specific permission(s) of choice, without changing system-level rules.

Delegative authorizations are very flexible and secure in terms of delegation power. First, an authority can flexibly customize 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 authorization 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 authorization rule is the sole means of delegation. Third, since delagative authorizations 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.

2) Dependent Authorization: Delegative authorizations can be considered as a part of a larger family of rules, that we call dependent authorizations. According to a dependent authorization rule, one authorization can be inferred based on another authorization. One type of such a rule is shown in Table II, in which the authorization predicate of antecedent and consequence of the rule are only different in their permissions (resource and/or operation). Due to space considerations, we only summarize the list of possible types in Table III, by indicating if the authority, subject, and/or permission differ in the authorization predicates of antecedent and consequence of the rule.

Type 1 dependent authorization is useful in scenarios where the same policy applies to a union of permissions. Personal authorization rule(s) for one of the components of the union can be defined, and authorization for the other components can be defined as dependent authorization to the first one. Types 2 and 3 are useful in similar situations, where the policy is good for a union of subjects, and subjects and permissions, respectively. Type 4 dependent authorization represents delegative authorizations, where one user delegates authority on a specific permission to another user. Types 5, 6, and 7 are a mixture of delegation and types 1, 2, and 3, respectively. The following example briefly shows how various dependent authorization rules enhance the policy expressiveness.

Example 3: In the scenario depicted in Figure 1.b, suppose Alice has a more restrictive policy in mind for dis-

Table II
ADVANCED ACCESS CONTROL POLICY RULES

Type	Rule Format
Delegative Authorization	$\ldots \wedge$ [?$rsc \leftarrow$ sn:$p(s,o)$] $\wedge$ [?$prm \leftarrow$ ac:canRead(sbj,?rsc)] $\wedge$ ac:authorizes(u_2, ?prm) $\Rightarrow$ ac:authorizes(u_1, ?prm)
Dependent Authorization	Type 1 (It has various types; refer to Table III): $\ldots \wedge$ [?$rsc_1 \leftarrow$ sn:$p_1(s_1,o_1)$] $\wedge$ [?$prm_1 \leftarrow$ ac:canRead(sbj,?rsc_1)] $\wedge$ [?$rsc_2 \leftarrow$ sn:$p_2(s_2,o_2)$] $\wedge$ [?$prm_2 \leftarrow$ ac:canRead(sbj,?rsc_2)] $\wedge$ ac:authorizes(u, ?prm_1) $\Rightarrow$ ac:authorizes(u, ?prm_2)
Multi-Authority Spec.	Disjunctive: $R = \{R_i\}$ $R_i = \ldots \wedge$ [?$rsc \leftarrow$ sn:$p(s,o)$] $\wedge$ [?$prm \leftarrow$ ac:canRead(sbj,?rsc)] $\wedge$ ac:authorizes(u_i, ?prm) $\Rightarrow$ ac:authorizes(pa, ?prm)
	Conjunctive: $\ldots \wedge$ [?$rsc \leftarrow$ sn:$p(s,o)$] $\wedge$ [?$prm \leftarrow$ ac:canRead(sbj,?rsc)] $\wedge \bigwedge_{i=1}^{n}$ ac:authorizes(u_i, ?prm) $\Rightarrow$ ac:authorizes(pa, ?prm)

Table III
PARAMETERS OF AC:AUTHORIZES IN DEPENDENT AUTHORIZATION

	Antecedent			Consequence		
Type	Authority	Sbj.	Prm.	Authority	Sbj.	Prm.
1	u	s	p_1	u	s	p_2
2	u	s_1	p	u	s_2	p
3	u	s_1	p_1	u	s_2	p_2
4	u_1	s	p	u_2	s	p
5	u_1	s	p_1	u_2	s	p_2
6	u_1	s_1	p	u_2	s_2	p
7	u_1	s_1	p_1	u_2	s_2	p_2

closing the annotations on Photo1 than for the content of the photo. Using rule (i) she can make sure whenever somebody is authorized for an annotation, she also grants authorization for the photo content. Now, suppose Alice wants to delegate to Bob authorization on the photo content that is restricted among her colleagues and friends-of-friends circle. Rule (ii) performs delegations for colleagues, and Rule (iii) makes sure someone in her friends-of-friends circle gets at least the same authorizations as her colleagues based on the delegation.

Rule (i) (type 1):
[?$rsc_1 \leftarrow$ sn:annotates(?s_1,Photo1)]
$\wedge$ [?$prm_1 \leftarrow$ ac:canRead(?sbj,?rsc_1)]
[?$rsc_2 \leftarrow$ sn:hasContent(Photo1,?c)]
$\wedge$ [?$prm_2 \leftarrow$ ac:canRead(?sbj,?rsc_2)]
$\wedge$ ac:authorizes(Alice, ?prm_1) $\Rightarrow$ ac:authorizes(Alice, ?prm_2)

Rule (ii) (type 4):
[?$rsc \leftarrow$ sn:hasContent(Photo1,?c)] $\wedge$ sn:isColleagueOf(Alice, ?sbj)
$\wedge$ [?$prm \leftarrow$ ac:canRead(?sbj,?rsc)] $\wedge$ ac:authorizes(Bob, ?prm)
$\Rightarrow$ ac:authorizes(Alice, ?prm)

Rule (iii) (type 6):
[?$rsc \leftarrow$ sn:hasContent(Photo1,?c)] $\wedge$ sn:isColleagueOf(Alice, ?sbj_1)
$\wedge$ [?$prm_1 \leftarrow$ ac:canRead(?sbj_1,?rsc)]
$\wedge$ sn:isFriendOf(Alice, ?$friend$) $\wedge$ sn:isFriendOf(?$friend$, ?sbj_2)
$\wedge$ [?$prm_2 \leftarrow$ ac:canRead(?sbj_2,?rsc)]
$\wedge$ ac:authorizes(Bob, ?prm_1) $\Rightarrow$ ac:authorizes(Alice, ?prm_2)

3) Multi-Authority Specification: There are scenarios in SNSs where more authorities are desired to weigh in 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 authorization rules as described in Section III-C1 to enable a *disjunctive multi-authority*. Such a multi-authority is disjunctive in the sense that a permission authorization by any corresponding authority in the set is a sufficient condition for that permission to be considered authorized by the principal authority. Alternatively, a principal authority may create a *conjunctive multi-authority*, in which every involved authority is required to authorize a permission in order that it would be considered authorized by the principal authority. Table II shows a rule set and a single rule that establishes disjunctive and conjunctive multi-authority, respectively, where principal authority pa shares the authority with users u_1, u_2, ..., and u_n.

D. 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/create/delete).

Definition 1 (Access Authorization): 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

$$[?rsc \leftarrow p(s,o)] \wedge \text{ac:canDo}(sbj, ?rsc)$$

where predicate ac:canDo 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, access authorization per such relations is not efficient, and needs modification of the retrieval engine. Alternatively, we augment a query with access check primitives and evaluate the access-augmented query using the retrieval engine.

Definition 2 (Query Access Authorization): Let $\langle sbj, Q \rangle$ be a query access request, where $Q_W = \bigwedge_{i=1}^{n} \text{sn:}p_i(s_i,o_i)$, represents a conjunctive *WHERE* clause (conjunction of a set of predicates). A retrieval engine automatically enforces the access control policy and retrieves the authorized result by evaluating $Q'_W = \bigwedge_{i=1}^{n} \{\text{sn:}p_i(s_i,o_i) \wedge [?rsc_i \leftarrow \text{sn:}p_i(s_i,o_i)] \wedge$

ac:canRead(sbj, $?rsc_i$)}.

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. The access control information is seamlessly captured in the ontology by using an ontology reasoner on the authorization rules.

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)
∧ sn:residesIn(?x, Pittsburgh)
∧ sn:hasFullname(?x, ?fname) }

The access-augmented *WHERE* clause is as follows.

sn:friendOf(Alice,?x) ∧
∧ [?rsc1 ← sn:friendOf(Alice,?x)] ∧ ac:canRead(Bob, ?rsc1)
∧ sn:residesIn(?x, Pittsburgh)
∧ [?rsc2 ← sn:residesIn(?x, Pittsburgh)] ∧ ac:canRead(Bob, ?rsc2)
∧ sn:hasFullname(?x, ?fname)
∧ [?rsc3 ← hasFullname(?x, ?fname)] ∧ ac:canRead(Bob, ?rsc3)

By 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 authorized information according to the access control policies.

E. Supporting Negative Authorization

The OSNAC policy model relies on (positive) authorizations. If the system cannot resolve a corresponding positive system authorization (i.e., descendants of ac:canDo) for an access request, then the request is denied. Although positive authorizations can be used to express security policies in general, it is sometimes desirable to express intended policies using a mixture of positive and negative authorizations. 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 authorizations. To be more specific, if an authorization cannot be inferred its negation cannot be inferred either. Our proposed workaround to support negative authorization is to introduce separate predicates for negative authorizations 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 III-A, as follows. Analogous to property ac:canDo and its descendants, we define property ac:cannotDo and its corresponding descendants (i.e., ac:cannotRead, ac:cannotInsert, and ac:cannotDelete), that represent negative permissions. There exist also reified versions of this negative permissions (e.g., ac:p_cannotRead). Analogous to a positive authorization, a negative authorization can be expressed by a user using property ac:authorizes between the corresponding sn:Person instance and a reified negative permission instance. Thus, a personal negative authorization can be expressed as follows.

$\ldots \wedge [?rsc \leftarrow \text{sn:}p(s,o)] \wedge [?prm \leftarrow \text{ac:cannotRead}(sbj, ?rsc)]$
$\Rightarrow \text{ac:authorizes}(u, ?prm)$

The last step to enable negative authorization is to specify corresponding basic authority specification rules, in a manner similar to what is described in Section III-B3 but for negative permissions. Our approach satisfies the need for the common use of negative authorizations. Users will be able to express negative exceptions to positive authorizations, and even use advanced policy rules to distribute their power to specify such denials. Note that there is no need for fine-grained conflict resolution, as conflicts are taken care of at the retrieval step. We follow a *denial-takes-precedence* approach, when both ac:canDo and ac:cannotDo are resolved for a specific access.

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

$$Q'_W = \bigwedge_{i=1}^{n} \begin{pmatrix} \text{sn:}p_i(s_i,o_i) \\ \wedge [?rsc_i \leftarrow \text{sn:}p_i(s_i,o_i)] \wedge \text{ac:canRead}(sbj, ?rsc_i) \\ \wedge \text{OPTIONAL}\{\ ?sbj_i\ \text{ac:cannotRead}\ ?rsc_i\} \\ \wedge \text{FILTER}(!\text{bound}(?sbj_i)) \end{pmatrix}$$

IV. Implementation

A. 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[4]. We leverage Jena's TDB for persistent storage of social network (SNO) and access control (ACO) ontologies. In an initialization phase, based on the SNS knowledge captured in SNO, ACO is populated with the corresponding reified properties and permissions as described in Section III-A.

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 III. However, since SWRL is

[4]http://jena.sourceforge.net/

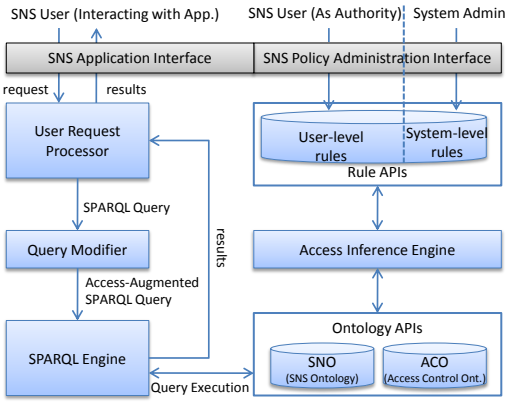

Figure 4. Architecture of the Prototype Implementation

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 III-D). The modified query is then sent to the *SPARQL Engine*. The SPARQL Engine 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:canDo) based on the knowledge stored in the ontologies and according to the access control policy rules. Finally, the authorized query results are returned to the *User Request Processor*.

B. 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 authorized information that is expected according to the sample access control policy rules.

We also developed a data generator that randomly populates an SNO ontology. Table IV shows the performance results of the prototype access engine based on the following input parameters: the number of users, friendship links, photos, and maximum number of people been tagged in a photo. Since the inference engine that Jena provides only works in memory, we were not able to run the experiment for very large ontologies. Our experiments show that the first access control inference is relatively expensive. However, subsequent access checks are performed almost instantaneously. This is because in the first round the inference model caches some of the inferred axioms, which enhances performance for subsequent inference. In fact, the first access check can be considered as part of the initialization phase, which can be triggered with a dummy access request.

Table IV
PROTOTYPE PERFORMANCE RESULTS

Data Generation Parameters				Access Check Times (s)		
#Person	#Photo	#Tag/Photo	#isFriendOf	Init.	First	Subsq.
25	5	5	60	2.9	0.4	0.004
50	25	5	150	4.8	33.0	0.004
75	50	10	175	5.9	163.1	0.005
100	75	10	275	10.4	1816.6	0.006
125	125	10	500	16.5	4584.3	0.006

V. RELATED WORK

Access control research in social network area is still in its early stages. Early 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. FOAF-Realm[5] is one of the earliest approaches that tries to quantify the *knows* relations in the context of FOAF (Friend Of A Friend) ontology. It support rules that control friends' access to resources in a social network by stating the maximum distance and minimal friendship level (a trust metric) [1][2]. Carminati et al. propose a conceptually-similar but more complete trust-based access control model [3]. Villegas et al. propose to use a slightly different trust measure by automatically classifying nodes in zones [4]. 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 by Carminati et al. [5], which also leverages OWL and SWRL. They define three type of policies, namely, access control policy, filtering policy, and admin policy. Access control policies are positive authorization rules; filtering policies can limit someone's access to information by herself (i.e., not conceptually a security issue); and admin policies can express who are authorized 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. propose a rule-based access

[5]http://www.foafrealm.org/xfoaf/0.1/

control model for semantic networks, based on a constrained first order logic [6]. 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 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 [8].

In the area of Semantic Web, several works have proposed access control solutions for RDF stores, although not in the context of SNSs. Reddivari et al. propose a rule-based model and architecture, called RAP [9]. 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 very efficient access control architecture. for a given query to the RDF store, the result set is retrieved first; Then for every triple in the result, the access inference is performed separately, and it will be included in the final result set if it is granted. Our query augmentation approach seems to perform more efficiently, as it seamlessly uses access primitive predicates in the query. This way, excessive overhead of access checks will be avoided by the query engine itself. There are other approaches to access control on RDF stores that are comparatively less grounded [10][11][12], or support a specific policy such as multi-level security [13].

VI. Conclusions and Future Work

In this paper, we have proposed OSNAC, an ontology-based access control model based on Semantic Web standards that empowers the individual users of a social networking system 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 have also implemented a framework prototype of the proposed model in order to show the applicability of our approach.

Although OSNAC provides powerful access control features to the users of the SNSs, even savvy users of such systems should 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. Our experimental results also raised some concerns about performance of ontology and rule reasoning. We will explore ways to improve this aspect in our implementation, and theoretically analyze the complexities introduced by ontological data and each policy component.

Acknowledgements. This research has been supported by the US National Science Foundation award IIS-0545912. We thank the anonymous reviewers for their helpful comments.

References

[1] S. R. Kruk, "FOAF-Realm: control oyur friends access to the resource," in *Proc. FOAF Workshop*, 2004.

[2] S. R. Kruk, S. Grzonkowski, A. Gzella, T. Woroniecki, and H. C. Choi, "D-FOAF: Distributed identity management with access rights delegation," in *Proc. 1st Asian Semantic Web Conference*. Springer, 2006, pp. 140–154.

[3] B. Carminati, E. Ferrari, and A. Perego, "Rule-based access control for social networks," in *Proc. OTM 2006 Workshops*, ser. LNCS, vol. 4278. Springer, Oct 2006, pp. 1734–1744.

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

[5] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "A semantic web based framework for social network access control," in *Proc. 14th ACM Symposium on Access Control Models and Technologies*. ACM, 2009, pp. 177–186.

[6] T. Ryutov, T. Kichkaylo, and R. Neches, "Access control policies for semantic networks," July 2009, pp. 150–157.

[7] E. Barka and R. S. Sandhu, "Framework for role-based delegation models," in *Proc. 16th Annual Computer Security Applications Conference*. IEEE Computer Society, Dec 2000, pp. 168–176.

[8] M. Shehab, A. Squicciarini, and G.-J. Ahn, "Beyond user-to-user access control for online social networks," in *ICICS '08: Proceedings of the 10th International Conference on Information and Communications Security*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 174–189.

[9] P. Reddivari, T. Finin, and A. Joshi, "Policy-based access control for an RDF store," in *Workshop on Policy Management for the Web*, 2005, pp. 78–81.

[10] S. Dietzold and S. Auer, "Access control on RDF triple stores from a semantic wiki perspective," in *Scripting for the Semantic Web Workshop at 3rd European Semantic Web Conference (ESWC)*, June 2006.

[11] A. Dersingh, R. Liscano, A. Jost, J. Finnson, and R. Senthilnathan, "Utilizing semantic knowledge for access control in pervasive and ubiquitous systems," *Mobile Networks and Applications*.

[12] M. Liu, D. Xie, P. Li, X. Zhang, and C. Tang, "Semantic access control for web services," vol. 2, April 2009, pp. 55–58.

[13] A. Jain and C. Farkas, "Secure resource description framework: an access control model," in *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2006, pp. 121–129.