A Semantic Web Based Framework for Social Network Access Control

Barbara Carminati
University of Insubria
Via Mazzini, 5
Varese - Italy
barbara.carminati@uninsubria.it

Elena Ferrari
University of Insubria
Via Mazzini, 5
Varese - Italy
elena.ferrari@uninsubria.it

Raymond Heatherly
University of Texas at Dallas
800 W. Campbell Road
Richardson, TX 75080 U.S.A.
rdh061000@utdallas.edu

Murat Kantarcioglu
University of Texas at Dallas
800 W. Campbell Road
Richardson, TX 75080 U.S.A.
muratk@utdallas.edu

Bhavani Thurainsingham
University of Texas at Dallas
800 W. Campbell Road
Richardson, TX 75080 U.S.A.
bxt043000@utdallas.edu

ABSTRACT

The existence of on-line social networks that include person specific information creates interesting opportunities for various applications ranging from marketing to community organization. On the other hand, security and privacy concerns need to be addressed for creating such applications. Improving social network access control systems appears as the first step toward addressing the existing security and privacy concerns related to on-line social networks. To address some of the current limitations, we propose an extensible fine grained access control model based on semantic web tools. In addition, we propose authorization, admin and filtering policies that depend on trust relationships among various users, and are modeled using OWL and SWRL. Besides describing the model, we present the architecture of the framework in its support.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access Controls; I.2.4 [**Knowledge Representation**]: Semantic Networks

General Terms

Design, Security, Theory

Keywords

Social Networks, Semantic Web, Access Control

1. INTRODUCTION

On-line Social Networks (OSNs) are platforms that allow people to publish details about themselves and to connect to other members of the network through links. Recently, the popularity of OSNs is increasing significantly. For example, Facebook now claims to have more than a hundred million active users.[1] The existence of OSNs that include person-specific information creates both interesting opportunities and challenges. For example, social network data could be used for marketing products to the right customers. At the same time, security and privacy concerns can prevent such efforts in practice [2]. Improving the OSN access control systems appears as the first step toward addressing the existing security and privacy concerns related on-line social networks. However, most of current OSNs implement very basic access control systems, by simply making a user able to decide which personal information are accessible by other members by marking a given item as public, private, or accessible by their direct contacts. In order to give more flexibility, some online social networks enforce variants of these settings, but the principle is the same. For instance, besides the basic settings, Bebo (`http://bebo.com`), Facebook (`http://facebook.com`), and Multiply (`http://multiply.com`) support the option "selected friends"; Last.fm (`http://last.fm`) the option "neighbors" (i.e., the set of users having musical preferences and tastes similar to mine); Facebook, Friendster (`http://friendster.com`), and Orkut (`http://www.orkut.com`) the option "friends of friends"; Xing (`http://xing.com`) the options "contacts of my contacts" (2nd degree contacts), and "3rd" and "4th degree contacts". It is important to note that all these approaches have the advantage of being easy to be implemented, but they lack flexibility. In fact, the available protection settings do not allow users to easily specify their access control requirements, in that they are either too restrictive or too loose. Furthermore, existing solutions are platform-specific and they are hard to be implemented for various different online social networks.

To address some of these limitations, we propose an extensible, fine-grained OSN access control model based on semantic web technologies. Our main idea is to encode social network-related information by means of an ontology. In particular, we suggest to model the following five important aspects of OSNs using semantic web ontologies: (1) user's

SACMAT'09, June 3–5, 2009, Stresa, Italy.
Copyright 2009 ACM 978-1-60558-537-6/09/06 ...$5.00.

[1] http://www.facebook.com/press/info.php?statistics

profiles, (2) relationships among users (e.g., Bob is Alice's close friend), (3) resources (e.g., online photo albums), (4) relationships between users and resources (e.g., Bob is the owner of the photo album), (5) actions (e.g., post a message on someone's wall). By constructing such an ontology, we model the Social Network Knowledge Base (SNKB). The main advantage for using an ontology for modeling OSN data is that relationships among many different social network concepts can be naturally represented using OWL. Furthermore, by using reasoning, many inferences about such relationships could be done automatically. Our access control enforcement mechanism is then implemented by exploiting this knowledge. In particular, the idea is to define security policies as rules (see Section 4), whose antecedents state conditions on SNKB, and consequents specify the authorized actions. In particular, we propose to encode the authorizations implied by security policies by means of an ontology, obtaining the Security Authorization Knowledge Base (SAKB). Thus, security policies have to be translated as rules whose antecedents and consequents are expressed on the ontology. To achieve this goal, we use the Semantic Web Rule Language (SWRL) [9]. As consequence, the access control policies can be enforced by simply querying the authorizations, that is, the SAKB. The query can be easily directly implemented by the ontology reasoner by means of instance checking operations, or can be performed by a SPARQL query, if the ontology is serialized in RDF. In this paper, we focus on how to model such a fine-grained social network access control system using semantic web technologies. We also assume that a centralized reference monitor hosted by the social network manager will enforce the required policies. Since our proposed approach depends on extensible ontologies, it could be easily adapted to various online social networks by modifying the ontologies in our SNKB. Furthermore, as we discuss in details later in the paper, semantic web tools allow us to define more fine grained access control policies than the ones provided by current OSNs.

The paper is organized as follows. In Section 2, we provide a brief discussion of current security and privacy research related to online social networks. In Section 3, we discuss how to model social networks using semantic web technologies. In Section 4, we introduce a high level overview of the security policies we support in our framework. In addition to access control policies, we state filtering policies that allow a user (or one of her supervisors) to customize the content she accesses. We also introduce admin policies, stating who is authorized to specify access control and filtering policies. In Section 5, we introduce the authorization ontology and the SWRL rule encoding of security policies. In Section 6, we discuss how security policies could be enforced. In Section 7, we give an overview of the framework that integrates the previous components. Finally, we conclude the paper in Section 8.

2. RELATED WORK

Past research on OSN security has mainly focused on privacy-preserving techniques to allow statistical analysis on social network data without compromising OSN members' privacy (see [5] for a survey on this topic). In contrast, access control for OSNs is a relatively new research area. As far as we are aware, the only other proposals of an access control mechanism for online social networks are [10], [1] and [4]. The D-FOAF system [10] is primarily a Friend of a Friend (FOAF) ontology-based distributed identity management system for social networks, where access rights and trust delegation management are provided as additional services. In D-FOAF, relationships are associated with a trust level, which denotes the level of *friendship* existing between the users participating in a given relationship. Although [10] discusses only generic relationships, corresponding to the ones modeled by the foaf:knows RDF property in the FOAF vocabulary [3], another D-FOAF-related paper [6] considers also the case of multiple relationship types. As far as access rights are concerned, they denote authorized users in terms of the minimum trust level and maximum length of the paths connecting the requester to the resource owner. In [1], authors adopt a multi-level security approach, where trust is the only parameter used to determine the security level of both users and resources. In [4], a semi-decentralized discretionary access control model and a related enforcement mechanism for controlled sharing of information in OSNs is presented. The model allows the specification of access rules for online resources, where authorized users are denoted in terms of the relationship type, depth, and trust level existing between nodes in the network.

Compared to existing approaches, we use semantic web technologies to represent much richer forms of relationships among users, resources and actions. For example, we are able to represent access control rules that leverage relationship hierarchies and by using OWL reasoning tools, we can infer a "close friend" is also a "friend" and anything that is accessible by friend could be also accessible by a "close friend". In addition, our proposed solution could be easily adapted for very different online social networks by modifying the underlying SNKB. A further discussion on the differences between the proposed framework and the access control mechanism in [4] is provided in Section 4.

Semantic web technologies have been recently used for developing various policy and access control languages for domains different from OSNs. For example, in [12], authors compare various policy languages for distributed agent based systems that define authorization and obligation policies. In [8], OWL is used to express role-based access control policies. In [15], authors propose a semantic access control model that separates the authorization and access control management responsibilities to provide solutions for distributed and dynamic systems with heterogeneous security requirements. None of these previous work deals with the access control issues related to online social networks. Among the existing work, [7] is the most similar to our proposal. Compared to [7], we provide a much richer OWL ontology for modeling various aspects of online social networks. In addition, we propose authorization, admin and filtering policies that depend on trust relationships among various users.

3. MODELING SOCIAL NETWORKS USING SEMANTIC WEB TECHNOLOGIES

Recently, semantic web technologies such as Resource Description Framework (RDF) and the Web Ontology Language (OWL) have been used for modeling social network data [11]. Although our goal in this paper is not to propose new semantic approaches for modeling online social network data, we would like to give a brief overview of current approaches for the sake of completeness by pointing out also other social network information that could be modeled by semantic technologies. In our discussion, we will use Facebook as a running example. At the same time, we would like to stress that our discussion could be easily extended to other social networking frameworks. In general, we identify five categories of social network data that could be modeled by semantic technologies. These are: (1) personal information; (2) personal relationships; (3) social network resources; (4) relationships between users and resources; (5) actions that can be performed in a social network. In the following, we discuss how these social network data can be represented.

3.1 Modeling Personal Information

Some of the personal information provided on OSNs such as Facebook can be modeled by using the Friend-of-a-Friend ontology (FOAF) [3]. FOAF is an OWL-based format for representing personal information and an individual's social network. FOAF provides various classes and properties to describe social network data such as basic personal information, online account, projects, groups, documents and images. However, these basic mechanisms are not enough to capture all the available information. For example, there is no FOAF construct to capture the meaning for *lookingFor* (e.g., John Smith is looking for friendship). Thanks to the extensibility of the RDF/OWL language, this is easily solvable. For example, consider the following case where we capture the information related to an individual with Facebook Profile Id 999999 using a new Facebook ontology written in the RDF/OWL language.[2] In this example, we assume that "fb" ontology has a property name *lookingFor* to capture the required information.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix fb: <http://example.org/facebook> .
<http://www.facebook.com/profile.php?id=999999999>
foaf:name "John Smith" .
<http://www.facebook.com/profile.php?id=999999999>
fb:lookingFor "Friendship" .
```

As the example suggests, existing ontologies such as FOAF could be easily extended to capture personal information available on online social networks.

3.2 Modeling Personal Relationships

Currently, online social networks do not support fine-grained definitions of relationships. For instance, Facebook allows you to specify whether you attended school or work with a friend, but offers no way to express what that truly means, that is, the strength of the relationship. It is this fine-grained structure that we wish to capture. Mika [11] proposes a reification based model for capturing relationship strength. Instead, to comply with W3C specifications [14], we adopt the use of the n-ary relation pattern rather than use simple statement reification, which is a violation of the specification [13]. If we were to violate the specification, then relationships would be modeled using a series of four RDF statements to create an identifier for the relationship. Unfortunately, as a result of that, SWRL would be unable to understand these relationships. We believe that using a specification-recommended pattern and retaining the ability to use SWRL to do inference on relationships is the best solution.

For the reasons stated above, we choose to model personal relationships using n-ary relation pattern. To comply with n-ary relation specification [13], we define a *FriendshipRelation* class which has subclasses that denote a general strength of friendship. The root *FriendshipRelation* class implies an unspecific friendship while the three subclasses, *Family*, *CloseFriend*, and *DistantFriend*, give an indicator of the closeness between people. The *CloseFriend* subclass has a further extension: *BestFriend*.

This basic structure allows us to easily mimic the existing structure of Facebook relationship types. However, as mentioned previously, these relationship types have no predefined meanings. In order to begin to quantify the meaning of relationship assignments, each instance of *FriendshipRelation* has a data property *TrustValue*. This represents the level of trust that the initiator has with the friend.

As an example suppose that an individual (e.g., John Smith) defines a relationship with a colleague (e.g., Jane Doe). This creates an instance of the *FriendshipRelation* class with the *TrustValue* data property, which represents the level of trust between the initiator and his friend. The instance also has an object property that links it to the instance of the friend. This instance of the *FriendshipRelation* class is then tied back to John Smith through the use of the *Friendship* object property.

It is important to note that any (uni-directional) relationship in the social network is a single instance of the *FriendshipRelation* class. Thus, to model the standard bidirectional nature of social network relations, we need two instances of this class. However, the simple logical inference that if B is a friend of A, then A is a friend of B can not be implemented by SWRL, in that this would imply to create a new instance of the Friendship class. Unfortunately, this is outside the realm of SWRL's capability. So this must be taken care of outside of the SWRL framework by an external application. It is also important to note that the *TrustValue* property of relationships is a value that is computed automatically outside the OWL/SWRL component of the social network. This value is used to do various inference tasks further in the network. At the most basic level, where the *TrustValue* is a static number based on the friendship type, this is a trivial component. We assume that there will be a more complicated formula used in calculating the *TrustValue* that may be beyond the bounds of the built-in mathematical operators of SWRL.

We experience a similar difficulty with indirect relationships. To define an inferred relationship, we would once again need to create a new instance of *FriendshipRelation*. We can, however, create these indirect relationships similar to how we maintain symmetry of relationships, detailed above. The only difference in the indirect relationship is that instead of creating an instance of the class *FriendshipRelation*, we create an instance of a separate class, *InferredRelation*, which has no detailed subclasses, yet is otherwise identical to the FriendshipRelation base class.

3.3 Modeling Resources

A typical OSN provides some resources such as Albums or Walls to share information among individuals. Clearly RDF/OWL could be used to capture the fact that Albums are composed of pictures and each picture may have multiple people in it. In our framework, we model resources as a class, beginning with a generic *Resource* class. As subclasses

[2] We use Turtle notation for representing OWL.

to this, we can have, for example, *PhotoAlbum*, *Photo*, and *Message*. Each of these has specific, unique properties and relationships. For instance, *PhotoAlbum* has a name and a description as data properties and has an object property called *containsPhoto* that links it to instances of *Photo*. These have a name, a caption, and a path to the stored location of the file. Messages have a sender, a receiver, a subject, a message, and a time stamp. We can also create a subclass of Messages called *WallMessage* which is similar to Messages in that it has the same data properties, but it has additional restrictions such as that a *WallMessage* may only be sent to a single individual.

3.4 Modeling User/Resource Relationships

Current applications such as Facebook assume that the only relationship between users and resources is the ownership. However, from an access control point of view this is not enough. Let us consider, for example, a photograph that contains both John Smith and Jane Doe. Jane took the picture and posted it on the social network. Traditionally, Jane would be the sole administrator of that resource. Since the photo contains the image of John (we say that John is tagged to the photo), in our model John may have some determination as to which individuals can see the photo.

To model something like a photo album, we can use two classes. The first is a simple *Photo* class that simply has an optional name and caption of the photo and a required path to the location of the file. A photo is then linked to each person that is listed as being in the photo. A *PhotoAlbum* has a name and a description. *PhotoAlbum* and *Photo* are linked using the *containsPhoto* relationship. The individual owner – the person who uploaded the photos – is indicated by the *ownsAlbum* relationship. Similarly, we can represent other relationships between users and resources.

3.5 Modeling Actions

In a social network, actions are the basis of user participation. According to the proposed representation an action is defined as object property that relates users, resources, and actions. Moreover, we model hierarchies for actions by means of subproperty. Take, for instance, three generic actions: *Read, Write, Delete*. We define a hierarchy in which *Delete* is a subtype of *Write* which is, itself, a subtype of *Read*. In a non-hierarchical model, if John Smith was able to read, write, and delete a photo, then we would need three authorizations to represent this property. However, as we have defined the hierarchy, with only the authorizations of <“John Smith”, Delete, Photo1>, John Smith has all three properties allowed.

We can also extend traditional access restrictions to take advantage of social networking extensions. For instance, the action *Post* can be defined as a subtype of Write. So, let us say that we define the actions *Write* to mean that an individual can send a private message to another individual, and that the action *Post* means that an individual can post a message to another's Wall so that any of their friends can see it. Then allowing a user the *Post* action would allow them to see the friends wall, send them a private message, and write on their wall, but she could not delete anything.

3.6 Running Example

In the remainder of this paper, we will use the small network shown in Figure 1 to illustrate our access control mechanism. Our running example has four individuals: Alice, Bob, Charlie, and David. Alice, Bob, and Charlie form a clique with different strengths of friendship connecting them. David is a friend only of Bob via the default Friendship type. There is also a *PhotoAlbum* that was uploaded by Alice that contains a single photo that is a picture of Charlie.

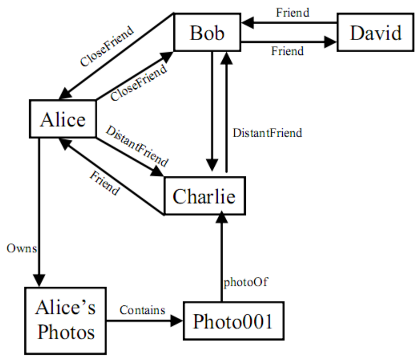

Figure 1: A portion of an OSN

4. SECURITY POLICIES FOR OSNS

As evinced by recent work on social network security, protecting resources in social networks requires us to revise traditional access control models and mechanisms. However, the approaches proposed so far have focused only on access control policies, that is, on the problem of regulating the access to OSN resources. We think that this is not enough, in that the complexity of the social network scenario requires the definition of further security policies, besides standard access control policies. In this section, we outline the security policies our framework supports.

Access Control policies. The framework supports access control policies to regulate how resources can be accessed by OSN participants. In particular, the supported access control policies are defined on the basis of our previous work [4]. Here, authorized users are denoted in terms of the type, depth, and trust level of the relationships existing between nodes in the network. For instance, an access control policy can state that the only OSN participants authorized to access a given resource are those with a direct or indirect friendship relationship with the resource owner, provided that this relationship has a given trust value. However, the access control policies supported by the proposed framework have some notable improvements w.r.t. those presented in [4]. These improvements are mainly due to the fact that our access control policies are defined according to the SNKB described in Section 3. This means that the object, subject and privilege of an access control policy are defined exploiting the semantic modeling of resources, users and actions. In particular, as it will be explained in Section 5, access control policies are defined as rules over ontologies representing the concepts introduced in Section 3. Thus rather than access control policies specified over each single participant and resource of a OSN, we are able to specify access control policies directly on the OSN semantic concepts. Indeed, it is possible to specify a generic access control policy stating that the photos can be accessed only by friends, by simply specifying the *Photo* class as a protected object. As such, the access control policy will be applied to all instances of the *Photo* class, i.e., to all photos, thus greatly simplifying policy administration. Specifying access control policies over semantic concepts has another benefit in that it is possible to exploit the hierarchy defined over the concepts to automatically propagate access control policies. For example, with respect to resources, if *Photo* has been defined with some subclasses, say *PrivatePhoto* and *HolidaysPhoto*, the previous access control policy can be automatically applied to all the instances belonging to any subclass of *Photo*. Access control policies can be also propagated along other dimensions, that is, according to hierarchies specified in the ontologies of other OSN concepts (e.g., ontologies for relationship types and actions). For example, in case the supported relationship ontology defines an hierarchy for the friendship relationship, the previous access control policy is propagated to all OSN participants with which the resource owner has any kind of friendship relationship. A similar propagation arises if the action ontology defines an hierarchy of actions. Note that also in [4], authorized subjects are defined in terms of user relationships rather than by listing specific instances (i.e., person ids). However, in that work policy propagation is not possible, since no hierarchies are defined over resources, relationships and actions. Moreover, the semantic modeling we propose in this paper makes us able to specify authorized users not only in terms of the relationships they should have with the resource owner (as in [4]), but also in terms of the relationships they should have with the resource. Thus, for example it is possible to specify an access control policy stating that all OSN participants that are tagged to a photo are authorized to access that photo. The only way to specify this access control policy in [4] as well as in all the other existing models for OSNs is to explicitly specify a different access control policy for each OSN participant tagged to the photo.

Filtering policies. In a OSN, users can publish information of very heterogeneous content, ranging from family photos to adult-oriented contents In this sense, the access control issues arising in OSNs are similar to those we have in the web, where the availability of inappropriate information could be harmful for some users (for example, young people). To protect users from inappropriate or unwanted contents, we introduce *filtering policies*, by which it is possible to specify which data has to be filtered out when a given user browses the social network pages. By means of a filtering policy, it is, for example, possible to state that from OSN pages fetched by user Alice, all videos that have not been published by Alice's direct friends have to be removed. Similar to access control policies, filtering policies are defined as rules over ontologies representing the concepts introduced in Section 3 (see Section 5). This implies that policy propagation is possible also in case of filtering policies. Another relevant aspect of filtering policies is related to the user that specifies the policy (i.e., the grantor). Indeed, in our framework, a filtering policy can be specified in two different ways. According to the first one, a filtering policy is specified by a user to state which information she prefers not to access, i.e., which data has to be filtered out from OSN pages fetched by her. Thus, in this case the grantor and the user to which the policy applies, i.e., the target user, are the same. These policies state user preferences w.r.t. the contents one wants to access and for that reason are called *filtering preferences.* However, we also supports the specification of filtering policies where the target user and the grantor are different. This kind of filtering policies makes the grantor able to specify how the SN pages fetched by target users have to be filtered. By means of these filtering policies, a grantor can *supervise* the content a target user can access. In this case, we refer to the filtering policy as *supervised filtering policy.* This represents an extremely useful feature in open environments like OSNs. For example, a parent can specify a supervised filtering policy stating that her children do not have to access those videos published by users that are not trusted by the parent herself. As it will be more clear later on, semantic technologies greatly facilitate the specification of this kind of policies.

It is worth noticing that both filtering preferences and supervised filtering policies can not be enforced by simply supporting negative access control policies, that is, policies avoiding access to resources. This is due to the fact that access control policies and filtering policies have totally different semantics. Indeed, an access control policy is specified by the resource owner to state who is authorized or denied to access her resources. Rather, a filtering policy is specified by a supervisor for a target user or by the target user herself, to specify how resources have to be filter out when she fetches an OSN page. Note that, according to the proposed semantics, this filtering takes place even in the case the target user is authorized to access the resource, that is, even if she satisfies the access control policies specified by the resource owner.

Admin policies. Introducing access control and filtering policies in a multi-user environment like OSNs requires to determine who is authorized to specify policies and for which target users and objects. To address this issue we introduce *admin policies*, that make the Security Administrator (SA) of the social network able to state who is authorized to specify access control and filtering policies. Admin policies have to be flexible enough to model some obvious admin strategies that are common to traditional scenarios (e.g., the resource owner is authorized to specify access control policies for her resources) as well as more complex strategies, according to the security and privacy guidelines adopted by the OSN. For instance, the SA could specify an admin policy stating that users tagged to a given resource are authorized to specify access control policies for that resource. Note that, as previously pointed out, the ontology modeling the relationships between users and resources described in Section 3 is extremely useful in the specification of such admin policies. Other kinds of admin policies are those related to filtering policies. For instance, by means of an admin policy, a SA could authorize parents to define supervised filtering policies for their young children. This admin policy can be defined by stating that if a user U1 has a relationship of type ParentOf with a user U2, which has age less than 16 (i.e., with the property age less than 16), then U1 can state supervised filtering policies where the target user is U1. The SA could further refine this admin policy to specify that the parents can state supervised filtering policies for their young children only for video resources. This would modify the previous admin policy by limiting the scope of the supervised filtering policy the parents are authorized to specify.

5. SECURITY POLICY SPECIFICATION

A policy language defines security policies according to three main components: a *subject specification* aiming to specify the entity to which a security policy applies (e.g., users, processes), an *object specification* to identify the resources to which the policy refers to (e.g., files, HW resources, relational tables), and an *action specification*, specifying the action (e.g., read, write execute, admin) that subjects can exercise on objects. Moreover, to make easier the task of policy evaluation, policies are enforced through a set of *authorizations*, stating for each subject the rights she has on the protected resources. We encode security policies by means of rules. In general, a rule consists of two formulae and an implication operator, with the obvious meaning that if the first formula, called the antecedent, holds then the second formula, called the consequent, must also hold. Thus, we encode each security policy as a *security rule*, that is, a rule whose antecedent represents the conditions stated in the policy subject and object specifications, and the consequent represents the entailed authorizations. Note that since the framework supports different types of security policies, the security rules could entail different types of authorizations. In particular, if the antecedent of a security rule encodes an access control or admin policy, the consequent denotes the entailed access control or admin authorizations. In contrast, if the rule's antecedent encodes a filtering policy (either a filtering preference or a supervised filtering policy), the consequent entails *prohibitions* rather than authorizations, since this policy limits access to resources.

We adopt SWRL to encode security rules. SWRL has been introduced to extend the axioms provided by OWL to also support rules. In SWRL, the antecedent, called the body, and the consequent, called the head, are defined in terms of OWL classes, properties and individuals. More precisely, they are modeled as positive conjunctions of *atoms*. Atoms can be of the form: (1) C(x), where C is an OWL description or data range; (2) P(x,y), where P is an OWL property and x and y could be variables, OWL individuals or OWL data values; (3) sameAs(x,y); (4) differentFrom(x,y); (5) builtIn(r,x,...), where r is a built-in predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate. More precisely, an atom C(x) holds if x is an instance of the class description or data range C, an atom P(x,y) holds if x is related to y by property P, an atom sameAs(x,y) holds if x is interpreted as the same object as y, an atom differentFrom(x,y) holds if x and y are interpreted as different objects, and builtIn(r,x,...) holds if the built-in relation r holds on the interpretations of the arguments.

Exploiting SWRL to specify security rules implies that authorizations and prohibitions must be represented in some ontology, thus to be encoded as a SWRL head. For this reason, before presenting the encoding of a security policy, we first introduce an ontology to model authorizations and prohibitions. We refer to the knowledge base derived by this ontology as Security Authorization Knowledge Base (SAKB).

5.1 Authorizations and Prohibitions

Since the framework supports three different types of security policies, it has to manage three different types of authorizations, namely access control authorizations, admin authorizations, and prohibitions. In the following, we introduce the proposed ontology for their representations. However, it is relevant to notice that this ontology is strictly related to the ontologies supported by the OSN (see Section 3), in that it defines authorizations/prohibitions on the basis of the supported actions and resources. As such, the following does not intend to be the standard ontology for SAKBs, rather it is the one that we adopt in our framework, based on the semantic modeling presented in Section 3. Thus, the discussion presented here must be read as a guideline for the definition of an ontology of a SAKB.

Access control authorizations. The first kind of authorizations are those entailed by access control policies. In general, an access control authorization can be modeled as a triple (u,p,o) stating that subject u has the right to execute privilege p on object o. Thus, in some way, an access control authorization represents a relationship p between u and o, meaning that u can exercise p on o. Therefore, we decide to encode an access control authorization for privilege p as an instance of an OWL object property, named p, defined between the authorized person and the authorized resource.

To model all possible access control authorizations we have to introduce a different object property for each action supported in the OSN (see Section 3). It is interesting to note that by properly defining the object property encoding access control authorizations we can automatically propagate the authorizations on the basis of the classification defined among actions.

Example 1. *Let us consider, for example, the action Post and assume it has been defined as subclass of action Write. In terms of access control, if the post privilege is authorized to a user, then the write privilege is also authorized. In the proposed framework, the access control authorizations can be automatically inferred provided that object property Post has been defined as subproperty of the object property Write. We do note that this hierarchy may be different than in traditional access control systems. When we use SWRL, anything that is defined for a superclass will also be defined for its subclasses. However, the reverse is not true. So, when we allow an individual to Write, it does not automatically confer the Post authority.*

Prohibitions. Filtering policies state whether the target user is not authorized to access a certain object, in the case of supervised filtering policies, or she prefers not to access, in the case of filtering preferences. Similarly to access control authorizations, a prohibition specifies a relationship between a user and the resource she is not authorized or she prefers not to access. For this reason, also prohibitions can be expressed as an object property between *Person* and *Resource* classes. More precisely, a prohibition for the Read privilege is defined as the OWL object property *PRead*. An instance of this object property <John, URI1>:*PRead* states that Bob has not to read resource URI1. Similarly, to access control authorizations, it is possible to specify how prohibitions have to be propagated by simply defining subproperty.

Example 2. *Let us again consider the three basic actions: Read, Write, Delete, and their prohibited versions: PRead, PWrite, PDelete. We again wish to form a hierarchy of actions in a logical order. That is, if an individual is prohibited from Reading a resource, then she should also be prohibited from Writing and Deleting that resource. To do this, we can simply define PRead to be a subtype of PWrite, which is a subtype of PDelete.*

Admin authorizations. Admin authorizations are those authorizations implied by admin policies, which, we recall, have the aim to authorize users to specify access control or filtering policies. Therefore, admin policies entail two types of authorizations: authorizations to specify access control policies, to which we simply refer as *admin authorizations*, and authorizations to specify filtering policies, i.e., *admin prohibitions*. In general, an admin authorization can be represented as a triple (u,p,o) stating that user u is authorized to specify access control policies for privilege p on object o. Thus, similarly to authorizations and prohibitions also admin authorizations can be expressed as an object property between *Person* and *Resource* classes.

Example 3. *According to this modeling, we can define the Object property AdminRead, whose instances state that a given user is authorized to express access control policies granting the read privilege on a given object. Consider the instance <Bob, URI1>:AdminRead, which states that Bob is authorized to specify access control policies granting the read privilege on the URI1 object.*

Similarly, to access control authorizations, it is possible to specify how admin authorizations have to be propagated by simply defining subproperty.

Example 4. *Let us declare the previously mentioned property AdminRead and further create the properties AdminWrite and AdminAll. We declare AdminAll to be a subproperty of both AdminWrite and AdminRead. Consider our running example where Alice owned a photo. Let us assume that this grants her the AdminAll authorization on the photo. If Alice attempts to allow Bob to Read the photo, an action which is restricted to individuals with the AdminRead property, then this is allowed via the AdminAll property.*

In contrast, an admin prohibition can be represented as a tuple (s,t,o,p), which implies that user s (supervisor) is authorized to specify filtering policies for the privilege p applying to the target user t and to object o. Differently from previous authorizations, admin prohibitions can not be represented as properties in that they do not represent a binary relationship. For that reason, we decide to model admin prohibitions as an OWL class *Prohibition*. This makes us able to specify all the components of the prohibition as class properties. More precisely, given an admin prohibition (s,t,o,p), we can model the authorized supervisor has an object property *Supervisor* between the *Prohibition* and *Person* classes. Similarly, the target user can be represented as an object property *TargetUser* between the *Prohibition* and *Person* classes, and the target object as an object property *TargetObject* between the *Prohibition* and the *Resource* classes. In contrast, the privilege over which the supervisor is authorized to state filtering policies is not represented as an object property. Indeed, to automatically propagate admin prohibitions we prefer to specify the privilege directly as the class name. Thus, as an example, instances of the *ProhibitionRead* class state admin prohibitions authorizing the specification of filtering policies for the read privilege. By properly defining subclasses it is possible to automatically infer new admin prohibitions.

Example 5. *Suppose we have a generic Prohibition class with the subclasses PRead and PView. We then create another subclass of each of these as PAll. Suppose we have two individuals, John and Jane, and John is Jane's father. In this scenario, John should be allowed to filter what videos his daughter is able to see. That is, we have a prohibition ("John", "Jane", Video, PAll). Now, for any video and any permission, John can disallow those that he wishes.*

5.2 Security Rules

The proposed framework translates each security policy as a SWRL security rule where the antecedent encodes the conditions specified in the policy (e.g., conditions denoting the subject and object specifications), whereas the consequent encodes the implied authorizations or prohibitions. In particular, since we model security rules as SWRL rules, the SWRL body states policy conditions over the SNKB, i.e., conditions on ontologies introduced in Section 3, whereas the SWRL head entails new instances of the SAKB, i.e., instances of the ontology introduced in Section 5.1. As a consequence, the specification of SWRL security rules is strictly bound to the ontologies supported by the OSN to model SN and SA knowledge bases. This implies that it is not possible to provide a formalization of generic SWRL rules, since these can vary based on the considered ontologies. In contrast, in this section, we aim to present some meaningful examples of possible SWRL security rules defined on top of ontologies adopted in our framework.

We start by considering the admin policy stating that the owner of an object is authorized to specify access control policies for that object. The corresponding SWRL rule defined according to the ontologies presented in the previous sections is the following:

```
Owns(?grantor, ?targetObject)⟹ AdminAll(?grantor,
?targetObject)
```

The evaluation of the above rule has the result of generating a different instance of the object property *AdminAll* for each pairs of user and corresponding owned resource. It is relevant to note that this authorization is propagated according to the ontology modeling the SAKB. Thus, since the framework exploits the one introduced in Section 5.1, the above authorization is propagated also to *AdminRead* and *AdminWrite*.

Another meaningful admin policy for a social network is the one stating that if a user is tagged to a photo then she is authorized to specify access control policies for the read privilege on that photo. This can be encoded by means of the following SWRL security rule:

```
Photo(?targetObject) ∧ photoOf(?grantor, ?targetOb-
ject) ⟹ AdminRead(?grantor,?targetObject)
```

The above rules are interesting examples stressing how in the proposed framework, it is possible to easily specify admin policies whose implementation in a non semantic-based access control mechanism would require complex policy management. Indeed, providing the OSN with ontologies modeling the relationships between users and resources (e.g., modeling ownership or tagging relationships) makes the SA able to specify admin policies by simply posing conditions on the type of the required relationship. In contrast, enforcing these admin policies in a traditional access control mechanism would require implementing complex policy management functionalities, in that it would be required to first determine all possible relationships between users and resources then to specify admin authorizations for all of them. Rather in the proposed framework this task is performed by the reasoner.

Example 6. *Table 1 presents some examples of SWRL security rules. The first security rule encodes a filtering pol-*

	SWRL rule
(1)	Video(?targetObject,) $\wedge$ ParentOf(Bob,?controlled) $\Longrightarrow$ PRead(?controlled,?targetObject)
(2)	Owner(Bob,?targetObject) $\wedge$ Photo(?targetObject) $\wedge$ Friend(Bob,?targetSubject) $\Longrightarrow$ Read(?targetSubject,?targetObject)
(3)	Photo(?targetObject) $\wedge$ photoOf(Alice,?targetObject) $\wedge$ Friend(Alice,?targetSubject) $\Longrightarrow$ Read(?targetSubject,?targetObject)
(4)	Photo(?targetObject) $\wedge$ Owns(?owner, ?targetObject) $\wedge$ Friend(?owner, ?targetSubject1) $\wedge$ Friend(?targetSubject1, ?targetSubject2) $\Longrightarrow$ Read(?targetSubject2, ?targetObject)

Table 1: Examples of SWRL security rules

icy stating that Bob's children can not access videos. Once this rule is evaluated, an instance of prohibition for each of Bob's children and video resource is created. In contrast, the second security rule corresponds to an access control policy stated by Bob to limit the read access to his photos only to his direct friend, whereas the third encodes an access control policy specifying that photos where Alice is tagged can be accessed by her direct friends. Finally, the fourth rule specifies that if a person has a photo, then friends of their friends (an indirect relationship) can view that photo.

6. SECURITY RULE ENFORCEMENT

Our framework acts like a traditional access control mechanism, where a *reference monitor* evaluates a request by looking for an authorization granting or denying the request. Exploiting this principle in the proposed framework implies retrieving the authorizations/prohibitions by querying the SAKB ontology. Thus, for example, to verify whether a user u is authorized to specify access control policies for the read privilege on object o, it is necessary to verify if the instance *AdminRead(u,o)* is in the ontology, i.e., to perform an instance checking. This implies that before any possible requests evaluation all the SWRL rules encoding security policies have to be evaluated, thus to infer all access control/admin authorizations as well as all prohibitions. For this reason, before policy enforcement it is required to execute a preliminary phase, called *policy refinement*. This phase aims to populate the SAKB with the inferred authorizations/prohibitions, by executing all the SWRL rules encoding security policies.

Once authorizations/prohibitions are inferred, security policy enforcement can be carried out. In particular, access control and filtering policies are evaluated upon an *access request* is submitted, whereas admin policies are evaluated when an *admin request* is submitted. In the following, we present both the request evaluation by showing how the corresponding policies are enforced.

6.1 Admin request evaluation

An admin request consists of two pieces of information: the name of the *grantor*, i.e., the user that has submitted the admin request, and the access control or filtering policy the grantor would like to specify, encoded as SWRL rule, that is, the *submitted SWRL*. The submitted SWRL has to be inserted in the system only if there exists an admin authorization in the SAKB for the grantor. For example, if the submitted rule requires to specify an access control policy for the read privilege on targetObject, then there must exist an instance of <grantor, targetObject>:*Read.* Note that information about the privilege and the targetObject can be retrieved directly from the submitted SWRL. Thus, in order to decide whether the request above can be authorized or not, a possible way is to query the SAKB to retrieve the corresponding admin authorization, if any. If there exists an instance, then the submitted SWRL can be evaluated, otherwise the framework denies to the grantor the admin request. An alternative way is to rewrite the submitted SWRL by adding in its body also condition to verify whether there exists an admin authorization in the SAKB authorizing the specification of the rule. The following example will clarify the underlying idea.

EXAMPLE 7. *Let us assume that the system receives the following admin request: {Bob, SWRL$_1$}, where SWRL$_1$ is the following:*

SWRL$_1$: Owns(Bob,?targetObject) $\wedge$ Photo(?targetObject) $\wedge$ Friend(Bob,?targetSubject)
$\Longrightarrow$ Read(?targetSubject,?targetObject)

In order to determine the result of the admin request, the framework has to verify the existence of <Bob,targetObject>: AdminRead instance in the SAKB. This check can be incorporated in the body of SWRL$_1$ by simply modifying it as follows:

New_SWRL$_1$: **AdminRead(Bob,?targetObject)** $\wedge$ Owns(Bob,?targetObject) $\wedge$ Photo(?targetObject) $\wedge$ Friend(Bob,?targetSubject)
$\Longrightarrow$ Read(?targetSubject, ?targetObject)

Then New_SWRL$_1$ is evaluated with the consequence that Read access control authorizations will be inserted in SAKB only if Bob is authorized to specify them by an admin policy.

In case of an admin request submitting a filtering policy, to decide whether the grantor is authorized to specify that policy, a search is required in the *Prohibitions* class (i.e., the subclass corresponding to the action the filtering policy requires to prohibit) for an instance having the property *Grantor* equal to the grantor and the properties *Controlled* and *TargetObject* equal to the controlled and *TargetObject* specified in the head of the submitted SWRL rule, respectively. Also in this case, we can adopt an approach based on SWRL rewriting.

EXAMPLE 8. *Let us assume that the framework receives the following admin request: {Bob, SWRL$_2$}, where SWRL$_2$ is the following:*

SWRL$_2$: Video(?targetObject) $\wedge$ ParentOf(Bob,?controlled) $\Longrightarrow$ PRead(?controlled,?targetObject)

Then, the system can modify the submitted SWRL as:

New_SWRL$_2$: **PRead(?p)** $\wedge$ **Grantor(?p,Bob)** $\wedge$ **Controlled(?p,?controlled)** $\wedge$ **TargetObject(?p,?targetOb-**

ject) ∧ `Video(?targetObject)` ∧ `ParentOf(Bob,?controlled)` $\Longrightarrow$ `PRead(?controlled,?targetObject)`

whose evaluation has the effect to insert instances of PRead property (i.e., read prohibitions) only if there exists an Admin Prohibition (Bob,c,o,Read), where c is Bob's children, and o is a video resource. Note that this is valid also if the submitted SWRL explicitly specifies the name of the controlled user (e.g.,.... $\Longrightarrow$ `PRead(Alice,?targetObject)`*).*

6.2 Access request evaluation

In general, an access request can be modeled as a triple (u, p, URI), which means that a user u requests to execute the privilege p on the resource located at URI. To evaluate this request the framework has to verify whether there exists an access control authorization granting p on URI to requester r. However, since the proposed system also supports filtering policies, the presence of such an authorization does not necessarily imply that r is authorized to access URI because there could be a prohibition denying access to the resource to the user. Thus, to evaluate whether an access request has to be granted or denied, it is necessary to perform two queries to the SAKB. The first to retrieve authorizations and the second to retrieve prohibitions. More precisely, if u requires the read privilege *Read*, the system has to query the instances of object property *Read* and *PRead*. In particular, both the queries look for instance $<u,URI>$ (i.e.,$<u,URI>$:*Read* and $<u,URI>$:*PRead*). Then, the access is granted if the first query returns an instance and the second returns the empty set. It is denied otherwise.

Example 9. *Consider again Example 5, with the addition of a person named "Susan". Susan is a friend of Jane and has posted a video which she allows to be seen by all of her friends. However, Jane's father prohibits her from viewing videos. When a request is made by Jane to see Susan's video, the authorization and the prohibition queries are performed. The authorization query returns a Read permission, but the prohibition query returns a PRead. This means that Jane will be unable to view the video.*

7. FRAMEWORK ARCHITECTURE

In our proposed framework, we plan to build several layers on top of the existing online social network application. We plan to implement our prototype using Java based open source semantic web application development framework called JENA [3], since it offers an easy to use programmatic environment for implementing the ideas discussed in this paper. Here, we describe each of these layers independently, as well as the motivation behind choosing specific technologies in our framework. While we use specific instances of Facebook as the over-arching application utilizing the lower level semantic layers, any social network application could be modified to use the design we describe here.

7.1 RDF Datastore

We assume the use of a general RDF triple-store to hold the underlying data. In this representation, all facts about an entity are recorded as a triple of the form <Subject, Predicate, Object>. So, suppose we have an individual named John Smith, who is assigned a unique identifier 999999, would give us the tuple <999999, *foaf:Name*, "John Smith">. We plan to use a similar format in a separate table to store a list of authorizations so that we do not have to re-infer them each time an authorization is requested. For the data storage system, we plan to use MySQL because of its availability and because of its ease of interface with JENA.

We note here that an RDF datastore differs from a relational database in that there is no database method of ensuring that constraints are maintained on the ontology as a whole, such as making sure that a defined *Person* has a name. The database representation of this fact is no different than the non-essential statement that the person lives in Albuquerque. However, we plan to use OWL-DL statements to define these constraints, and then allow the RDF/OWL engine to enforce the constraints as described below.

7.2 Reasoner

Any reasoner that supports SWRL rules can be used to perform the inferences described in this paper. However, we chose SweetRules[4] because it interfaces with JENA and has a rule-based inference engine. This means that we can use both forward and backward chaining in order to improve the efficiency of reasoning for enforcing our access control policies. Forward chaining is an inference method where the engine will begin with the data provided and look at the established inference rules in an attempt to derive further information. This can be used when the system needs to infer permissions on a large scale, such as when a resource is added for the first time. At this point, there will be a large one-time addition of authorizations to the allowed list of users. However, later, after other friends are added, checking to see if a user has access to a limited number of resources can be done through backward chaining. Basically, in backward chaining, we begin with the desired goal (e.g., goal is to infer whether "John have permission to see the photo album A"), and check whether it is explicitly stated or whether it could be inferred by some other rules in a recursive fashion. Obviously, this will allow a result to be inferred about an individual (e.g., John) without re-checking all other individuals.

In [11], Mika proposes a basic general social network, called Flink, based on a semantic datastore using a similar framework to that we have proposed, but using several different specific semantic technologies. However, he does specify that their implementation, using backwards- and forward-chaining is efficiently scalable to millions of tuples, which provides an evidence of the viability of our proposed scheme.

7.3 RDF/OWL engine

For the RDF/OWL interface, we chose to use the JENA API. We use this to translate the data between the application and the underlying data store. JENA has several important features that were considered in its use. First, it is compatible with SweetRules. Secondly, it supports OWL-DL reasoning which we could use to verify that the data is consistent with the OWL restrictions on the ontology. The OWL restrictions are simple cardinality and domain/range constraints such as every person has to have a name and must belong to at least one network. To enforce these constraints, we plan to have the application layer pass the statements to be entered about an individual until all have been collected. We then have JENA insert these statements into

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

[4] http://sweetrules.projects.semwebcentral.org/

the database and then check the new model for consistency. If there are any constraints that have been violated, then we pass this information back to the social network application and have it gather the required information from the user.

8. CONCLUSIONS

In this paper, we have proposed an extensible fine grained on-line social network access control model based on semantic web tools. In addition, we propose authorization, admin and filtering policies that are modeled using OWL and SWRL. The architecture of a framework in support of this model has also been presented. We intend to extend this work toward several directions. A first direction arises by the fact that supporting flexible admin policies could bring the system to a scenario where several access control policies specified by distinct users can be applied to the same resource. Indeed, in our framework social network's resources could be related to different users according to the supported ontology. For example, a given photo could be connected to the owner, say Bob, as well as to all users with which is tagged with, say Alice and Carl. According to the semantics of admin policies, it could be the case that some of these tagged users are authorized to specify access control policies for that resource, say only Alice. For example, Alice could have specified that the photos tagged to her can be accessed only by her direct friends, whereas Bob could have specified that his photos have to be accessed by his direct friend and colleagues. In order to enforce access control the framework have to decide how the specified access control policies have to be combined together. As such, a first important extension of the proposed framework will be the support of a variety of policy integration strategies. As a further important future work, we plan to implement our framework using the ideas discussed in Section 7 and test the efficiency of various ways of combining forward and backward chaining based reasoning for different scenarios.

9. ACKNOWLEDGMENTS

This work was partially supported by National Science Foundation Grants Career-0845803, CNS-0716424 and Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265.

10. REFERENCES

[1] B. Ali, W. Villegas, and M. Maheswaran. A trust based approach for protecting user data in social networks. In *2007 Conference of the Center for Advanced Studies on Collaborative research (CASCON'07)*, pages 288–293, 2007.

[2] S. Berteau. Facebook's misrepresentation of Beacon's threat to privacy: Tracking users who opt out or are not logged in. CA Security Advisor Research Blog, Mar. 2007.

[3] D. Brickley and L. Miller. FOAF vocabulary specification 0.91. RDF Vocabulary Specification, Nov. 2007. Available at http://xmlns.com/foaf/0.1.

[4] B. Carminati, E. Ferrari, and A. Perego. Enforcing Access Control in Web-based Social Networks. *ACM Transactions on Information & System Security, 2008. To appear*, 4(3):191–233, 2001.

[5] B. Carminati, E. Ferrari, and A. Perego. Security and privacy in social networks. In M. Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology, 2nd Edition*, volume VII, pages 3369–3376. IGI Publishing, Sept. 2008.

[6] H.-C. Choi, S. R. Kruk, S. Grzonkowski, K. Stankiewicz, B. Davis, and J. G. Breslin. Trust models for community aware identity management. In *Identity, Reference, and the Web Workshop (IRW 2006)*, 2006. Available at: `http://www.ibiblio.org/hhalpin/irw2006/skruk.pdf`.

[7] N. Elahi, M. M. R. Chowdhury, and J. Noll. Semantic access control in web based communities. In *ICCGI '08: Proceedings of the 2008 The Third International Multi-Conference on Computing in the Global Information Technology (iccgi 2008)*, pages 131–136, Washington, DC, USA, 2008. IEEE Computer Society.

[8] T. W. Finin, A. Joshi, L. Kagal, J. Niu, R. S. Sandhu, W. H. Winsborough, and B. M. Thuraisingham. R*owl*bac: representing role based access control in *owl*. In *SACMAT*, pages 73–82, 2008.

[9] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium, May 2004. Available at: `http://www.w3.org/Submission/SWRL`.

[10] S. R. Kruk, S. Grzonkowski, H.-C. Choi, T. Woroniecki, and A. Gzella. D-FOAF: Distributed identity management with access rights delegation. In *Proceedings of the 1st Asian Semantic Web Conference (ASWC 2006)*, LNCS 4185, pages 140–154. Springer Verlag, 2006.

[11] P. Mika. *Social Networks and the Semantic Web*, volume 5 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.

[12] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. *Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder*. 2003.

[13] World Wide Web Consortium. Defining n-ary relations on the semantic web, 2006. Available at: `http://www.w3.org/TR/swbp-n-aryRelations/`.

[14] World Wide Web Consortium. Status for resource description framework (rdf) model and syntax specification. Available at: `http://www.w3.org/1999/.status/PR-rdf-syntax-19990105/status`.

[15] M. I. Yagüe, M. del-mar Gallardo, and A. Maña. Semantic access control model: A formal specification. In *ESORICS 2005*, pages 24–43, 2005.