Commitment-Based Privacy Management in Online Social Networks

Nadin Kökcıyan and Pınar Yolum

Department of Computer Engineering, Bogazici University, 34342 Bebek, Istanbul, Turkey
{nadin.kokciyan, pinar.yolum}@boun.edu.tr

Abstract. Systems need to preserve their users’ privacy. In social software, such as online social networks, preserving privacy is especially difficult since content is managed by more than one entity. As a result, users are faced with privacy breaches, where their private content becomes visible to people to which the content is not targeted for. To manage privacy more diligently, social networks have been employing customizable privacy agreements such that each individual can set its own settings. However, these customizations can be in themselves conflicting or may cause inconsistencies when other users’ agreements are in place. To deal with these, this paper develops an approach for managing users’ privacy in online social networks and implements it in a system called PriGuARD. The approach starts with a user interface that allows users to enter their own constraints on whom to show content to. The system, then generates appropriate commitments between the users and the system to formalize the users’ needs. The privacy information, such as the relations among users, various content types in the system, and so on are captured in an ontology. Using ontological reasoning, the system checks whether the current situation of the system indeed violates any of the commitments and notifies the user to take appropriate action.

1 Introduction

Online social systems have become an important part of everyday life. While initial examples were used to share personal content with friends (e.g., Facebook.com), more and more online social systems are also used to do business (e.g., Yammer.com). Generally, these systems serve a large number of users; however each user shares content with only a small subset of these users. This subset may even change based on the type of the content or the current context of the user. For example, a user might share contact information with all of her acquaintances, while a picture might be shared with friends only. If say, the picture shows the person sick, the user might not even want all her friends

⋆ This work has been supported by BAP under grant 03A102P and by TUBITAK under grant 113E543. We would like to thank the anonymous reviewers for their helpful comments.
to see it. That is, privacy constraints vary based on person, content, and context. This requires systems to employ a customizable privacy agreement with their users. However, when that happens, it is difficult to enforce users’ privacy requirements.

Consider an online social network, where both Charlie and Linus are users. They have various privacy constraints as depicted with the following scenarios that are inspired from Kafali et al. [1].

**Example 1** Charlie wants his friends to see the people that he is together with but not his colleagues. However, Linus is both a friend and a colleague.

**Example 2** Linus wants his friends to see his media but does not want other that are not friends to see it.

**Example 3** Charlie wants his friends to see his media but not his location. He posts pictures but not his location.

On a first look, the above examples resemble typical access control. However, in typical access control scenarios, there is a single authority (i.e., administrator) that can grant accesses as required. However, in social systems, each user can essentially contribute to the sharing of content (e.g., by resharing content put by others). Thus, there are multiple sources of control. This requires a different perspective to understanding privacy [1].

This paper develops an approach for managing users’ privacy constraints in online social systems and implements it in a tool (PriGuard) to detect privacy violations. We adopt the social network representation that was used in PROT OSS [1], where users are related to each other with some relations (such as friendOf) and share some content (such as picture) with certain others. Contrary to PROT OSS, here we represent all the information relevant to the social network using an ontology. As a starting point, we take a customizable privacy agreement, where users can specify what types of content is to be shown or not shown to which individuals or groups of people. Online social network (OSN) is responsible for satisfying these constraints. Our system takes user input, checks for inconsistencies, resolve if any, and form commitments [2] among users and the OSN. These commitments capture the privacy obligations of the OSN to each individual user. Then, the system computes the particular conditions that would violate these commitments based on the current state of the social network as represented with the ontology. The reasoning is done in OLP, which is a tool that can do both Prolog and ontological inferences [3]. We walk through the above examples to illustrate parts of our approach.

The rest of this paper is organized as follows: Section 2 gives a brief introduction on the concepts used in our framework. Section 3 explains our approach in detail with examples. Section 4 discusses our approach in relation to relevant related work and gives pointers for future work.
2 Technical Framework

In order to represent the privacy agreement between a user and the online social network, we make use of commitments. A commitment is a contract made between two parties [2]. A commitment is denoted as a four-place relation: \( C(\text{debtor}, \text{creditor}, \text{antecedent}, \text{consequent}) \). The debtor is committed to the creditor to bring about the consequent if the creditor brings about the antecedent [4]. Initially, when the commitment is created, the commitment is in a conditional state. If the antecedent is achieved, the commitment moves to an active state. Moreover, if the debtor fails to provide the consequent of an active commitment then this commitment is violated. For example, in an online social network, a commitment between the OSN operator and its user :charlie can be formalized as follows: 
\[
C_0(\text{osn}, :\text{charlie}, \text{isFriendOf}(:\text{charlie}, X), \text{canSeeMediaOf}(X, :\text{charlie}))
\]
In \( C_0 \), the debtor :osn promises to the creditor :charlie for revealing :charlie’s media to \( X \) if :charlie declares \( X \) to be a friend. For example, if :charlie declares :patty to be a friend then \( C_0 \) becomes an active commitment as the condition \( \text{isFriendOf}(:\text{charlie}, :\text{patty}) \) holds. Furthermore, if :osn fails to bring about \( \text{canSeeMediaOf}(\text{patty}, :\text{charlie}) \) i.e. :patty cannot see :charlie’s media, \( C_0 \) is violated.

Each commitment exists in a specific domain. We develop an ontology to describe the OSN domain using Web Ontology Language (OWL) [5]. An ontology is a conceptualization of a domain [6], and it consists of three main entities: (1) concepts (classes) are sets of instances e.g. Agent is a concept representing OSN users, the ClassAssertion(Agent :charlie) states that :charlie is an instance of concept Agent, (2) data properties are used to describe attributes of a concept e.g. DataPropertyAssertion(hasName :charlie “Charlie Brown”) is a property assertion stating that :charlie’s name is “Charlie Brown”, (3) object properties (relationships) are used to relate instances to each other e.g. ObjectPropertyAssertion(isFriendOf :charlie :patty) is a property assertion stating that :charlie is related to :patty via isFriendOf object property. Hence, an ontology describes a domain with a set of class and property assertions. Moreover, OWL restrictions can be used to add constraints on concepts and properties; properties can be defined as being functional, symmetric, inverse or transitive. Such modeling capabilities are useful to specify semantics, so new information can be inferred based on the described domain. Using an ontology, each place of a commitment can be represented semantically. OWL uses the open world assumption (OWA), i.e. any statement that is not known cannot be considered as a false statement. Moreover, ontologies can be augmented with rules such as Semantic Web Rule Language (SWRL) rules for more expressiveness [7]. Description Logic reasoners such as Pellet [8] can reason on ontologies augmented with SWRL rules.

A domain can also be modeled using logic programming (e.g., Prolog [9]). Prolog programs describe a domain with \( k \)-ary relations represented as facts and rules. A fact is a predicate expression, and a Prolog rule consists of a Head (a positive atomic expression) and a Body (conjunction and disjunction of predicates) and is of the form \( \text{Body} \implies \text{Head} \). The Prolog interpreter runs queries
about the facts and rules represented in its knowledge base. Given a query (goal),
the Prolog interpreter attempts to prove it using backtracking search. Therefore,
alternative solutions can be found for a given query. In contrast to ontologies,
Prolog obeys the closed world assumption (CWA) i.e. if a proposition cannot
be proved then it is assumed to be false. In privacy settings, CWA is more
appropriate as we can deal with negations e.g. finding non-friends of a user.

Ontological Logic Programming (OLP) [3] combines the logic programming
and ontological reasoning. It uses Prolog [9] as the logic programming framework
and Pellet [8] as the DL reasoner. OLP uses two knowledge bases: (1) Prolog
Knowledge Base stores non-ontological facts and rules that are interpreted by
the Prolog interpreter, (2) Semantic Knowledge Base stores ontological entities
(concepts, properties, instances) and SWRL rules that are used for reasoning.
An OLP program can import ontologies and all ontological entities together
with SWRL rules can be used within the OLP program. OLP can also be used
to modify the Semantic Knowledge Base by adding or removing ontological
entities. We use OLP to detect privacy violations as we can reason on ontologies
with CWA.

3 Privacy Management

Our proposed approach is depicted as a flow diagram in Figure 1. A user of the
OSN specifies her privacy agreement where she declares her privacy preferences.
Then, the system processes the privacy agreement to generate corresponding
commitments between the users and the OSN. Following this, the system generates
the statements wherein these commitments would be violated. Finally, the
system checks whether these statements hold in the current state, which would
mean a violation of privacy.

![Fig. 1. Detecting privacy violations according to a user's privacy agreement](image)

3.1 OSN Representation

An OSN consists of various components: (i) users are both providers and con-
sumers of information in an OSN, (ii) content is shared by users e.g. a user
posts a video, (iii) :osn manages users’ access to shared content via its behavior
rules e.g. :osn shows public pictures to everyone, (iv) relations are initiated and
terminated by users e.g. a user is connected to another user via the friendship
relation, (v) inference rules can be used to make further inferences based on the domain.

**Ontology** of OSN domain is being developed\(^1\). **Agent** represents a single user of the OSN. A user may be together with other users, for this we use **Group**. Each user in the group is connected to the group with **hasMember**. A user can be at a specific **Location** with a group of people, for this **LocationItem** is used to represent both location and group information. In other words, **LocationItem** is related to **Location** via the functional property **hasLocation** and to **Group** via the functional property **hasGroup**. The fact that these properties are functional ensures that each location item consists of one location and one group. A user’s shared content is represented with **Media** and its subclasses \{**Picture**, **Video**\}. **Media** can include geotags for which we use **hasGeotag**. Remaining object properties can be grouped into three categories: (i) properties that represent content sharing behavior of users: **sharesMedia** and **sharesLocationItem**, (ii) relation properties that represent symmetric relationships between users: **isConnectedTo** and its subproperties \{**isFriendOf**, **isColleagueOf**\} e.g. **isFriendOf** \(X,Y\) meaning that \(X\) declares \(Y\) to be a friend and \(Y\) declares \(X\) to be a friend, (iii) privacy properties that represent access of users to shared content of other users: **canSeeLocationOf**, **canSeeMediaOf** and **canSeeWithOf**. Properties described in (ii) and (iii) are defined between users e.g. : \(u_1\) is connected to : \(u_2\) via **isConnectedTo**. In real social networks, if a user can access some content of another user, then the user can share the content with others. Thus, the content can freely propogate in the system. To mimic this, we defined **canSeeMediaOf** as a transitive property so that if a user can see media of another user, the first users’ friends can potentially see the same content. The described ontology is depicted in Figure 2 where ovals depict concepts, labeled arrows depict object properties between concepts.

A state is captured by the class and object property assertions in the given ontology. An example state, which is inspired by the work of Kafalı et al. \[^1\]

\[^1\] Each concept is denoted with text in mono-spaced format Concept, each relationship is denoted with italic text relationship, and each instance is denoted with a colon followed by text in mono-spaced format :instance
is specified in functional-style syntax in Table 1. In this example, the state is as the following: :charlie and :patty, :charlie and :linus, :linus and :sally are friends; :charlie and :linus are colleagues; :charlie shares two media files: :videoBeach and :pictureBeach that is geotagged with :Istanbul; :linus shares :pictureConcert; :patty is located in :Istanbul together with :charlie. The state can also be represented in a separate ontology (e.g. a private ontology of an OSN) by simply importing the domain ontology and instantiating it in the imported ontology.

### Table 1. Assertions for an example OSN state expressed in functional-style syntax

| ClassAssertion(Picture :pictureBeach) | ClassAssertion(Video :videoBeach) |
| ClassAssertion(Location :Istanbul) | ClassAssertion(Agent :charlie) |
| ClassAssertion(Agent :linus) | ClassAssertion(Agent :sally) |
| ClassAssertion(Agent :patty) | ClassAssertion(LocationItem :l1) |
| ClassAssertion(Group :g1) |
| ObjectPropertyAssertion(isFriendOf :charlie :patty) |
| ObjectPropertyAssertion(isFriendOf :charlie :linus) |
| ObjectPropertyAssertion(isFriendOf :linus :sally) |
| ObjectPropertyAssertion(isColleagueOf :charlie :linus) |
| ObjectPropertyAssertion(sharesMedia :charlie :pictureBeach) |
| ObjectPropertyAssertion(hasGeotag :pictureBeach :Istanbul) |
| ObjectPropertyAssertion(sharesMedia :charlie :videoBeach) |
| ObjectPropertyAssertion(sharesMedia :linus :pictureBeach) |
| ObjectPropertyAssertion(hasMember :g1 :charlie) |
| ObjectPropertyAssertion(hasMember :g1 :patty) |
| ObjectPropertyAssertion(hasGroup :l1 :g1) |
| ObjectPropertyAssertion(hasLocation :l1 :Istanbul) |
| ObjectPropertyAssertion(sharesLocationItem :patty :l1) |

**Rules** are used to represent complex inferences that cannot be achieved with basic OWL constructs. There are two types of rules: (i) domain specific inference rules and (ii) behavior rules of the :osn operator that describes how the :osn operates. Both rules contribute into the reasoning process. Inference rules come with the domain. One such inference rule ($I_1$) is shown in Table 2. This rule states that if $x$ shares media $m$ that is geotagged with location $l$ and $y$ can see media of $x$, then $y$ will see the location $l$ of $x$. On the other hand, behavior rules are private rules used by the :osn operator to control access of users to the shared content. A user can only see some content if the behavior rules apply. In another words, the default privacy rules are the behavioral rules. Here we consider behavior rules ($B_1$ and $B_2$) as shown in Table 2. $B_1$ states that if $x$ is a friend of $y$ then :osn will show the media of $y$ to $x$. In this rule, we use ontological reasoning as Picture and Video are subclasses of Media. In other words, if $B_1$ is applied, $x$ can see both pictures and videos of $y$. $B_2$ states that if $x$ is a friend of $y$ then :osn will show the people whom $y$ is together with to $x$. 
Table 2. Inference (I) and Behavior Rules (B) as SWRL Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>SWRL Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1:</td>
<td>( \text{sharesMedia}(!x,!m) \land \text{hasGeotag}(!m,!l) \land \text{canSeeMediaOf}(!y,!x) \rightarrow \text{canSeeLocationOf}(!y,!x) )</td>
</tr>
<tr>
<td>B1:</td>
<td>( \text{isFriendOf}(!x,!y) \rightarrow \text{canSeeMediaOf}(!x,!y) )</td>
</tr>
<tr>
<td>B2:</td>
<td>( \text{isFriendOf}(!x,!y) \rightarrow \text{canSeeWithOf}(!x,!y) )</td>
</tr>
</tbody>
</table>

Privacy Agreement is made between a user and the :osn. It consists of privacy preferences of the user and informs :osn about how to reveal content to other users. However, the existence of a privacy agreement does not mean that :osn will honor its clauses; the :osn is free to act according to its behavior rules. The incompatibility between a user’s privacy agreement and behavior rules of :osn leads to a typical privacy violation. OSNs generally provide an interface to manage the privacy settings of shared content. For example, in Facebook, one can allow or restrict access to posts (status updates, pictures, videos etc. as a whole) to some audience. We developed a prototype of PriGuard\(^2\) to enable users to manage their privacy agreement in a fine grained way. A user can specify who can or cannot see his media, location, and people that are together with him. For a specific content, the audience is divided into two groups of people: a group who can see the content (canSeeGroup) and a group who cannot (cantSeeGroup).

Table 3. Mappings from specific groups to conditional statements. :user denotes the creditor of a commitment, X and Y are variables, [:u\(_1\),:u\(_2\), ..,:u\(_n\)] is a list of users

<table>
<thead>
<tr>
<th>specific groups</th>
<th>conditions used in antecedent of commitments</th>
</tr>
</thead>
<tbody>
<tr>
<td>everyone</td>
<td>( \text{isConnectedTo}(\text{:user,X}) \lor \neg(\text{isConnectedTo}(\text{:user,X})) )</td>
</tr>
<tr>
<td>friends</td>
<td>( \text{isFriendOf}(\text{:user,X}) )</td>
</tr>
<tr>
<td>not friends</td>
<td>( \text{Agent}(X) \land \neg(\text{isFriendOf}(\text{:user,X})) )</td>
</tr>
<tr>
<td>colleagues</td>
<td>( \text{isColleagueOf}(\text{:user,X}) )</td>
</tr>
<tr>
<td>not colleagues</td>
<td>( \text{Agent}(X) \land \neg(\text{isColleagueOf}(\text{:user,X})) )</td>
</tr>
<tr>
<td>[:u(_1),:u(_2), ..,:u(_n)]</td>
<td>( \text{isConnectedTo}(\text{:user,X}) \land \text{member}(X,[:u(_1),:u(_2), ..,:u(_n)]) )</td>
</tr>
</tbody>
</table>

Commitments can be represented semantically using an ontology. In a social network, :osn is always the debtor to preserve privacy and the user is the creditor. The antecedent is a condition that depends on the existence of relationships between the creditor and other users (denoted as X). Mappings from specific group information to conditions are shown in Table 3. For example, if :charlie is the creditor, :user is replaced with :charlie then the specific group “friends” is mapped to the condition “if :charlie declares X to be a friend”. A more complex condition can be represented with the disjunction of relationships e.g. “if :charlie declares X to be a friend or a colleague”. The antecedent condition can also depend on the membership of X to a specific list of users e.g. the user list [:patty,:linus] is mapped to the condition \( \text{isConnectedTo}(\text{charlie,X}) \land \text{member}(X,[:patty,:linus]) \).

\(^2\) A demonstration is available at http://mas.cmpe.boun.edu.tr/nadin/priguard
member(X,:patty,:linus) stating that “if :charlie declares X to be connected to him and to be a member of [:patty,:linus]”. Moreover, the antecedent can be a condition depending on the disjunction of specific groups e.g. “if :charlie declares X to be a friend or a colleague or being a member of [:patty,:linus]”.

On the other hand, the consequent is a privacy property (canSeeLocationOf, canSeeMediaOf, canSeeWithOf) defined between X and the creditor e.g. canSeeMediaOf(X,:charlie) denotes that X can see media of :charlie. The idea is that the antecedent is a declaration done by the user, whereas the privacy constraint captured by the consequent is realized by the debtor (i.e., :osn).

**Generation of Commitments** Once a privacy agreement is specified by the user, a set of commitments will be generated such that the contents of the commitments are constructed using the ontology. :osn commits to the user to act according to the generated commitments. In the privacy agreement, the user specifies canSeeGroup and cantSeeGroup for each specific content. Now we consider some scenarios where a user specifies his privacy preferences differently.

(i) A user specifies neither canSeeGroup nor cantSeeGroup for any content. In such a case, there is no commitment to generate.

(ii) A user can only specify canSeeGroup or cantSeeGroup for a specific content. In such a case, the generation of commitments is straightforward. For example, if :patty declares his friends not to see his media, only one commitment will be generated as: C(:osn,:patty,isFriendOf(:patty,X),canSeeMediaOf(X,:patty)).

(iii) A user can specify both canSeeGroup and cantSeeGroup for a specific content. In this case, firstly we identify the two groups, and secondly we check whether there is any intersection between these two groups. For identifying the groups, we use mappings shown in Table 3. All users that satisfy a specific condition are identified with OLP [3]. For example, if :user selects “friends”, this entity is mapped to the condition isFriendOf(:user,X). Then, X is populated by running the query “isFriendOf(:user,X)” with OLP engine that returns a set of users satisfying the specified condition. The generation of commitments is not always straightforward because a user may both allow and disallow a specific group to see some content. For this, we adopted a conservative approach and we moved users who are specified in both groups to cantSeeGroup. Otherwise, generated commitments would be in conflict among themselves. However, the approach is customizable such that if the user prefers the conflict can be resolved by moving the individuals to canSeeGroup. We refer to the examples described in Section 1 and we consider the example state in Table I.

**Example 1:** If :charlie wants his friends to see the people whom he is together with and his colleagues to not see them, canSeeGroup becomes [:patty,:linus] and cantSeeGroup becomes [:linus]. The intersection is [:linus] so we remove :linus from canSeeGroup. Then, commitments become: C_1(:osn,:charlie,isFriendOf(:charlie,X) ∧ ¬(member(X,[:linus])),canSeeWithOf(X,:charlie)); C_2(:osn,:charlie,isColleagueOf(:charlie,X),¬(canSeeWithOf(X,:charlie))).

**Example 2:** If :linus wants his friends to see his media and non-friends to not
Table 4. Violation Statements

| v1: isFriendOf(:charlie,X), not(member(X,:linus)), not(canSeeWithOf(X,:charlie)) |
| v2: isColleagueOf(:charlie,X), canSeeWithOf(X,:charlie) |
| v3: isFriendOf(:linus,X), not(canSeeMediaOf(X,:linus)) |
| v4: Agent(X), not(isFriendOf(:linus,X)), canSeeMediaOf(X,:linus) |
| v5: isFriendOf(:charlie,X), not(canSeeMediaOf(X,:charlie)) |
| v6: isFriendOf(:charlie,X), canSeeLocationOf(X,:charlie) |

Example 3: If :charlie wants his friends to see his media but not his location, two commitments will be generated as: $C_5(\text{osn, :linus, isFriendOf(:linus,X), canSeeMediaOf(X,:linus)}); C_6(\text{osn, :linus, Agent(X) \land \neg(isFriendOf(:linus,X), \neg(canSeeMediaOf(X,:linus)))})$.

3.2 Generation of Violation Statements

A violation occurs when the debtor fails to bring about the consequent of a commitment, even though the creditor has brought out the antecedent [4]. For detecting violations, violation statements have to be identified according to the generated commitments. For this, we model violation statements as Prolog rules. In a commitment, the consequent is true if the antecedent is true that can be represented as the rule: $\text{antecedent} = \Rightarrow \text{consequent}$ (i.e. $\neg\text{antecedent} \lor \text{consequent}$). Then, the violation rule of a commitment $c$ is the logical negation of this rule that is $\text{antecedent} \land \neg\text{consequent}$. Note that $\neg$ means logical not while “not” has the same meaning as in Prolog. After that a commitment is created, the violation statement is generated accordingly. For commitments $C_1 - C_6$, violation statements $v_1 - v_6$ are created as the following:

Violation statements are defined in Prolog syntax thus logical operators used in commitments are replaced with corresponding Prolog symbols e.g. $\land$ is replaced with “,”, $\lor$ is replaced with “;” and $\neg$ is replaced with “not”. This replacement is straightforward for the antecedent of a commitment as the antecedent remains the same in the violation statement. Additionally, the logical negation of the consequent is used in the violation statement. First, logical negation is applied to the consequent and then syntactic replacements take place if needed. In $v_1$, $C_1$’s antecedent remains the same while the logical negation of $C_1$’s consequent is written in Prolog syntax. In $v_4$, $C_4$’s antecedent is transformed to Prolog syntax. The logical negation of $C_4$’s consequent is $\neg\neg\neg\text{canSeeMediaOf(X,:linus)}$ that is equal to $\text{canSeeMediaOf(X,:linus)}$. Other violation statements are constructed in the same way.
3.3 Detection of Privacy Violations

The generation of commitments consists of two steps. First, we check whether the privacy agreement is consistent in itself, i.e. there should not be any conflicting group declarations (canSeeGroup and cantSeeGroup) for a specific content. Second, the commitments are generated according to a consistent privacy agreement. If the privacy agreement is inconsistent then we transform it into a consistent one. This step consists of detecting overlapping groups as specified in the privacy agreement. In Example 1, :osn will be in a conflicting situation as it does not know whether to reveal :charlie’s content to :linus or not. In such cases, the generation of commitments will cause :osn to violate one of its commitments.

For detection, PriGuard uses the ontology, the inference rules, the behavior rules, the state information and the violation statements. PriGuard detects privacy violations through violation queries. For this, PriGuard runs the query $v_{cid}$ where $cid$ is the commitment id and checks to see if the Body of the violation statement holds. If PriGuard can prove it then corresponding commitment is violated and users are notified about it, otherwise the commitment is not violated. Now, we consider the detection of violations with our motivating examples.

Example 1: $C_1$ and $C_2$ are the commitments, $v_1$ and $v_2$ are the corresponding violation statements. The :osn will apply $B_2$ then :patty and :linus will see the people that are together with :charlie. We ask PriGuard whether any of these violation statements occur. PriGuard cannot prove the query $v_1$ while it can prove the query $v_2$ with the substitution $\{X/:linus\}$. Hence, $C_2$ is violated because of $B_2$, which is not compatible with :charlie’s privacy agreement. This is a typical case where a system does not act in compliance with a user’s privacy agreement.

Example 2: $C_3$ and $C_4$ are the commitments, $v_3$ and $v_4$ are the corresponding violation statements. The :osn will apply $B_1$ then :charlie and :sally will see :linus’ media. PriGuard cannot prove the query $v_3$ while it can prove the query $v_4$ with the substitution $\{X/:patty\}$. $C_4$ is violated because canSeeMediaOf is a transitive property, and as a result of ontological reasoning :patty can see :linus’ media. In other words, :patty can see media of :charlie, :charlie can see media of :linus thus :patty can too. This is essentially a different type of violation, where the violation takes place because the media ends up being propagated by other friends of the user, not because the OSN acted against the user’s will.

Example 3: $C_5$ and $C_6$ are the commitments, $v_5$ and $v_6$ are the corresponding violation statements. The :osn will apply $B_1$ then :patty and :linus will see :charlie’s media. PriGuard cannot prove the query $v_5$ while it can prove the query $v_6$ with the substitutions $\{X/:patty\}$ and $\{X/:linus\}$. Thus :patty and :linus can see location of :charlie, and $C_6$ is violated. This violation takes place because PriGuard applies $I_1$ with the substitutions $\{?x/:charlie\}$,
Even that the location information is not posted explicitly, it can be inferred because of a geotagged picture. This is a case that resembles various privacy attacks on celebrities [10]. In principle, this is a different type of violation from the previous ones, where the violation takes place because of an inference rule that contributes into the reasoning process.

4 Discussion

There is a rich body of work on privacy management on online social networks. Akcora, Carminati and Ferrari point out that the inclusion of a stranger (a friend of friends) into the social graph of a person implies the release of some personal information to the social graph of this stranger [11]. They propose a risk model to learn risk labels of strangers. This will enable them to detect individuals who are likely to violate privacy constraints. Our focus, here, is not on the individuals but identifying the state of the system that would lead to a violation.

Liu and Terzi address the privacy problem in OSNs from the user’s perspective [12]. They propose a model to compute a privacy score of a user. The privacy score increases with the sensitivity and visibility of the revealed information. Sensitivity is specific to a profile item while visibility of a profile item depends on the privacy settings of the user. It would be interesting to capture these contexts in the ontology of PriGuard and make inferences based on that.

Squicciarini et al. propose PriMa (Privacy Manager), which supports semi-automated generation of access rules according to the user’s privacy settings and the level of exposure of the user’s profile [13]. They further provide quantitative measurements for privacy violations. Quantifying violations is an interesting direction that we want to investigate further. Our use of an ontology can make it possible to infer the extents of the privacy violation, indicating its severity.

Carminati et al. study a semantic web based framework to manage access control in OSNs by generating authorization, administration and filtering policies [14]. They represent the OSN domain using an ontology but unlike us, they do not seem to use any OWL property characteristics; e.g. owl:SymmetricProperty. Similar to them, we use SWRL rules but we also augment that with OLP reasoning which enables us to express commitments and their corresponding violation statements.

Our architecture presents similarities to PROTOSS [1]. In that work, commitments were given as input and privacy violations were checked using model checking. However, here users are allowed to enter their privacy requirements using a user interface, which is then converted into commitments automatically. Next, the statements that would violate the commitments are generated. Finally, OLP is used to infer whether these violation statements hold in the current state of the online social network. Our work opens up interesting lines for future research. One interesting line is to enable PriGuard to proactively violate its commitments when necessary to provide a context-dependent privacy management. This will enable the system to behave correctly without asking the user
explicitly about privacy constraints. Another interesting line is to integrate this approach in relation to a real social network such as Facebook or community-based applications. This would require significant improvements on the ontology and possibly integration with existing community ontologies such as SIOC [15].

References