Negotiating Privacy Constraints in Online Social Networks*

Yavuz Mester, Nadin Kökciyan, and Pınar Yolum

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

Abstract. Privacy is a major concern of Web systems. Traditional Web systems employ static privacy agreements to notify its users of how their information will be used. Recent social networks allow users to specify some privacy concerns, thus providing a partially personalized privacy setting. However, still privacy violations are taking place because of different privacy concerns, based on context, audience, or content that cannot be enumerated by a user up front. Accordingly, we propose that privacy should be handled per post and on demand among all that might be affected. To realize this, we envision a multiagent system where each user in a social network is represented by an agent. When a user engages in an activity that could jeopardize a user's privacy (e.g., publishing a picture), agents of the users negotiate on the privacy concerns that will govern the content. We employ a negotiation protocol and use it to settle differences in privacy expectations. We develop a novel agent that represents its user's preferences semantically and reason on privacy concerns effectively. Execution of our agent on privacy scenarios from the literature show that our approach can handle and resolve realistic privacy violations before they occur.

1 Introduction

Privacy has long been accepted as an important concept in developing and running software. Typically, the software publishes its privacy agreement and the user accepts the agreement to use it. A similar pattern applies in online social networks, with additional handles to customize various parts of the policy. For example, a user can choose to share content with her friends but not anyone else, whereas a second user may choose to share a content with everyone in the system.

However, many times users' privacy requirements or expectations do not agree with each other. Contrary to typical software, in social networks, users can manipulate the content by tagging users, resharing content, and so on. Thus, content could become available to various audience without the consent of some of the users involved. Consider a user who wants to post a picture of herself with one of her friends. For the user, the posting could be harmless but her friend may be unhappy about it because the picture violates her privacy [2]. One common way of dealing with this in current social networks, such as Facebook, is to report the picture as inappropriate and let the user know that the person is unhappy about the content and would prefer it to be removed.

* This work has been supported by TUBITAK under grant 113E543.

As a result, the user is free to decide what to do about it. This requires a lot of human interaction online. More importantly, by this time, the picture has already been up and might well been viewed by many social network users [8]. This simple example is significant because it suggests various interesting properties for privacy protection:

Automation: Privacy protection calls for automated methods. Considering the volume of social media transactions that are done every hour, it is not plausible for humans to discuss every content that is related to them in person. Hence, an agent that represents a user is needed to keep track of its user's preferences and policies and act on behalf of them, accordingly.

Fairness: If a particular post is deemed private for a user, one approach is to take it down from the system completely. This all-or-nothing approach is simple but leaves the party who wants to keep the post up in a disadvantaged position. Instead, it is best if the users identify what is wrong with the post (in terms of privacy) and improve those aspects only. For example, if the text of a picture post is causing a privacy violation, the text can be removed while keeping the picture.

Concealment of privacy concerns: Privacy preferences may reveal personal information that should be kept private from others. That is, a user might say that she does not like the text of a post, but she does not need to identify why that is the case. Ideally, a proposed approach to privacy protection should keep users' privacy concerns private, without revealing users' privacy concerns as a whole.

Protection before exposure: Contrary to existing systems, such as Facebook, where a privacy violation is caught after it happens, the fact that a content is private should be identified before the content is put up online. Otherwise, there is a risk that the content reaches unintended audience. Hence, a proposed approach should be in place before the content is put up rather than after.

Accordingly, this paper proposes an agreement platform for privacy protection that addresses the above properties. One recent study has shown that many times invading a friend's privacy on a social network is accidental and the user actually invading the privacy is not aware of it. When the user is notified for the situation, many of them choose to put down a content rather than breaching their friend's privacy [13]. This naturally suggests that if the users could have agreed on the content before it went up, then many privacy leakages could have been avoided to begin with. We exploit this idea by developing a platform where the agents interact to reach a consensus on a post to be published. We assume that each user is helped by a software agent to manage her privacy. The agent is aware of the user's privacy concerns and expectations but also knows about the user's social network, such as her friends. When a user is about to post a new content, she delegates the task to her agent. The agent reasons on behalf of the user to decide which other users would be affected by the post and contacts those users' agents. The negotiation protocol we develop enables agents to discuss their users' constraints and agree on a suitable way to post the content such that none of the users' privacy is violated. We show the applicability of our approach on example scenarios from the literature.

The rest of this paper is organized as follows: Section 2 develops our negotiation framework and shows how the agents use it to reach agreements. Section 3 describes

our agent architecture with an emphasis on semantic representation and reasoning that it can do for its user. Section 4 evaluates our approach first by using different scenarios from the literature and then comparing it to prominent approaches from the literature based on the defined criteria above. Finally, Section 5 discusses our work in relation to recent work on negotiation and privacy.

2 Negotiation Framework

We propose a negotiation framework for privacy, PRINEGO, where users are represented by agents. An agent is responsible for keeping track of its user's privacy constraints and managing its user's privacy dealings with others. Before sharing a post, an agent decides if a post could violate other users' privacy (e.g., those involved in a picture). For this, an agent interacts with other agents to negotiate on a mutually acceptable post. When an agent receives a post request from a negotiator agent, it evaluates if the content would be acceptable for its user in terms of the media it contains, post audience, and so on. If the post request is not acceptable, that agent returns a rejection reason (e.g., audience should not contain a certain individual) to help the negotiator agent in revising its initial request. Once the negotiator agent collects all the rejection reasons from other agents, it revises its post request accordingly (e.g., removing the mentioned individual from the audience). A post negotiation terminates when all agents agree on a content or it reaches the maximum number of iterations set by the negotiator agent.

2.1 A Negotiation Protocol

Agents negotiate by employing a negotiation protocol. Our proposed negotiation protocol is in principle similar to existing negotiation protocols, which are used in e-commerce [7]. However, there are two major differences. First, contrary to negotiations in e-commerce, the utility of each offer is difficult to judge. For example, in e-commerce, a seller could expect a buyer to put a counter-offer with a lower price than that it actually made in the first place. Here, however it is not easy to compare two offers based on how private they are. Second, partly as a result of this, counter-offers are not formulated by the other agents but instead the negotiator agent collects the rejection reasons from other agents to update its initial offer.

There are two important components in our negotiation protocol: a *post request* and a *response*. A *post request* is essentially a post, which can include media, text, location, tagged people, audience and so on. Since this post has not been finalized yet (i.e., put up online), we consider this as being requested. Hence, agents try to negotiate on a post request. A post request must contain information about: (1) owner of the post request, (2) a post content (text, medium and so on) to be published, and (3) a target audience. A *response* is generated when an agent receives a post request. A response must contain: (1) owner of the response, (2) a response code ("Y" for accept, "N" for reject), and (3) a rejection reason, which is optional. We explain rejection reason types in Section 2.3. We use Example 1 that is inspired from Wishart *et al.* [15] as our running example.

Example 1 Alice would like to share a party picture with her friends. In this picture, Bob and Carol appear as well. However, both of them have some privacy concerns. Bob

does not want to show party pictures to his family as he thinks that they are embarrassing. Carol does not want Filipo to see this picture because she did not tell him that she is a bartender.

In Example 1, Alice instructs her agent to share a post by specifying her party picture where she tags Bob and Carol, and sets the audience to her friends. Alice's agent may decide to negotiate the post content with Bob and Carol as they are tagged in the picture. Then Alice's agent should initialize a *post request* (i.e., offer in our negotiation protocol) and send it to agents of Bob and Carol. These agents should then evaluate that post request with respect to their respective owner's privacy concerns, and create a *response*. Their responses should make it clear whether Bob and Carol accept or reject that post request, and ideally specify a reason in case of a rejection.

Algorithm 1 p NEGOTIATE$(p, altM, c, m)$

Require: p, the post request to be negotiated
Require: $altM$, the owner's choice of alternative media
Require: c, current iteration index
Require: m, maximum number of iterations
Return: the final post request resulting from the negotiation

```
1: if c > m then
2:     return p
3: else
4:     responses ← ∅
5:     for all agent ∈ DECIDEAGENTSTONEGOTIATE(p) do
6:         response ← agent.ASK(p)
7:         responses ← responses ∪ {response}
8:     end for
9:     if ∀r ∈ responses, r.responseCode = "Y" then
10:        return p
11:    else
12:        p′ ← REVISE(p, altM, c, m, responses)
13:        if p′ ≠ NULL then
14:            return NEGOTIATE(p′, altM, c + 1, m)
15:        else
16:            return NULL
17:        end if
18:    end if
19: end if
```

2.2 NEGOTIATE Algorithm

Algorithm 1 presents our algorithm for negotiating privacy constraints. The algorithm tries to finalize a post request iteratively, while negotiating with other agents. Basically, at each iteration, the negotiator agent decides which agents to negotiate with, collects their responses after that they evaluate the corresponding post request, and revises the

initial post request if any rejection response is received and the negotiation continues with the revised post request. Whenever all agents agree, our algorithm returns a post request to be shared on the social network. If an agreement cannot be reached after a predefined maximum number of iterations, then the negotiator agent returns the latest post request without negotiating it further.

Our algorithm takes four parameters as input: (1) the newly created post request (p), (2) the media selected by the owner as alternatives to the original medium ($altM$), (3) the current iteration index (c), (4) the maximum number of negotiation iterations (m). When the algorithm is first invoked, p should contain the owner's original post request. If the owner specifies any alternative media, $altM$ should contain them. c should be 1, and m can be configured by the owner. Our algorithm returns the final post request resulting from the negotiation. We use $responses$ to keep the incoming responses of other agents. If $responses$ contain a rejection, the negotiator agent inspects the reasons included in $responses$ to revise its initial post request (p')[1]. There are three auxiliary functions used in our algorithm, each of which can be realized differently by agents:

- DECIDEAGENTSTONEGOTIATE takes p and decides which agents to negotiate with. A privacy-conscious agent may decide to negotiate with every agent mentioned or included in the post text and medium components of a post request[2], whereas a more relaxed agent may skip some of them. We provide more details of DECIDEAGENTSTONEGOTIATE function in Section 3.2.
- ASK is used to ask agents to either accept or reject p. Each agent evaluates p according to their owners' privacy concerns. We explain how our semantic agent evaluates a post request in Section 3.3.
- REVISE is used to revise a post request with respect to $responses$ variable that includes rejection reasons. This function may also return $NULL$ at any iteration, which indicates a disagreement. The details of REVISE function are provided in Section 3.4.

NEGOTIATE is a recursive algorithm, which starts by checking whether the current iteration index (c) exceeds the allowed maximum number of iterations (m) (line 1). If this is the case, then the algorithm returns the current p without further evaluation (line 2). Otherwise, the algorithm negotiates p as follows. $responses$ variable is set to an empty set (line 4). In order to start the negotiation, the algorithm first computes which agents to negotiate with (line 5). Then, the algorithm asks each such agent to evaluate p (line 6), and adds each incoming response to $responses$ (line 7). After that, the algorithm checks whether there is an agreement (line 9). This is simply done by inspecting the response codes; a response code "Y" means an acceptance. If all agents accept p, then the algorithm returns p (line 10). Otherwise, the algorithm calls the auxiliary function REVISE to generate the revised post request p' (line 12). If p' is not $NULL$ (line 13), then p' has to be negotiated. Thus, the algorithm makes a recursive call by providing p' as the first argument (line 14), and c is incremented by 1 to specify the next

[1] p' is similar to "counter-offer" in other negotiation protocols with the difference that the negotiator agent makes it.

[2] We assume that mentioned or included people information can be inferred from post text and medium by employing textual and facial recognition algorithms.

iteration number (line 14). Otherwise (line 15), the algorithm returns $NULL$ (line 16), which indicates that no agreement has been reached.

2.3 Rejection Reasons

Agents may reject any post request and optionally provide the underneath reason along with the response. A rejection reason should specify the field of discomfort, such as audience, post text, and properties of the medium. Providing a reason for rejection is important because it gives the negotiator agent the opportunity to revise its request to respect the other agents' privacy constraints. Generally, an agent may have multiple reasons to reject a post request. We design our protocol to accept the reasons one by one to ease the processing. For example, a user may not want to reveal her party pictures to her colleagues. Whenever her agent is asked about a post request including such a medium and target audience, the agent should choose reasons related to medium or audience, but not both. This is a design decision to simplify the revision process. The post request is refined iteratively based on the rejection reasons collected in each iteration. Getting one reason from each agent may already result in a dramatic change in the issued post request (e.g. the medium can be altered and the new one may be tagged with a different set of people, such that different agents has to be consulted in the next iteration). The other rejection reasons may not be valid after the change, thus no need to be evaluated up front. If they are still valid, rejections may rise again in the next iterations and be evaluated. By adopting such a restriction, the privacy rules themselves have some degree of privacy as well. Only an implication of a privacy concern is exposed at a time. We provide more details about reason selection in Section 3.3.

3 A Semantic PRINEGO Agent

Our negotiation framework, PRINEGO, is open in the sense that it enables agents that are built by different vendors to operate easily. PRINEGO mainly requires agents to implement the negotiation protocol given in Algorithm 1 and conform with an agent skeleton. The agent skeleton is meant to describe the minimal set of functionalities that should be handled by an agent in order to qualify. The agents should be able to have the following functionalities: (1) Each agent should represent knowledge about the social network, in terms of relations among users, context of posts, user preferences, and so on. (2) Each agent should be able to implement the three methods of Algorithm 1; that is, DECIDEAGENTSTONEGOTIATE, ASK, and REVISE. Hence, an agent can start a post negotiation. (3) Each agent should be able to implement the EVALUATE algorithm so that it can evaluate a post request and decide if it is acceptable. The following sections explain how these are handled in our semantic PRINEGO agent.

3.1 A Social Network Ontology

We develop PRINEGO ontology to represent the social network as well as the privacy constraints of users. Each user has her own ontology that keeps information about her relationships, content and privacy concerns. An agent has access to its user's ontology and our negotiation framework uses ontologies to model interactions between agents.

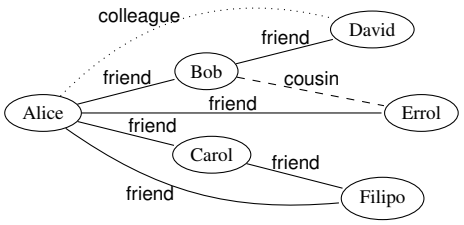

Fig. 1. Relation properties in PRINEGO ontology

The Social Network Domain A social network[3] consists of *users* who interact with each other by sharing some *content*. Each user is connected to another user via *relations* that are initiated and terminated by users. The social network domain is represented as an ontology using Web Ontology Language (OWL). In our negotiation framework, each `Agent` interacts with other agents by sending post requests. A `PostRequest` consists of a post that is intended to be seen by a target `Audience`, *hasAudience* is used to relate these two entities. An audience consists of its agents and we use *hasAudienceMember* to specify members of an audience. A `PostRequest` can contain some textual information `PostText` that may mention people or some `Location` (e.g., `Bar`). For this, we use *mentionsPerson* and *mentionsLocation* respectively. Moreover, a post request can include some `Medium` (e.g., `Picture` and `Video`). *hasMedium* is a property connecting `PostRequest` and `Medium` entities. A medium can give information about people who are in that medium, or location where that medium was taken. Such information is described by *includesPerson* and *includesLocation* properties respectively. Moreover, *isDisliked* is a boolean property that a user can use to dislike a medium.

Relation Properties In a social network, users are connected to each other via various relationships. Each user labels his social network using a set of relationships. We use *isConnectedTo* to describe relations between agents. This property only states that an agent is connected to another one. The sub-properties of *isConnectedTo* are defined to specify relations in a fine-grained way. *isColleagueOf*, *isFriendOf* and *isPartOfFamilyOf* are properties connecting agents to each other in the social network of a user. They are used to specify agents who are colleagues, friends and family respectively. Figure 1 depicts the social network for our motivating and evaluation examples. Here, a node represents an agent and the edges are the relations between agents. Each edge is labeled with a relation type. For simplicity, we use friend, cousin and colleague keywords in the figure. All edges are undirected as the relations between agents are symmetric. For example, Alice and Carol are friends of each other.

Context Context is an essential concept in understanding privacy. Many times, context is simply interpreted as location. The idea being that your context can be inferred

[3] We denote a `Concept` with text in mono-spaced format, a *relationship* with italic text, and an **:** `instance` with a colon followed by text in mono-spaced format.

from your location; e.g., if you are in a bar, then your context can be summarized as leisurely. Sometimes, the location is combined with time to approximate the context better. However, even location and time combined does not suffice to understand context [10]. Contrast a businessman going to a bar at night and the bartender being there at the same time. Time and location-wise their situation is identical, but most probably their contexts are different as the first one might be there to relax; the second to work.

This idea of context applies similarly on the content that is shared. A context of a picture might be radically different for two people who are in the picture. Hence, it is not enough to associate a context based on location only. Further information about a particular person needs to be factored in to come up with a context. Our ontology contains various `Context`s that can be associated with a post request for a given person. Hence, each agent infers context information according to the context semantics associated with its user. Following the above example, a medium taken in a bar may reveal `EatAndDrink` context for the businessman and `Work` context for the bartender. We use *isInContext* to associate context information to a medium.

Protocol Properties Each post request is owned by an agent that sends a post request to other agents. In a post negotiation, an agent may accept or reject a post request according to its user's privacy concerns. In the case where an agent *rejects* a post request, *rejectedIn* defines which concept (Medium, Audience or Post text) causes the rejection. The user can provide more detailed information about the rejection reasons by the use of *rejectedBecauseOf* and *rejectedBecauseOfDate* properties. In OWL, there is a distinction between object and data properties. Object properties connect pairs of concept instances while data properties connect instances with literals. Hence, *rejectedBecauseOf* is used if a post is rejected because of a concept (e.g., `Audience`) while we use *rejectedBecauseOfDate* to refer to a date literal (e.g., 2014-05-01). For example, a user can reject a post request because of an audience which includes undesired people or a medium where the user did not like herself. We explain protocol properties in detail in Section 3.3.

3.2 DECIDEAGENTSTONEGOTIATE Algorithm

In many negotiation problems, an agent already knows with whom to negotiate. However, in the social network domain, an agent can decide with whom to negotiate. Consider a picture of Alice and Bob taken in Charlie's office. When Alice's agent is considering of putting this up, it might find it useful to negotiate with Bob only but not with Charlie. A different agent might have found it necessary to also negotiate with Charlie; even though he is not in the picture himself, his personal information (e.g., workplace) will be revealed.

The choice to pick agents to negotiate with can also be context-dependent. In certain contexts, an agent can prefer to be more picky about privacy and attempt to negotiate with all that are part of a post. For example, if the post contains information that could reveal medical conditions about certain people, an agent might be more careful in getting permissions from others beforehand. In our case, our agent chooses to negotiate with everyone that is tagged in a post.

3.3 EVALUATE Algorithm

Once an agent receives a post request, it evaluates whether it complies with its user's privacy concerns. A post request includes various types of information: media, post text, target audience, and so on. Essentially, any of these points could be unacceptable for the agent that receives the request, which provides information about rejected attributes of the post request in case of a rejection. This can be considered as an explanation for the negotiator agent who revises its post request as accurately as possible so that it may be accepted by other agents in the following iterations.

Privacy Rules The privacy rules reflect the privacy concerns of a user. In a privacy rule, the user declares what type of post requests should be rejected at negotiation time. A privacy rule may depend on a specific location, context, relationship or any combination of these. Each agent is aware of the privacy concerns of its user. At negotiation time, the privacy rules are processed by an agent in order to decide how to respond to incoming post requests. For this, we augment PRINEGO with the privacy rules by the use of Semantic Web Rule Language (SWRL) rules for more expressiveness [6]. Description Logic reasoners can reason on ontologies augmented with SWRL rules. Each rule consists of a *Head* and a *Body* and is of the form *Body* $\Longrightarrow$ *Head*. Both the body and head consist of positive predicates. If all the predicates in the body are true, then the predicates in the head are also true. We use SWRL rules to specify Privacy Rules (P) in our negotiation framework. In a privacy rule, concepts and properties that are already defined in PRINEGO are used as predicates in rules. Moreover, we use protocol properties in the head of a rule. Here, we consider privacy rules to specify why a particular post request would be rejected. Any post request that is not rejected according to the privacy rules is automatically accepted by an agent. An agent can reject a post request without providing any reason. In this case, *rejects* is the only predicate observed in the head of a privacy rule. On the other hand, if an agent would like to give reasons about a rejection, then *rejectedIn* property is used to declare whether the rejection is caused by a medium, a post text or an audience. Furthermore, *rejectedBecauseOf* is used to specify more details about the rejection. For example, an audience can be rejected in a post request because of an undesired person in the audience. Or a medium can be rejected in a post request because of the location where the medium was taken. Here, we assume that privacy concerns of a user are already represented as privacy rules in her ontology.

In Example 1, Bob and Carol have privacy constraints that are shown in Table 1. P_{B_1} is one of Bob's privacy rules. It states that if a post request consists of a medium in `Party` context and has an audience member from Bob's family then Bob's agent rejects the post request because of the audience and this audience member becomes a rejected member in the corresponding post request. Carol's privacy rule (P_{C_1}) states that if a post request consists of a medium in `Work` context and if Filipo is an audience member in the audience of the post request then Carol's agent rejects the post request because of two reasons. Filipo is an undesired person in the audience hence her agent rejects the post request because of the audience and Filipo becomes a rejected member in the post request. The second reason is that Carol does not want to reveal information about her work, her agent rejects the post request because of the context as the medium discloses information about `Work` context.

Table 1. Privacy Rules (P) as SWRL Rules

P_{A_1}:	*hasAudience*(?postRequest, ?audience), *hasAudienceMember*(?audience, ?audienceMember), `Leisure`(?context), *hasMedium*(?postRequest, ?medium), *isInContext*(?medium, ?context), *isColleagueOf*(?audienceMember, `:alice`) $\implies$ *rejects*(`:alice`, ?postRequest), *rejectedIn*(?audience, ?postRequest), *rejectedBecauseOf*(?audience, ?audienceMember)
P_{B_1}:	*hasAudience*(?postRequest, ?audience), *hasAudienceMember*(?audience, ?audienceMember), `Party`(?context), *hasMedium*(?postRequest, ?medium), *isInContext*(?medium, ?context), *isPartOfFamilyOf*(`:bob`, ?audienceMember) $\implies$ *rejects*(`:bob`, ?postRequest), *rejectedIn*(?audience, ?postRequest), *rejectedBecauseOf*(?audience, ?audienceMember)
P_{B_2}:	*hasMedium*(?postRequest, ?medium), *isDisliked*(?medium, true) $\implies$ *rejects*(`:bob`, ?postRequest), *rejectedIn*(?medium, ?postRequest), *rejectedBecauseOf*(?medium, `:bob`)
P_{C_1}:	*hasAudience*(?postRequest, ?audience), *hasAudienceMember*(?audience, `:filipo`) *hasMedium*(?postRequest, ?medium), `Work`(?context), *isInContext*(?medium, ?context) $\implies$ *rejects*(`:carol`, ?postRequest) *rejectedIn*(?medium, ?postRequest), *rejectedBecauseOf*(?medium, ?context), *rejectedIn*(?audience, ?postRequest), *rejectedBecauseOf*(?audience, `:filipo`)
P_{C_2}:	*hasMedium*(?postRequest, ?medium), *dateTime*(?t), *equal*(?t, "2014-05-01T00:00:00Z"), *hasDateTaken*(?medium, ?t) $\implies$ *rejects*(`:carol`, ?postRequest), rejectedIn(?medium, ?postRequest), *rejectedBecauseOfDate*(?medium, ?t)

As shown in P_{C_1}, multiple components of a post request may be marked as rejected components in the head of a rule; e.g., medium and audience. Moreover, each component may be rejected because of one or more reasons. In P_{C_1}, each component is rejected because of one reason; e.g., context and audience member respectively. When multiple components of a post request are marked as rejected as the result of ontological reasoning, the agent has to decide on the component to share as a rejection reason. In this work, we consider reasons that would require minimal change in the initial post request. For this, we adopt a hierarchy between the components as follows: (i) audience, (ii) post text and (iii) medium. However, this behavior can vary from agent to agent. When P_{C_1} fires, the post request will be rejected by the reasoner because of the medium and the audience. Then, regarding our hierarchy, `:carol` first checks whether there is any rejection caused by the audience and she finds one. Hence, it rejects the audience because of `:filipo`. In another words, `:carol` will share this reason (and not the context reason) with the negotiator agent.

3.4 REVISE Algorithm

In case a post request is rejected by one or more agents, the negotiator agent needs to revise the post request with respect to the responses received. However, each agent is free to decide if it wants to credit a rejection reason or to ignore it. This is correlated with the fact that some people may have more respect to others' privacy, whereas some may be more reluctant to it. Moreover, the way the agents honor a rejection reason can vary as well. For example, consider a case where an agent rejects a post request because its owner is not pleased with his appearance in the picture. The negotiator agent can either alter the picture or remove it completely. In case it is altered, the new picture may or may not include the previously rejecting owner. The agents have the freedom of revising as they see appropriate.

An important thing to note is that the rejection reasons cannot possibly conflict with each other. This is guaranteed by two design decisions included in PRINEGO. First, the privacy concerns define only the cases where an agent should reject a post request. Second, the agents cannot reject a post request because it does not have some desired attributes (e.g., the audience should have also included some other person). In such a case, the agent may initiate another negotiation with a new post request that puts that person into the audience; i.e., resharing through negotiation is possible at any time. In this manner, a negotiator agent does not need to worry about conflicts and can handle rejection reasons in a suitable manner. Our semantic agent honors every rejection reason since we know that it will not create any conflict with other reasons. Our algorithm discards undesired audience members if any from the audience, and removes the text content if rejected by some agent. In Section 4.1, we show how our approach works step by step on Example 1.

When the medium component is rejected, our algorithm inspects all of the rejection reasons gathered during the negotiation, and clusters them under these five groupings: undesired included people, locations, and media dates, self-disliked mediums, context-disliked mediums. Then, our algorithm inspects each medium in the alternative media selection of the owner to find a suitable medium. The suitable medium should comply with other agents' privacy concerns. The approach we use to select a medium comes in handy for some cases and one such example is carried out in Section 4.2. After a revision, our algorithm checks whether the resulting post request is still reasonable. A post request is not reasonable if it does not have any audience or a content (neither text nor medium content). Thus, it is possible to result in a disagreement not because the rejection reasons may conflict with each other, but because the sanity of the post request may be lost after the revision process.

Our algorithm can be extended so that all the iterations for all the negotiations can be taken into consideration. Machine learning methods may suit here to estimate the possibility of a post request to be rejected with the help of past experience. Such enhancements may lead to intelligent revisions, and thus increase the chance and the speed of converging to an agreement dramatically. For example, a negotiator agent can learn the behavior of another agent (e.g., an agent that accepts every post request) and then decides not to ask that agent as it already knows that it will accept.

4 Evaluation

We implement the semantic PRINEGO agent that was detailed in Section 3 using Java and the Spring framework. We simulate the communication between the semantic agents with RESTful web services such that each agent has exactly two web services: one to be asked by the owner to negotiate a desired post, and the other to be asked by other agents to evaluate a post request. We use the OWL API [5] to work with ontologies and Pellet as the reasoner [11]. Each semantic agent uses an ontology as described in Section 3.1. The ontology is used as a knowledge base and also contributes into the reasoning of privacy rules (Section 3.3). When asked to evaluate a post request via a web service call, the agent first puts the incoming post request into its ontology and then makes ontological reasoning on the post request against the owner's privacy constraints.

The agent can run from a Web browser and additionally it can run as an Android application that we have developed. Our mobile application is integrated with a real social network, Facebook. A user logs into our application by providing her Facebook credentials. To initialize a post request, she selects a picture from her device and tags some of her Facebook friends. Moreover, she sets an audience for her post request. Our application starts a negotiation between the negotiator agent and the agents of tagged users. Once the negotiation is done, the negotiator agent shares the resulting post.

We evaluate our proposed approach using three scenarios from the literature. For this, we create six test users on Facebook, each of which has one of the PRINEGO agents we created, namely `:alice`, `:bob`, `:carol`, `:david`, `:errol`, and `:filipo`. We log into our mobile application with the appropriate test user and share a post request as suggested by each scenario. For each post request, PRINEGO is used for negotiating with other agents and the negotiator agent shares the agreed upon post requests on Facebook. Then, we examine the results to see how the agents reach agreements. We provide a walk-through examination of PRINEGO on Example 1 in Section 4.1 and then discuss two more scenarios in Section 4.2.

4.1 A Walk-through of PRINEGO

In Example 1, Alice wants to share a post request p, which includes a party picture. Bob and Carol are tagged in this picture. The audience of the post request is set to Bob, Carol, Errol and Filipo. In order to publish the post request, Alice's agent (`:alice`) would like to negotiate with other agents. `:alice` does not have any other alternative media to share and is allowed to finalize a negotiation in five iterations as configured by Alice. `:alice` invokes NEGOTIATE (p,[],1,5). It decides to negotiate with all the tagged agents in the post request, namely `:bob` and `:carol`, as a result of DECIDEAGENTSTONEGOTIATE (p). Then, `:alice` asks both of them by invoking ASK (p). `:bob` checks whether p is compatible with its privacy rules P_{B_1} and P_{B_2}. It notices that p includes a picture taken in `Party` context and Errol, who is part of Bob's family, is included in the audience of p. This condition fires one of the rules, P_{B_1}. As a consequence, `:bob` rejects p, the audience is rejected in p because of Errol, who is in the audience. `:alice` keeps `:bob`'s rejection reason $\{$-Errol$\}$. On the other hand, `:carol` has two privacy rules P_{C_1} and P_{C_2}. `:carol` computes that p violates Carol's privacy rule P_{C_1}. p includes a picture, which is in `Work` context for Carol, moreover Filipo is in the audience of p hence `:carol` rejects p because of two reasons. It rejects p because of: (i) the audience, which includes Filipo, (ii) the `Work` context. `:carol` prioritizes these reasons (see Section 3.3) and chooses $\{$-Filipo$\}$ as the rejection reason. The set of rejected reasons then becomes $\{$-Errol, -Filipo$\}$, and `:alice` wants to make a revision on p. `:alice` invokes REVISE (p,[],1,5,$\{$-Errol, -Filipo$\}$), which in turn revises p by removing the undesired people from the audience. The audience of p' (the revised post request) is set to Bob and Carol. `:alice` invokes NEGOTIATE (p',[],2,5) to negotiate p'. `:alice` asks `:bob` and `:carol`, and finally both of them accept p', because neither of their rules are dictating otherwise. Thus, `:alice` finalizes the negotiation in two iterations and the resulting post request is harmful for neither Bob's nor Carol's privacy.

Table 2. Iteration steps for Example 2 where :bob starts the negotiation

Iter.	Content	Audience	Asked Agents	Evaluate	Response
1	beach picture of Alice	A, D, E	:alice	:alice $\rightarrow P_{A_1}$	:alice $\rightarrow$ -David
2	beach picture of Alice	A, E	:alice	:alice $\rightarrow$ N/A	:alice $\rightarrow$ ✓

Table 3. Iteration steps for Example 3 where :alice starts the negotiation

Iter.	Content	Audience	Asked Agents	Evaluate	Response
1	pic_1	B, C, E, F	:carol	:carol $\rightarrow P_{C_2}$	:carol $\rightarrow$ -date
2	pic_2	B, C, E, F	:carol, :bob	:carol $\rightarrow$ N/A, :bob $\rightarrow P_{B_2}$	:carol $\rightarrow$ ✓, :bob $\rightarrow$ -self
3	pic_3	B, C, E, F	:carol, :bob	:carol $\rightarrow$ N/A, :bob $\rightarrow$ N/A	:carol $\rightarrow$ ✓, :bob $\rightarrow$ ✓

4.2 PRINEGO in Action

In this section, we show how our semantic agents negotiate with each other in Example 2 and Example 3. Table 2 and Table 3 summarize the negotiation steps. We use initial letters for users in the audience description (e.g., A represents Alice). Example 2 is inspired from the work of Squicciarini *et al.* [12]. In this example, a user's privacy is compromised because of a friend sharing some content about this user.

Example 2 Alice and David are friends of Bob. Bob organizes a party where Alice and David meet each other. David offers Alice a job in his company and she accepts. One day Bob shares a beach picture of Alice with his connections. David could view this picture. Alice is worried about jeopardizing her position in David's company and asks Bob to remove her beach picture.

In Example 2, :bob starts the negotiation as Bob wants to share a beach picture of Alice. In the first iteration, :bob asks :alice, which rejects this post request according to P_{A_1} with the reason that the audience includes David. Then, :bob revises the post request by removing David from the audience (the content remains untouched) and sends the updated post request to :alice again. This time :alice accepts, and :bob terminates the negotiation as an agreement is reached.

Example 3 Alice would like to share a picture, which was taken on May 1, 2014 with her friends. Carol is tagged in this picture and she does not want to show any picture that was taken on May 1, 2014. Alice decides on sharing another picture where Carol and Bob are both tagged. This time Bob does not like himself in this picture. Alice finds another picture of Carol and Bob and finally they all agree to share.

Differently in Example 3, the rejection reasons are about the medium content of the post request, and the negotiator agent makes use of the alternative media. In the first iteration, :carol rejects the picture because of its date taken. Then, :alice alters the picture and the new picture includes Carol as well as Bob. Thus this time :alice chooses to ask not only :carol but also :bob. However, :bob rejects since Bob marks the picture as self-disliked, although the new picture does not violate :carol's privacy constraints. :alice asks another picture of Bob and Carol in the third iteration and finally both of them accept. The negotiation terminates with success.

5 Discussion

We have developed a framework in which agents can negotiate their privacy constraints. Our framework can be used by any agent that adheres to the agent skeleton. Our particular agent makes use of ontologies and semantic rules to reason on its user's privacy constraints. Contrary to typical negotiation frameworks, only one agent proposes offers and the other agents comment on the offer by approving or giving reasons for disapproving. Agent itself collects the reasons and creates a new offer if necessary.

Particularly, in various negotiation frameworks, modeling the opponent and learning from that has been an important concept. Aydogan and Yolum have shown how agents can negotiate on service descriptions and learn each other's service preferences over interactions [3]. In this work, we have not studied learning aspects. However, the general framework is suitable for building learning agents. Particularly, since the history of reasons for rejecting a post request is being maintained, a learning algorithm can generalize over the reasons. The idea of privacy negotiation has been briefly studied in the context of e-commerce and web-based applications. Bennicke *et al.* develop an add-on to P3P to negotiate service properties of a website [4]. Their negotiation scheme is based on users' predefined demands without any semantics as we have done here. Similarly, Walker *et al.* propose a privacy negotiation protocol to increase the flexibility of P3P through privacy negotiations [14]. Their negotiation protocol is expected to terminate within a finite number of iterations and produce Pareto-optimal policies that are fair to both the client and the server. However, their understanding of privacy is limited to P3P and do not consider context-based and network-based privacy aspects.

The following three systems are important to consider regarding our purposes. Primma-Viewer [15] is a privacy-aware content sharing application running on Facebook. It is a collaborative approach where privacy policies are managed by co-owners of the shared content. Briefly, a user uploads a content and initializes a privacy policy where she defines who can access this content. She can invite others to edit the privacy policy together. Facebook [1] gives its users the ability to report a specific post shared by others. The user can simply ask her friend to take the post down or provide more information about why she would like to report the post by selecting predefined text (e.g., "I am in this photo and I do not like it") or messaging her friend by using her own words. FaceBlock [9] is a tool that converts regular pictures taken by a Google Glass into privacy-aware pictures. For this, a user specifies context-based privacy policies that include the conditions under which her face should be blurred. These policies are automatically shared with Glass users who enforce the received policies before publishing a picture. Table 4 compares our approach with these three systems, namely Primma-Viewer [15], Facebook [1], and FaceBlock [9] using the four criteria we defined in Section 1.

– *Automation* refers to the fact that the system actually acts on behalf of the user to align privacy constraints. Since our approach is agent-based, it is automated. The agent negotiates the users' constraints and reaches a conclusion on behalf of the user. Primma-Viewer allows users themselves to script a joint policy collaboratively; hence it is not automated. Facebook allows users to report conflicts and deals with them on individual basis. Again, the interactions are done by the users.

Table 4. Comparison of Privacy Criteria

Software	Automation	Fairness	Concealment	Protection
PRINEGO	✓	✓	✓	✓
Primma-Viewer	✗	✓	✗	✓
Facebook	✗	✗	✓	✗
FaceBlock	✓	✗	✗	✓

FaceBlock, on the other hand, acts on behalf of a user to detect potential users that can take pictures and interacts accordingly. Hence, their solution is automated.

– *Fairness* refers to how satisfied users are with an agreement. For example, if a user has to remove an entire post because another user does not like it, it is not fair to the initial person. Accordingly, our approach tries to negotiate the details of the post so that only the relevant constraints are resolved. Thus, we say it is fair. In the same spirit, Primma-Viewer allows user to put together a policy so that various constraints are resolved cooperatively. In Facebook, however, the only option is to request a post to be taken off all together without worrying about what details are actually violating the user's privacy. FaceBlock is more fair than Facebook in that the content is not taken off the system but the person that has privacy concerns is blurred in the picture. However, still the user might be unhappy about other details of the picture, such as its date or context, but that cannot be specified or resolved. Hence, we consider both Facebook and FaceBlock as unfair.

– *Concealment* refers to whether the users' privacy constraints become known by other users. A user might want to say in what ways a post violates her privacy but may not want to express more. Our approach enables this by allowing users to respond to post requests with reasons. For example, a user might say that she does not want Alice to see her pictures but does not need to say why this is the case. In this respect, our approach conceals the actual privacy concerns. Contrast this with Primma-Viewer, where the users together construct policies. In that case, all users will be aware of privacy rules that are important for each one of them; thus, the privacy constraints will not be concealed. In Facebook, everyone's privacy setting is known only by the individual and thus conceals privacy concerns from others. In FaceBlock, the privacy rules are sent explicitly to the user that is planning to put up a picture. The user evaluates the privacy rules of others to decide if the picture would bother them. This contrasts with our approach where privacy rules are private and only evaluated by the owner of the rules.

– *Protection* refers to when a system deals with privacy violations. In our work, we resolve privacy concerns before a content is posted; hence privacy is protected up front. The same holds for Primma-Viewer since the joint policy is created up front, whenever a content is posted, it will respect everyone's privacy policy. In Facebook, this is the opposite. After a content is posted, if it violates someone's privacy, that individual complains and requests it to be taken off. At this time, many people might have already seen the content. Finally, FaceBlock enforces the privacy rules before the content is put up; hence protects the privacy.

In our future work, we first plan to have adapting agents that can learn the privacy sensitivities of other agents, in terms of contexts or individuals so that the negotiations can be handled faster. For example, assume an agent never wants her protest pictures to be shown online. If her friends' agents learn about this, they can stop tagging her in such pictures. Second, we want to improve our revise algorithm to generate better offers by the use of argumentation approach so that agents can convince each other to negotiate on a content.

References

1. Facebook web site. http://www.facebook.com. Accessed: 2014-11-11.
2. L. Andrews. *I know who you are and I saw what you did: Social networks and the death of privacy*. Simon and Schuster, 2012.
3. R. Aydogan and P. Yolum. Learning opponent's preferences for effective negotiation: an approach based on concept learning. *Autonomous Agents and Multi-Agent Systems*, 24(1):104–140, 2012.
4. M. Bennicke and P. Langendorfer. Towards automatic negotiation of privacy contracts for internet services. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 319–324. IEEE, 2003.
5. M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.
6. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*, 21:79, 2004.
7. N. R. Jennings, A. R. L. P. Faratin, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.
8. O. Kafalı, A. Günay, and P. Yolum. Detecting and predicting privacy violations in online social networks. *Distributed and Parallel Databases*, 32(1):161–190, 2014.
9. P. Pappachan, R. Yus, P. K. Das, T. Finin, E. Mena, and A. Joshi. A Semantic Context-Aware Privacy Model for FaceBlock. In *Second International Workshop on Society, Privacy and the Semantic Web - Policy and Technology (PrivOn), Riva del Garda (Italy)*, October 2014.
10. A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.
11. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
12. A. C. Squicciarini, H. Xu, and X. L. Zhang. Cope: Enabling collaborative privacy management in online social networks. *J. Am. Soc. Inf. Sci. Technol.*, 62(3):521–534, Mar. 2011.
13. M. G. Stewart. How giant websites design for you (and a billion others, too). TED Talk, 2014.
14. D. D. Walker, E. G. Mercer, and K. E. Seamons. Or best offer: A privacy policy negotiation protocol. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 173–180. IEEE, 2008.
15. R. Wishart, D. Corapi, S. Marinovic, and M. Sloman. Collaborative privacy policy authoring in a social networking context. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, pages 1–8, Washington, DC, USA, 2010. IEEE Computer Society.