NEGOTIATING PRIVACY CONSTRAINTS IN ONLINE SOCIAL NETWORKS

by

Yavuz Mester

B.S., Computer Engineering, Bilkent University, 2009

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2015

NEGOTIATING PRIVACY CONSTRAINTS IN ONLINE SOCIAL NETWORKS

APPROVED BY:

Assoc. Prof. Pınar Yolum (Thesis Supervisor)	.....
Assoc. Prof. Şule Gündüz Öğüdücü	.....
Assist. Prof. Arzucan Özgür	.....

DATE OF APPROVAL: 14.01.2015

ACKNOWLEDGEMENTS

First and foremost, I would like to offer my sincerest gratitude to my thesis supervisor Assoc. Prof. Pınar Yolum for her continous encouragement, guidance, teaching and support. She inspired me greatly to work in this thesis.

I would like to thank Assist. Prof. Arzucan Özgür and Assoc. Prof. Şule Gündüz Öğüdücü for being in my thesis committee and provide valuable suggestions.

I would also like to thank Nadin Kökciyan for her support and tremendous contributions. I am deeply indebted to my family for their support in every aspects of my life. I would like to offer my special thanks to Melike Korucuoğlu for encouraging me all the way.

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

ABSTRACT

NEGOTIATING PRIVACY CONSTRAINTS IN ONLINE SOCIAL NETWORKS

Privacy is a major concern of Web systems. Traditional Web systems employ static privacy policies 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.

ÖZET

ÇEVRİMİÇİ SOSYAL AĞLARDA MAHREMİYET SINIRLARININ MÜZAKERESİ

Mahremiyet Web tabanlı sistemler için önemli bir konudur. Web sistemleri mahremiyet politikalarını yayımlayarak kullanıcı bilgilerine karşı olan tutumlarını açıklarlar. Kullanıcıların mahremiyet gereksinimlerinin birbirinden çok farklı olabilmesine rağmen, Web sitelerindeki mahremiyet politikaları büyük ölçüde sabittir ve kullanıcı bazında değişmez. Çevrimiçi sosyal ağlar kullanıcılarına kısmi olarak kişisel mahremiyet ayarları yapma imkanı tanısa da sosyal ağlarda mahremiyet ihlali sıklıkla karşılaşılan bir durumdur. Bunun nedeni bağlam, izleyici ve içerik temelli mahremiyet endişelerinin kullanıcı tarafından yeteri kadar detaylı girilmesine olanak sağlanmamasıdır. Bu sorunu çözmek için, her paylaşım öncesi paylaşımın ilgilendirebileceği tarafların otomatik olarak müzakere etmesine olanak sağlayan bir yapı öneriyoruz. Buna yönelik olarak, sosyal ağlardaki her bir kullanıcının kendisine ait bir etmen yazılımı olmasını öngörüyoruz. Bir sosyal ağ kullanıcısı başka bir kullanıcının mahremiyetini ihlal edebilecek bir aktivitede bulunacağı zaman (ör. fotoğraf paylaşımı), ilgili kullanıcıların etmen yazılımları sahiplerinin mahremiyetini koruyacak şekilde müzakere ederek bu aktivitenin oluşturacağı mahremiyet ihlalini daha en başından önlemiş olacaklardır. Bu çalışma ile bahsedilen etmen yazılımların iletişim kurallarını belirleyen bir müzakere protokolü geliştirilmiştir. Ayrıca bu protokole uyan örnek bir etmen yazılım ontoloji teknolojisi kullanılarak geliştirilmiş olup, bu etmen yazılımda kullanıcı tercihlerinin nasıl modellendiği ve mahremiyet sınırlarının nasıl etkin bir biçimde müzakere edilebileceği gösterilmiştir. Yazdığımız etmen yazılım üzerinde yaptığımız değerlendirmeler, sunduğumuz yöntemin sosyal ağlarda yaşanan bazı mahremiyet ihlallerini daha gerçekleşmeden önleyebileceğini göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$altM$	Alternative media
c	Current iteration index
m	Maximum number of iterations
P	Privacy rules
p	Post request
p'	Revised post request
P_{A_1}	Alice's privacy rule
P_{B_1}	Bob's first privacy rule
P_{B_2}	Bob's second privacy rule
P_{C_1}	Carol's first privacy rule
P_{C_2}	Carol's second privacy rule

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
FIPA	Foundation for Intelligent Physical Agents
FTC	Federal Trade Commission
JADE	Java Agent DEvelopment framework
JSON	JavaScript Object Notation
OBO	Or Best Offer
OECD	Organization for Economic Co-operation and Development
OWL	Web Ontology Language
P3P	Platform for Privacy Preferences
PriGuard	Privacy Guard
PriNego	Privacy Negotiation Framework
RDF	Resource Description Framework
SQL	Structured Query Language
SWRL	Semantic Web Rule Language

1. INTRODUCTION

1.1. Motivation

Information privacy is regarded as the right of individuals to determine for themselves when, how, and to what extent information about them is communicated to others [3]. As the Internet evolves, many Internet users have become concerned about their online privacy. The primary concern is that Web sites have the ability to record personal data of their users, and use and distribute them for their own external purposes such as marketing, selling and so on. Personal data may include, for example, name, address, phone number, location, credit card number or any other information relating to the individual. To ease privacy concerns, many Web sites publish an online *privacy policy* that outlines how the company keeps information private, how the information is shared and why it is collected. However, privacy policies alone are not sufficient for user privacy, as they do have some major drawbacks.

First, there are readability and usability problems associated with privacy policies. Jensen *et al.* argue that privacy policies are in most cases the users' only source of information, and therefore present an important challenge in terms of Human Computer Interaction (HCI); how to convey a lot of complicated but critical information without overwhelming users [4]. However, the work of Jensen *et al.* show that privacy policies are hard to read, and are hardly understood by a large segment of the population. Besides, privacy policies are subject to change; the companies may revise them from time to time. Jensen *et al.* point out that Web sites fail to provide adequate notification of changes; and thus users must, if they are serious about protecting their privacy, check the privacy policy every time they visit a Web site. The readability problem and the need to repeatedly check the privacy policies put too much burden on the user. In fact, a study notes that if a person actually bothers to read all the privacy policies he encounters on a daily basis, it will take him 244 hours per year - or about 30 workdays [5]. As a result, the users seldom consult the privacy policies [6], and the policy content does not actually support rational privacy decision making [7].

Second drawback of privacy policies is that they merely constitute privacy promises, and as such do not guarantee that Web sites adhere to their declared privacy practices. To this end, *privacy enforcement authorities* — public bodies that are responsible for enforcing laws protecting privacy, and that has powers to conduct investigations or pursue enforcement proceedings — should be established and maintained as the Organization for Economic Co-operation and Development (OECD) [8] suggests in their recent framework of guidelines regarding privacy [9]. An example privacy enforcement authority is the Federal Trade Commission (FTC) [10], which is an independent agency of the United States government responsible for protecting American consumers. However, in a rapidly changing online ecosystem, studies show that government regulation is not sufficient to protect consumers [11]; therefore, technological means have to be devised and adopted to enforce privacy policies.

Finally, privacy policies are static and they do not offer much customization on an individual level. However, one size does not fit all, as Web users' *privacy constraints* may differ greatly. Privacy constraints define the boundaries related to the gathering, use, storing and relaying of personal data by service providers. Consider an e-commerce site that stores its users' online activities such as the products they look at or buy in order to recommend the users products they may like. While some users may be content with this approach and the recommendations the site is offering, another user may have a privacy constraint as his online activities shall not be recorded. However, Web sites employ "accept it or leave it" approach as for their privacy policies, which is too rigid and deprives the users of acting according to their privacy concerns. Users should be able to express their privacy constraints to service providers and a *privacy agreement* has to be established prior to using online services. A privacy agreement represents a contract between the user and the service provider, and at the same time, a perceived alignment between the user's privacy constraints and the service provider's future treatment on that user's data. Conceptually, privacy agreements are more formal and more legal (enforceable) than privacy policies, which are simply promises. Approaches and mechanisms have to be devised in order to reach and use privacy agreements in the online world.

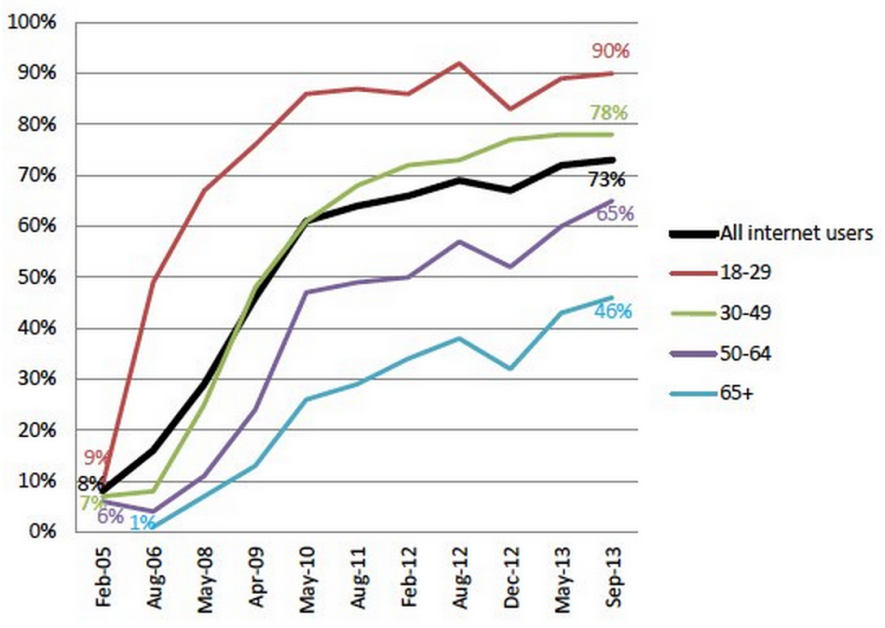

Figure 1.1. The rise of social networking site use by age group, 2005-2013: % of Internet users in each age group who use social networking sites, over time [1].

Apart from these issues, online social networks become very popular from teenagers to adults over the years (see Figure 1.1), and introduce new concerns, as every new technology does. The social networks change our world by bringing pervasive connectivity to others. The users are encouraged to share anything and everything with their network, by posting information about themselves and others. Each user has a profile where they can include their name, profile photo, physical and email addresses, occupation, and so on. And each user may post, for example, a photo/video including only himself, a group photo/video possibly including his friends or outsiders, a link to another Web site, a text message about himself or his opinions, or a comment to an existing post. The additional concern social networks bring is that user-to-user privacy. Each post is meant to be seen only by certain people; the users are very concerned about which other users will see the content as unintended consequences may arise if the content reaches undesirable audience. For example, in 2007, a man purchased a diamond ring for his girlfriend to propose; but the surprise is ruined as the Facebook broadcasts the purchase to his network including his girlfriend. This caused Facebook a big lawsuit [12]. Secondly, employees getting fired due to questionable posts in social media is on the rise [13], i.e., such posts better not revealed to colleagues or bosses. In

this regard, the social networks allow the user to specify whom to share the content with each time they post (e.g., see Figure 1.2). However the incidents prove that such a control alone is not sufficient, as in reality the content is not managed by a single user, rather being discussed and reshared by other social network users, and easily become available to various audience without the consent of some of the users involved [14]. Consequently, ensuring user privacy in online social networks is a complex problem by their very nature [15] [16].

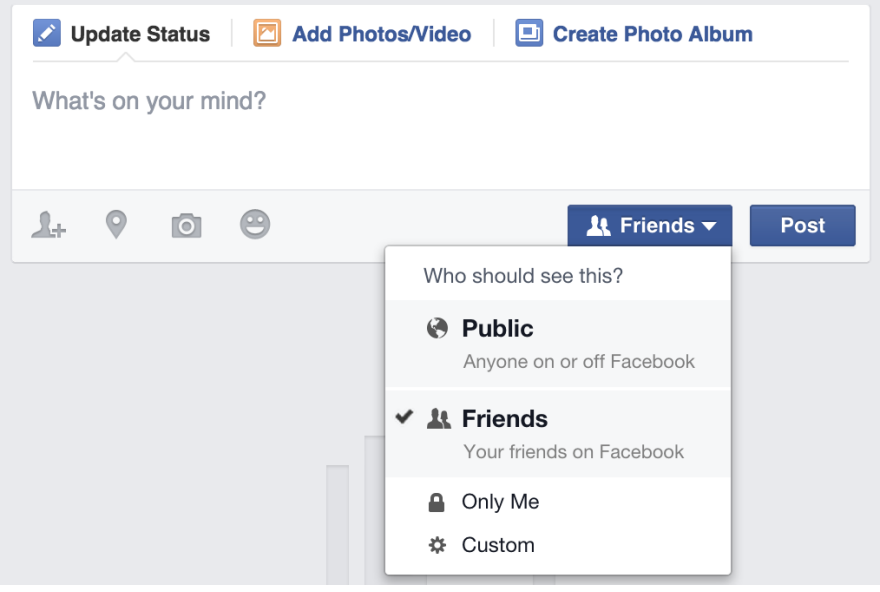

Figure 1.2. "Who should see this?" option while creating a new post on Facebook [2].

Many times users' privacy requirements or expectations do not agree with each other. While this is not a problem in regular software, where transactions are independent from each other, it creates concerns in social networks. 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 [17]. 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. 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 have been viewed by many social network users [18].

This simple example is significant because it suggests various interesting properties for privacy protection in online social networks:

Automation: Privacy protection in social networks 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 preferences: 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 preferences 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.

1.2. Our Focus & Proposed Solution

"Privacy is a complex beast" as Clarke states [19]. Privacy has multiple dimensions that can hardly be throughly addressed in any one effort [20]. Some of the possible improvement areas are explained in Section 1.1. To sum up, privacy policies are hard to read, ever changing and hardly useful, their enforcement requires both technical and legislative measures, and they have be converted into personalized privacy agreements as one size does not fit all. Moreover, with the rise of online social networks, new ways have to be created to give social network users more control on their private data. Our focus is to create a social network privacy protection approach to govern the dissemination of personal content through users by addressing all of the aforementioned properties. To the best of our knowledge, such an approach meeting the defined criteria is not devised yet.

Accordingly, our work proposes an agreement platform for privacy protection. 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 [21]. 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 their 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.

1.3. Thesis Overview

Chapter 2 develops our negotiation protocol and shows how the agents use it to reach agreements. Chapter 3 describes our semantic agent architecture with an emphasis on knowledge representation and reasoning that it can do for its user. We implement our PriNego protocol by creating a negotiation framework and a semantic PriNego agent. Our implementation details and considerations along the way are depicted in Chapter 4. Chapter 5 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, Chapter 6 discusses our work in relation to recent work on privacy and negotiation.

2. A PRIVACY NEGOTIATION PROTOCOL FOR ONLINE SOCIAL NETWORKS: PriNego

Agents negotiate by employing our privacy negotiation protocol called PriNego. The protocol we create is in principle similar to existing negotiation protocols that are used in e-commerce [22]. In e-commerce, negotiations are primarily on objects of the real world and the goal for the negotiation can be considered as minimizing the object's price from the buyer's perspective, and maximising profit from the seller's perspective. Both the buyer and the seller have to pursue some negotiation strategies, which are private approaches the parties use to hopefully help them achieve their goals. The negotiating parties should also adopt an evaluation mechanism to determine whether an offer at a time is enough for time. During the negotiations both parties may concede from their initial requests and finally either reach an agreement or disagreement. However, PriNego dictates a different negotiation scheme than the ones in e-commerce domain because there are basically two major differences. First, contrary to the 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 asked agent but instead reasons on why the made offer is not acceptable is provided to the negotiator agent. This will hopefully enable the negotiator agent to formulate its offer in a way that will be acceptable for other agents.

PriNego defines message constructs to be exchanged by the agents and also rules for the negotiation. Each agent may follow their own private negotiation strategies as long as they obey the negotiation rules and produce valid messages that are aligned with our definitions.

2.1. Message Constructs

There are two important representations in our negotiation protocol. The first one is a *post request*. This is essentially a post, which can include media, text, location, tagged people, audience and so on. Since the post has not been finalized (i.e., put up online), we consider this as being requested. This is the main object on which the negotiation is taking place. We require that a post request must contain information about: (i) the owner of the post request, (ii) either or both of a text and a medium content to be published, and (iii) the target audience. The second representation is a *response*. When an agent receives a post request, it must generate a response. The response must contain: (i) the owner of the response, (ii) a response code ("Y" for accept, "N" for reject), and (iii) a rejection reason if provided. Rejection reason types are explained in detail in Section 2.3.

We use Example 2.1 that is inspired from Wishart *et al.* [23] as our running example.

Example 2.1. *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 2.1, Alice should input her agent the party picture, the tagged people Bob and Carol, and the audience as her friends. Alice's agent negotiates the post content with Bob and Carol as they are tagged in the picture. Then Alice's agent should transform Alice's wishes to a *post request*, which represents an offer in a negotiation protocol, and send it to Bob and Carol for their evaluation. Bob's and Carol's agents should then evaluate the 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 the post request, and ideally specify a reason in case of a rejection. To this end, the negotiator agent broadcasts a post request to the agents to

negotiate with, and the receiving agents send a response in return.

2.2. Negotiate Algorithm

Algorithm 2.1 is for negotiating privacy constraints Thus, the algorithm revises an initial post request iteratively, while negotiating with other agents. The hope is that making slight changes on the post requests such as removing a person from target audience or trying another medium, may result in a post request on which each negotiating agent agrees. Basically, in each iteration the negotiator agent decides which agents to negotiate with, asks their responses about the post request, and if some agents reject then revises the post request and continues the negotiation. Whenever all the agents agree, the algorithm returns the agreed upon post request. 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.

The algorithm takes four parameters as input: (i) the newly created post request (p), (ii) the media selected by the owner as alternatives to the original medium ($altM$), (iii) the current iteration index (c), (iv) 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. The algorithm returns the final post request resulting from the negotiation.

In the algorithm, we use *responses* to keep the incoming responses from the asked agents. In case *responses* contains a rejection, the negotiator agent inspects the reasons included in *responses*, and produces a revised post request p' [1] .

There are three auxiliary functions used in the algorithm, each of which can be realized differently by the agents:

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

DecideAgentsToNegotiateWith 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 text and medium[2] components of p, whereas a more relaxed agent may skip some of them. The details of DecideAgentsToNegotiateWith function is provided in Section 3.2.

Ask is used to ask agents to either accept or reject p. Each agent will evaluate the request according to their owners' privacy concerns. The details of how our semantic agent evaluates a post request is explained in Section 3.3.

Revise is the function to revise a post request with respect to *responses* that include rejection reasons. This function may also return $NULL$ at any iteration to indicate a disagreement. The details of Revise function is explained in Section 3.4.

Negotiate is a recursive algorithm with the details below. It starts by checking whether the current iteration index (c) exceeded 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. To begin with, *responses* is set to an empty set (line 4). In order to start the negotiation, the algorithm first decides which agents to negotiate with (line 5). Then, the algorithm asks each such agent about 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 the agents are all sending their acceptances, 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 to itself by providing p' as the first argument (line 14). Also notice that c is incremented by 1 to specify the next

[2]We assume that the owner tags the other users in texts and mediums while specifying the post components to his agent.

Require p, the post request to be negotiated
Require $altM$, the owner's choice of alternative media
Require c, current iteration index
Require m, allowed max number of iterations
Return the final post request resulting from the negotiation
if $c > m$ **then**
  return p ;
else
  $responses \leftarrow \emptyset$;
  for all $agent \in$ DecideAgentsToNegotiateWith(p) **do**
    $response \leftarrow agent.$Ask($p$) ;
    $responses \leftarrow responses \cup \{response\}$;
  end for
  if $\forall r, r.responseCode =$ "Y", $r \in responses$ **then**
    return p ;
  else
    $p' \leftarrow$ Revise($p, altM, c, m, responses$) ;
    if $p' \neq NULL$ **then**
      return Negotiate($p', altM, c+1, m$) ;
    else
      return $NULL$;
    end if
  end if
end if

Figure 2.1. Negotiate Algorithm.

iteration number (line 14). In the other case (line 15), the algorithm returns NULL (line 16) to indicate that the negotiation has failed.

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. Essentially, the agent may be uncomfortable with following aspects of the post request.

Audience: The agent may specify the undesired people who are currently included in the audience. Many times, a user's main concern is that certain individuals will view the content and that if their not seeing is guaranteed, they would not mind the content being online.

Post Text: The agent may specify the undesired people or locations which are mentioned in the current post text.

Medium: There could be various concerns about the media in the post. First, the agent may be uncomfortable with certain people tagged in the medium or the location information of the medium. Second, the agent may specify that the medium is undesired because of its date taken. This is particularly true if the date has a certain connotation; e.g., picture taken on dates of protests are particularly uncomfortable for certain individuals. The owners having these concerns may instruct their agents to reject any medium taken on specified dates. Third, the agent may specify the undesired personal context that is exposed by medium. For example, a person might be uncomfortable with a picture because it is taken at a location or reflects a context where she does not want to be associated with. For example, a picture may reveal a person working in a company. Fourth, many times people just do not like how they look in a picture. The agent may be configured in advance such that its owner must be consulted in certain situations (e.g., certain people in the audience or a certain context of the medium). The owner may then check how he looks like and tell his agent to reject when the medium is not appealing.

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 coworkers. 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 protocol, 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 2.1, and have access to knowledge about the owner's social network and privacy constraints. We implement a semantic PRINEGO agent, which represents its knowledge as an ontology. The following sections explain our social network ontology model and how protocol functionalities 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. An ontology conceptualizes a domain [24], and it consists of three main entities: (i) *concepts (classes)* are groups of instances e.g., `Picture`, (ii) *data properties* describe attributes of a concept e.g., *hasDateTaken* is a data property of a `Picture` instance, (iii) *object properties (relationships)* relate one instance to another e.g., *includesPerson* is an object property that relates a `Picture` instance to an `Agent` instance. Moreover, OWL provides capabilities to add constraints on concepts and properties that are used as part of the ontological reasoning.

3.1.1. 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) [25].

In our ontology model, a `PostRequest` consists of a post that is intended to be

[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.

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` (`Picture` and `Video`). *hasMedium* is the 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.

3.1.2. 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 subproperties 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 3.1 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.

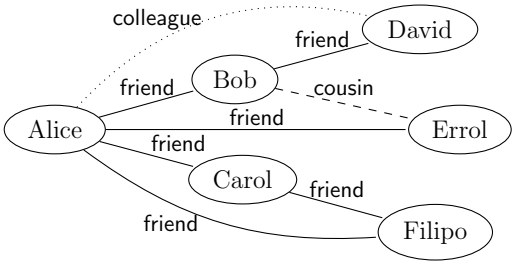

Figure 3.1. Relation properties in our social network ontology.

3.1.3. 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 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 [26]. 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.

3.1.4. Protocol Properties

Each post request is owned by an agent that sends a post request to other agents. In a negotiation process, an agent may accept or reject a post request according to its privacy rules. 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. 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. DecideAgentsToNegotiateWith Algorithm

An important step for the agent is deciding agents to negotiate with. In many negotiation problems, the parties that will negotiate are already known. However, here whom to negotiate with can be considered agent-dependent. 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 or mentioned in a post.

3.3. Evaluate Algorithm

Once an agent receives a post request, it needs to evaluate whether its user would be happy with it. Remember that a post request contains the post information, such as a possible medium, text content, target audience, and so on. Essentially, any of these points could be unacceptable for the agent that receives the request. An agent should be able to decide if the post request is acceptable and if not, it provides information about rejected attributes of the post request. This can be considered as an explanation for the negotiator agent. By responding with an explanation, the agent enables the negotiator agent to revise its post request as accurately as possible so that it is more likely to be accepted by the opposing agent in the following iteration.

Privacy Rules The privacy rules reflect the privacy preferences 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 preferences 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 [27]. Description Logic reasoners such as Pellet [28], which we use in our work, can reason on ontologies augmented with SWRL rules. Each rule consists of a *Head* and a *Body* and is of the form $Body \Longrightarrow Head$. The body part enumerates undesired properties of a post request such as the medium being in a certain personal context, e.g, party context. On the other hand, the head part specifies the rejection and rejection reasons such as target audience including a specific person, e.g., Filipo. 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 semantic agent. 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.

In Example 2.1, Bob and Carol have privacy constraints that are shown in Table 3.1. 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 3.1. 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 is 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 original 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`. i.e., `: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 employed in PriNego. First, the privacy preferences 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 5.1.1, we show how our approach works step by step on Example 2.1.

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 to find a suitable medium. The suitable medium should neither include any of the undesired people or locations, nor be taken on any of the undesired dates, nor be disliked because of neither appearance nor the context information it reveals. The approach we used to select a medium comes in handy for some cases and one such example is carried out in Section 5.1.3.

After the revision, our algorithm checks whether the resulting post request is still reasonable. A post request is unreasonable 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 at all.

4. IMPLEMENTATION

There are plenty of ways to implement the PriNego protocol. We create our own implementation of PriNego - a semantic agent that makes good use of ontologies. We also create a negotiation framework, which constitutes a general software infrastructure that allows independent agents to interact with each other. We describe our implementation details below while mentioning possible alternatives as agents can be implemented in many ways.

An agent is meant to reason on behalf of its owner based on his privacy constraints. Privacy constraints define the cases that are unacceptable for the user, for example "party pictures shall not be shown to my family members" (Bob's constraint in Example 2.1). Each agent should formally represent privacy constraints in order to operate on them. Privacy constraints can only be specified by using a set of sub-concepts; post request, the owner's relationships with connections, context and location information. These sub-concepts as well have to be formally represented, however the language to represent these sub-concepts is not necessarily same as the language to represent the privacy constraints. We make use of ontologies for knowledge representation; and use OWL to represent the sub-concepts, and SWRL to represent the privacy constraints (see Section 3.1). Using an ontology has some major advantages; OWL and SWRL together are quite expressive [27], Description Logic reasoners such as Pellet [28], which we use in our work, can reason on ontologies augmented with SWRL rules, through reasoning many inferences could be done automatically. However, different agents may differ on representing the privacy constraints and the required sub-concepts. For example, the sub-concepts and the privacy constraints can be represented by using databases, property files, RDF [29] and so on. Checking the privacy constraints in our implementation is easy with the help Pellet, a semantic reasoner. Other semantic reasoners can be used alternatively by other agents as well. For the ontologies, HermiT [30] and `FaCT++` [31] are possible candidates, for other knowledge base infrastructures CWM [32], Drools [33], Jena [34] can be examined. Some agent vendors may choose to implement their own reasoner as well, for example by making

use of SQL stored procedures if the knowledge is on a database.

Protégé [35] is a good ontology editor and also has many reasoner plugins, for example for Pellet. We use Protégé to create our ontology as in Figure 4.1. In Protégé, we first enter the concepts to work with (left on the figure), then the data and object properties associated with those concepts (data properties are on the middle, object properties on the right). We save the resulting ontology as a base ontology to use commonly in all of our semantic agents. Then, whenever we are to create a new semantic agent, we copy the former base ontology and build on top of it. For each agent, we enter their social network information into their own ontology by creating agent individuals and setting relations among them in Protégé. In addition, we enter their privacy constraints as SWRL rules by using Protégé as well (see Figure 4.2). In the body section of each rule, the undesired properties of a post request are to be specified, and the head section marks the post request as rejected, and the undesired properties as rejection reasons. From Protégé, a new post request individual can be created, and Pellet reasoner can be prompted to reason on the post request. After the reasoning, if any rules are fired, the post request is inferred as rejected by Pellet; i.e. new inferred triples indicating the rejection are added to the ontology automatically. However, our semantic agent does this automatically so no manual editing of the ontology is needed to be carried out. Whenever asked by other agents about a post request, our semantic agent implementation puts the incoming post request to its ontology and reason on it programmatically. We implement our agent using JAVA, Spring framework [36], and the OWL API [37].

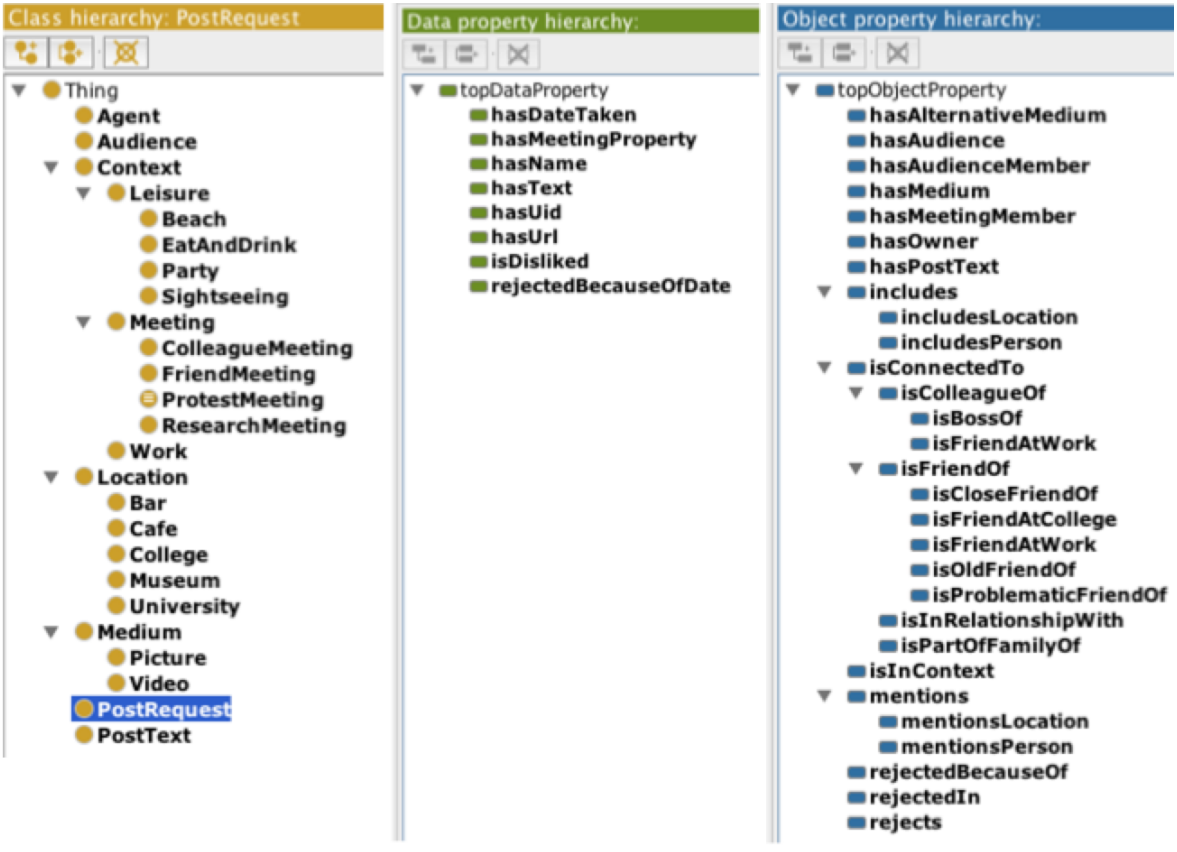

Figure 4.1. Our social network ontology model in Protégé. (i) concepts on the left, (ii) data properties on the middle, (iii) object properties on the right.

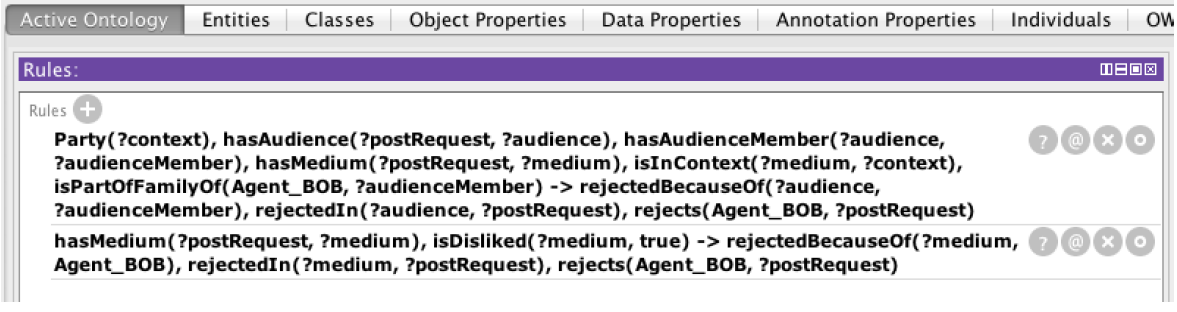

Figure 4.2. Our privacy constraints as SWRL rules in Protégé.

We create an object model to represent a post request as in Figure 4.3. We create JAVA classes for each box in the figure. As it can be seen from the figure, we represent a hierarchical context information. For now, we use a flat hierarchy for location information, however additional hierarchy levels can be introduced with ease. These hierarchies are good for expressivity. For example, a user can have a privacy constraint on his meeting photos, or more specifically protest meeting photos, alternatively. The first option covers more context types as from the figure. In addition, our system can handle media types other than photo as well such as video; picture and video are different medium types as in the figure.

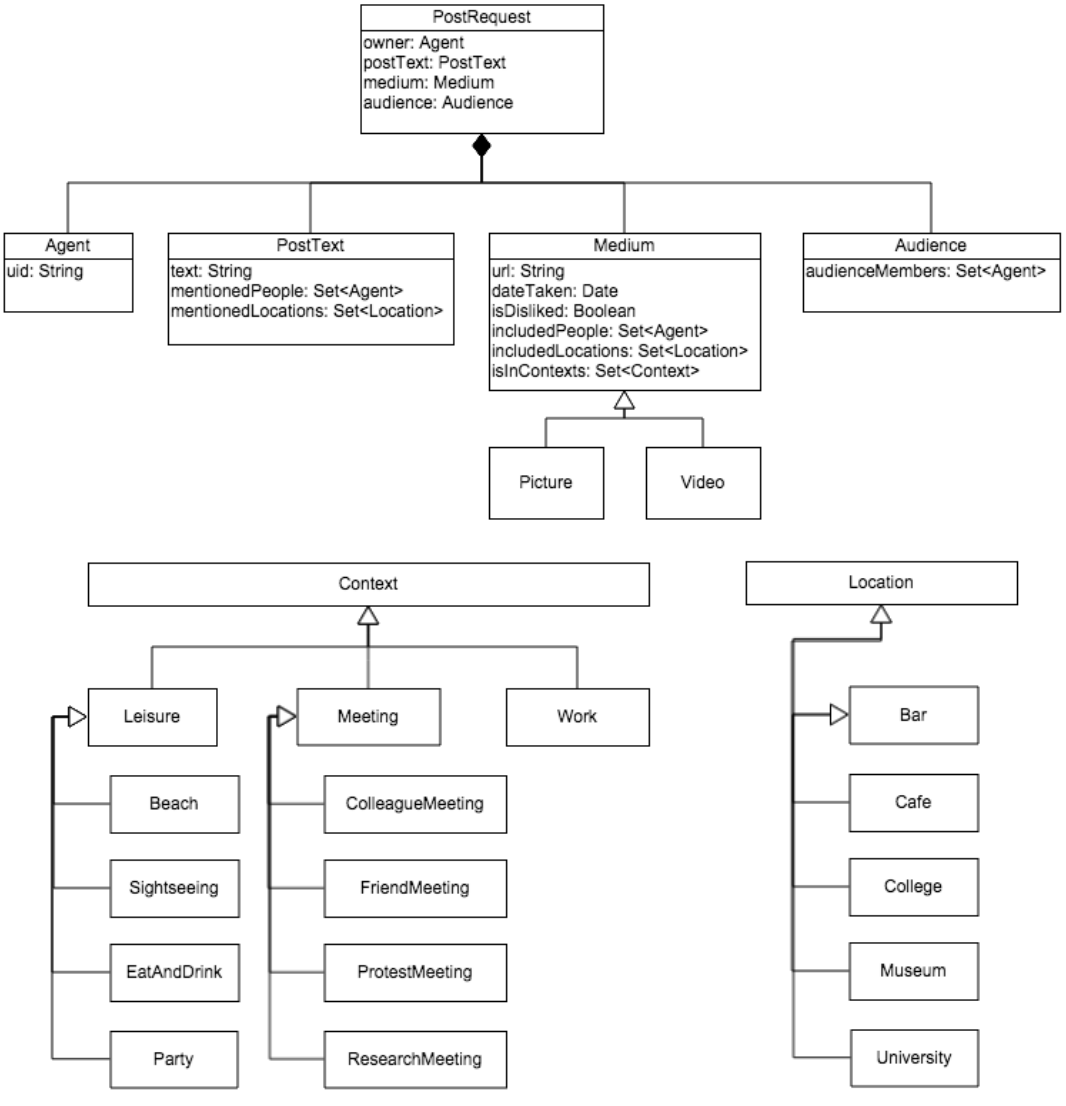

Figure 4.3. Our object model to represent a post request.

For test purposes, we use JUnit [38] - a unit testing framework for JAVA. With the help of JUnit, we are able to test each unit independently. To test a unit, we basically write a JUnit test method that isolates the unit from other units by supplying the required information flow from them in a hardcoded way. We create a post request instance in our JUnit test methods whenever needed with the same spirit. For example, to test post request evaluation unit of our semantic agent, we create an individual post request manually in the JUnit test method.

When our agent has a post request to evaluate, since SWRL rules are meant to be used only on the ontology individuals, we use OWL API and persist the post request instance in JAVA into the agent's ontology. OWL API requires the ontology file path from the very beginning. After providing the ontology, we are ready make operations on the ontology such as query or triple add/edit/delete operations. To convert a post request in JAVA into an individual in the ontology, we use class assertions, and data and object property assertions of OWL API. Then, after creating the post request individual in the ontology, we again use OWL API to trigger Pellet to reason on the individual with regard to the SWRL rules in the ontology. Then we query the post request individual in the ontology to check whether there is a rejection triple associated with it by using OWL API. If there is not such a triple, then our JAVA code returns and indicates that there is no problem with the post request being published for the agent. Otherwise, we make more queries on the ontology to understand the underlying reason on rejection, specifically we query each post request components such as post text, medium, audience to see which one is undesired and why. Our semantic agent returns a response accordingly.

We create a negotiation framework, on which we run our semantic PriNego agent and test them by observing their interactions. We embrace RESTful Web services (see [39] for more details on the technology) and the standard JSON format [40] for the agent communication. We use Restful Web services due to our familiarity to the technology, however other agent frameworks can be used as well such as JADE [41]. Each framework in general puts some boundaries for the agent implementations so that a common ground can be achieved. For example, each agent should expose one Web

service endpoint in order to negotiate with other agents through our framework. The Web service is meant to be asked by other agents about an arbitrary post request, and as such serves as an interaction point. As for the exchanged messages, the Web service accepts a JSON serialized post request, and a JSON serialized response. We create such a Web service in our semantic agent implementation. Table 4.1 shows the exchanged JSON messages (modified slightly for readability) during the interaction between our semantic agent instances for Alice and Carol.

Table 4.1. Alice's post request and Carol's response in Example 2.1 in JSON.

```
{
  "owner": "Alice"
  "medium": {
    "mediumUrl": "url_of_the_medium"
    "includedPeople": ["Bob", "Carol"]
  }
  "audience": ["Bob", "Carol",
          "David", "Errol", "Filipo"]
}
```

```
{
  "owner": "Carol"
  "responseCode": "N"
  "reason": {
    "fieldOfDiscomfort": "audience"
    "includedPeople": ["Filipo"]
  }
}
```

We use Spring in our negotiation framework, as it has many advantages such as it is quite easy to create RESTful Web services in Spring, unit tests and integrations

tests are simple with Spring's JUnit facilities [42], and deploying the Web services to an application server (Tomcat [43] in our case) is smooth. We use Maven [44] to manage our project builds.

We write our code in a modular way and highly make use of interfaces so that the code is amenable to change and new modules can be added with ease. For example, we have a domain module that represents post request and response as simple Java classes, we have another module for persisting a Java object with the ontology. Thus, if we want to use another API for ontological operations we may switch from OWL API to another ontology API, without affecting other modules. Another example is that, we model an agent skeleton as an abstract class, so that any other agent implementation extending it will be able to negotiate with the other agents we instantiate. Third, new reason types, or any other message constructs can easily be integrated with our code.

Finally, our code is validated by a mobile project that cooperates with us. The project is an Android application and integrated with a real social network, Facebook. A user logs into the 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. Then it calls our semantic agent implementation and a successful negotiation is achieved. Our code can be used by another application with quite ease as well.

5. EVALUATION

We evaluate our privacy protection approach by first testing it against realistic privacy problems arising in online social networks. Secondly, we prove the soundness of our implementation of Algorithm 2.1 in our semantic agent (see Chapter 3). Third, we compare our privacy protection approach through negotiating privacy constraints with prominent approaches from the literature in the context of online social network privacy management.

5.1. Testing on Case Scenarios

We use 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 shared 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 2.1 in Section 5.1.1 and then discuss two more scenarios in Section 5.1.3.

5.1.1. A Walk-through of PriNego

In Example 2.1, 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 DecideAgentsToNegotiateWith (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.

5.1.2. PRINEGO in Action

In this section, we show how our semantic agents negotiate with each other in Example 5.1 and Example 5.3. Table 5.1 and Table 5.2 summarize the negotiation steps. Example 5.1 is inspired from the work of Squicciarini *et al.* [45]. In this example, a user's privacy is revealed because of a friend sharing some content about this user.

Example 5.1. *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.*

Table 5.1. Iteration steps for Example 5.1 where :bob starts the negotiation.

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

In Example 5.1, :bob starts the negotiation as Bob inputs a beach picture of Alice. :bob asks to :alice and :alice rejects 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 finishes the negotiation as an agreement is reached.

Example 5.2. *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.*

Table 5.2. Iteration steps for Example 5.3 where :alice starts the negotiation.

Iter.	**Content**	**Audience**	**Asked Agents**	**Evaluate**	**Response**
1	May 1 pic.	Carol, Errol, Bob, Filipo	:carol	:carol $\rightarrow P_{C_2}$	:carol $\rightarrow$ -date
2	May 28 pic.$_1$	Carol, Errol, Bob, Filipo	:carol, :bob	:carol $\rightarrow$ N/A, :bob $\rightarrow P_{B_2}$	:carol $\rightarrow$ ✓, :bob $\rightarrow$ -self
3	May 28 pic.$_2$	Carol, Errol, Bob, Filipo	:carol, :bob	:carol $\rightarrow$ N/A, :bob $\rightarrow$ N/A	:carol $\rightarrow$ ✓, :bob $\rightarrow$ ✓

Differently in Example 5.3, 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.

5.1.3. Disagreement in PriNego

There are cases where the agents cannot reach an agreement, i.e. the negotiation ends up with a disagreement. We use the aforementioned three scenarios again to show the possible disagreements. In Example 2.1, if we change the scenario just a little bit, our semantic agents would not reach an agreement. For brevity, we present the new version of Example 2.1 here while indicating the changes we apply. Basically, strikethrough text is altered with the bold text next to it.

Example 5.3. *Alice would like to share a party picture with ~~her friends~~* ***Errol and Filipo****. 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 5.3, Alice's agent consults the agents of Bob and Carol. Bob's agent rejects the post request indicating that the audience is a problem for Bob since it includes Errol, and Carol's agent complain about Filipo being in the audience. Up to now, the flow is the same as Example 2.1. However this time, Alice's agent tries to revise the post request but is unable to find a post request that is both sane and not against the received rejection reasons. Basically, after removing both Errol and Filipo from the audience, the audience of the post request becomes an empty set, and a post request with an empty audience fails the sanity check at the end of the revision step (see Section 3.4).

Example 5.1 could result in a disagreement if Alice's privacy constraint is slightly different. P_{A_1} in Table 3.1 states that "Alice's leisure context photos shall not be seen by her coworkers". Instead of that constraint, Alice may well have this one: "Alice's leisure context photos shall not be seen by **anyone**". We can express the new privacy constraint using SWRL easily:

$P_{A_{1'}}$: `Leisure`(?context), *hasMedium*(?postRequest, ?medium), *isInContext*(?medium, ?context), *isColleagueOf*(?audienceMember, `:alice`) $\implies$ *rejects*(`:alice`, ?postRequest), *rejectedIn*(?medium, ?postRequest), *rejectedBecauseOf*(?medium, ?context)

Notice that, $P_{A_{1'}}$ does not have any audience related terms in the rule body, and the medium component is the new rejection field as it can be seen from the rule head. When we rerun Example 5.1 with Alice having the new rule instead of the old one, then Alice's agent will reject the medium when Bob's agent make the request. After that, Bob's agent is out of options to make a revision, as Bob did not provide any alternative media.

As for the Example 5.3, agents reach an agreement after 3 negotiation iterations as it can be seen from Table 5.2. In case, where Alice's agent has a maximum allowed iteration count as 2, the negotiation cannot reach an agreement. Because, our semantic agent for Alice would return the post request resulting from the iteration 2, without further negotiating it, i.e., the agent would disallow the third iteration which is critical for reaching an agreement in Example 5.3. This time, differently from the former two disagreement examples, the reason of disagreement is not due to being unable to make a sane revision, it is because of the maximum allowed iteration count.

5.2. Soundness property

As we presented both agreement and disagreement scenarios, we consider soundness of our protocol. Note that, we assume negotiations are unbounded, i.e., maximum allowed iteration counts are sufficiently large for the agents.

Theorem 5.4 (Soundness)**.** *Our protocol is sound iff privacy-rational agents conform-*

ing to the protocol can be created.

Definition 5.5 (Definition of a privacy-rational agent)**.** *An agent is privacy-rational iff the negotiations that are moderated by that agent all respect without exceptions all of the privacy constraints of the agents that are different than the owner, and either tagged in medium or mentioned in text parts of the agreed contents.*

Lemma 5.6. *Our semantic* PriNego *agent is privacy-rational.*

Proof. We assume the undesired result and reach a contradiction as our proof.

- Let p be the initial post request owned by one of our semantic agents and p' be the output post request resulting from the negotiation.
- Let u be another semantic agent of our creation that is either tagged or mentioned in p', and not the owner of p.
- Assume u has privacy constraints and p' violates one of them.
- We present how contradictory each possibility is:
  - (i) In the last iteration, the negotiator agent did not ask u, hence did not act according to the privacy constraints of u.
    - Our semantic PriNego agent would definitely ask u because our agent chooses to negotiate with everyone that is tagged or mentioned in a post (see Section 3.2), and we initially assume that u is tagged or mentioned in p'.
  - (ii) In the last iteration, the negotiator agent asked u and u returns a rejection without exposing a rejection reason. Since the reason is not seen by the negotiator agent, a privacy constraint of u is violated.
    - Our semantic PriNego agent returns a rejection reason if any (see Section 3.3). So this case is not possible. Moreover, in fact, rejection without reason results makes Algorithm 2.1 halt as the revision step cannot produce anything. Thus, it is impossible for a privacy constraint violation to occur in this case (see Section 3.4).
  - (iii) In the last iteration, the negotiator agent asked u and u returns a rejection with a rejection reason. The negotiator agent ignored the rejection reason

while revising p, hence a violation occurred.

- Our semantic PriNego agent honors each rejection reason (see Section 3.4), thus ignoring a reason is not possible.

(iv) In the last iteration, the negotiator agent asked u and u returns an acceptance while hurting its own privacy constraint.

- Our semantic PriNego agent respects its own privacy constraints in evaluation, and never returns an acceptance if there is a privacy constraint that is violated (see Section 3.3). Only owner of p can ignore its own privacy constraints, as the inputs of owner are unquestionable for his agent, privacy preferences come second. However u is not the owner as we assume at first.

(v) In the last iteration, the negotiator agent asked u and u returns an acceptance without hurting any of its privacy constraints.

- This case is trivial (If no privacy constraints are hurt, then no violation).

- We prove that our initial assumption cannot be true, i.e. p' cannot violate a privacy constraint of u. This proves that our semantic PriNego agents are privacy-rational, hence our protocol is sound.

□

5.3. State of the Art Comparison

The following three systems are important to consider regarding our purposes. Primma-Viewer [23] 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 [23]. 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 [2] 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 is a tool that converts regular pictures taken by a Google Glass into privacy-aware pictures [46]. 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 5.3. Comparison of PriNego with prominent approaches from the literature.

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

Table 5.3 compares our approach with these three systems, namely Primma-Viewer [23], Facebook [2], and FaceBlock [46] using the four criteria we defined in Chapter 1. Remember that 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. 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 doesn't 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 doesn't 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.

6. DISCUSSION

We present a novel approach for negotiating privacy constrains in the context of social networks to ensure the privacy of the users. We design and implement a privacy negotiation protocol - PriNego - which allows users to reach a consensus through their software agents prior to publishing a post. We assume that each user is helped by a software agent to manage their privacy. Contrary to typical negotiation frameworks, where all negotiating agents make offers, here only one agent proposes offers and the other agents comment on the offer by approving or giving reasons for disapproving. The agent itself collects the reasons and creates a new offer if necessary.

PriNego can be used by any agent that adheres to the following agent skeleton. The agent skeleton describes the minimal must-have functionalities for the agents and can be summarized as follows. The agent has to have access to knowledge about its owner's social network and privacy constraints. When the user wants to make a post, his agent should act as a negotiator with other agents following our negotiation protocol. 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. The negotiation protocol is composed of three (recurring) steps: deciding whom to negotiate with, asking them about the post, and revising according to their rejection reasons. Thus, the agent has to be able to perform these steps. On the other hand, the agent can also be asked by the other agents, because every agent can start a negotiation to carry out their owner's posting wishes. In that case, the agent has also be able to evaluate the post request with respect to its owner's privacy constraints. Here, we assume that the agents are able to extract personal context, included or mentioned people and locations from post contents. This way, the agent can judge whether or not the incoming post request invades any of the privacy constraints. We also create a particular PriNego agent as an example implementation of the agent skeleton. Our particular agent makes use of ontologies and semantic rules to reason on its user's privacy constraints. We have a running system and show the applicability of our approach on example scenarios from the literature.

Negotiation has been widely studied in multiagent systems, mostly from an e-commerce perspective. In e-commerce negotiation, the main idea is that all the negotiating agents concede over time from their most preferred solution to a solution that will be acceptable for all. The rational being that if not everyone is satisfied then there is no solution [22]. However, here the negotiation is asymmetric: the agent that starts the negotiation has an upper hand and cut the negotiation at any point it wants. When revising an offer, currently the agent tries to satisfy all the agents in the system; however, if the agent itself had some properties that it were not willing to let go, it could override the rejection reasons sent by some of the agents.

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 [47]. 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.* propose an extension of the Platform for Privacy Preferences (P3P) [48] to lead better expressiveness and allow the initiation of a negotiation process respectively [20]. P3P creates a standard way for Web sites to express their privacy practices regarding the gathering and harnessing of user data in a commonly agreed machine and human readable format. These privacy policies can be retrieved automatically by user agents so that the agents may assist the user in deciding when to exchange data with Web sites. Bennicke *et al.* argue that service users and service providers have different set of demands, and P3P lacks a negotiation mechanism. In this regard, they create a negotiation scheme that is based on users' predefined demands, however they do not employ semantics as we have done in our research.

Bennicke *et al.* define two complementary roles: (i) proposal maker and (ii)

accepter. There should be at least two parties for a negotiation to take place; one in proposal maker role (makes a proposal), and the other in accepter role (decides on the acceptance of the proposal). The parties can also switch roles during the negotiation, because the accepter can choose to make a proposal itself along with the response. Bennicke *et al.* formulate five different types of replies for the accepter to choose from. The accepter can accept without further restrictions (accept) or postpone the request and make the acceptance decision dependent on the proposal maker's behaviour towards different subjects (conditional accept). The accepter can also reject in three ways: (i) without any explanation (reject), (ii) delivers a pointer to the parts that are not acceptable (descriptive reject), (iii) makes a counter-proposal (proposing reject). As the authors point out, among these response types "conditional accept" is the one that require the most care. Because both parties can consecutively choose to respond each other with conditional accepts a number of times. This may cause to deadlock, which is to be avoided with additional handles. In PriNego, we employ neither "conditional accept" nor "proposing reject" for the sake of simplicity. The asked agents can either accept, or reject with or without explanations. Moreover, any agent can start over with another negotiation anytime; however accommodating these message types would be straightforward extensions.

Similarly, Walker *et al.* propose a privacy negotiation protocol to increase the flexibility of P3P through privacy negotiations. [49]. The client can be prompted to provide data about himself by the server (e.g., a telephone number). The server can have different intentions (to obtain revenue) such as (i) make the data accessible to some other parties, (ii) store the data for a long time, (iii) use the data for different purposes. The proposed protocol allows privacy negotiation between the client and the server, and is based on an "Or Best Offer" (OBO) negotiation style, similar to sellers expressing a willingness to entertain a "best offer". The negotiations are guaranteed to be terminated within a maximum of three rounds by the protocol. Particularly, the negotiation succeeds if one of the parties accepts the proposal made in any round, and fails only in the last (third) round if the client rejects the final proposal. The first round begins after the client is prompted to either accept or reject the server's default policy. The client can accept or make a counter-proposal along with some hints about

how the server can best satisfy her needs. Then, the server can accept or create a best offer to conform to the client's preferences, while at the same time meeting its own needs. The client can either accept this best (final) offer, or reject it. The authors describe a set of rules for the negotiation strategy to produce Pareto-optimal policies, thus 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 as we have done here.

In the context of social networks, there are different approaches to ensure privacy of the users. However, to the best of our knowledge, none of them negotiate the post contents (text and medium) and the target audience according to the users' privacy constraints prior to publishing the post in order to converge to an agreement that is slightly different than the original post but respects the users' privacy constraints. The closest research to ours is FaceBlock by Pappachan *et al.* [46]. FaceBlock is a tool for Google Glass users that enables the exchange of privacy policies and blurs some of the faces in the images a result. Their solution is automated and contains context-based policies similar to our work. However, we provide a more fair solution than FaceBlock and also conceal the privacy preferences itself. (Please see Section 5.3 for a detailed comparison).

Besides FaceBlock, collaborative access control models to manage privacy in the social networks are also closer to our research. One such study is a collaborative privacy management tool that Squicciarini *et al.* propose [45]. The tool helps users to protect their private information shared among social network users. Authors categorize the users into three groups: content-owners (those that create), co-owners (those that are tagged) and content-viewers (those that view). They advocate that co-owners should also manage the privacy of the content and propose a collaborative environment to enable this. We agree with the authors on empowering co-owners and thus in PriNego the negotiation establishes this. Further, since our approach is agent-based, we can establish this with little help from the users.

Similarly, Carminati and Ferrari propose a collaborative access control model for

social networks [50], where the users collaborate during the access request evaluation and the access rule administration. On the access request evaluation, their focus is on resource confidentiality however our focus in on user privacy. For access rule administration, the resource owner receives feedbacks from the collaborative users. The feedback is limited to acceptance or rejection of rules; however, in our case, each agent can report a reason on rejection and the content owner can update the privacy settings accordingly. Further, we currently have a running system that can automate the negotiation on behalf of the users; rather than the users providing the feedback on their own.

Another approach is to detect the privacy violations that would occur in social networks, and proposed by Kökciyan and Yolum as PriGuard [14] . PriGuard transforms privacy requirements into commitments between the social network and the users. In a particular state of the online social network, PriGuard checks for privacy violations and notifies the user if any. While they use commitments to represent a user's privacy preferences, we make use of SWRL rules for this. They provide a user interface where users specify their privacy requirements. Similar to them, we are currently developing such a user interface for the specification of the privacy constraints. They use a simple ontology of the social network domain to represent the commitments semantically. In our ontology, we cover the context information and the negotiation protocol as well. Our negotiation framework is in place before the content is put up contrary to PriGuard, which detects privacy violations after a content is posted.

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. We may work on utilising a learning algorithm to estimate other agents' behavior, however this in principle is against to one of our initial purposes, concealment of privacy preferences. Our protocol makes the agent know less about the others by design. We will try to find out the sweet spot between these two ends: faster negotiations or more privacy. Second, we want to improve our

revise algorithm to generate better offers. For instance, our semantic agent clears controversial post texts altogether to move on with the medium content, or tries an arbitrary medium from the alternative media set of the user if the current medium is rejected by some agent. There are endless other possibilities to reach an offer. To name a few, we can keep the post text but change the medium, and utilize a priority for the mediums in the alternative set to name a few. We will inspect these possibilities and try to create a "revision guide" that includes best practices. Third, we want to apply prioritization techniques in our protocol and validate its efficiency. Consider a case where the negotiator agent asks Alice and David about a post request, and they both do not want themselves in the picture. Currenty, as a result of the negotiation our semantic agent removes both of them from the picture. However, as possible approach would be making a prioritization between them and negotiate one by one. With such additional handles, we want to aim greater fairness for our protocol. Last but not least, we can improve our mobile Facebook app and distribute it to a selection of real users for a survey. The survey may point other interesting directions.

REFERENCES

1. "Social Networking Fact Sheet", Website, 2014, http://www.pewinternet.org/fact-sheets/social-networking-fact-sheet/, [Accessed January 2015].

2. "Facebook", Website, 2004, http://www.facebook.com, [Accessed November 2014].

3. Westin, A. F., *Privacy and Freedom*, Athenaeum, New York, NY, USA, 1967.

4. Jensen, C. and C. Potts, "Privacy Policies as Decision-making Tools: An Evaluation of Online Privacy Notices", *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 471–478, ACM, 2004.

5. McDonald, A. M. and L. F. Cranor, "The Cost of Reading Privacy Policies", *ISJLP*, Vol. 4, pp. 543–564, 2008.

6. Jensen, C., C. Potts and C. Jensen, "Privacy Practices of Internet Users: Self-reports Versus Observed Behavior", *International Journal of Human-Computer Studies*, Vol. 63, No. 1, pp. 203–227, 2005.

7. Acquisti, A. and J. Grossklags, "Privacy and Rationality in Individual Decision Making", *IEEE Security & Privacy*, Vol. 2, pp. 24–30, 2005.

8. "The Organization for Economic Co-operation and Development", Website, 1961, http://www.oecd.org/, [Accessed January 2015].

9. OECD, "Recommendation of the Council Concerning Guidelines Governing the Protection of Privacy and Transborder Flows of Personal Data", Website, 2013, http://www.oecd.org/sti/ieconomy/2013-oecd-privacy-guidelines.pdf, [Accessed January 2015].

10. "Federal Trade Commission", Website, 1914, http://www.ftc.gov/, [Accessed January 2015].

11. Hans, G. S., *Privacy Policies, Terms of Service, and FTC Enforcement: Broadening Unfairness Regulation for a New Era*, 2012, 19 Michigan Telecommunications and Technology Law Review 163-200.

12. "Lane v. Facebook, inc. Wikipedia entry", Website, 2014, http://en.wikipedia.org/wiki/Lane_v._Facebook,_Inc., [Accessed January 2015].

13. Carr, A., "Meet the Big Brother Screening Your Social Media for Employers", Website, 2010, http://www.fastcompany.com/1692172/meet-big-brother-screening-your-social-media-employers, [Accessed January 2015].

14. Kökciyan, N. and P. Yolum, "Commitment-Based Privacy Management in Online Social Networks", *First International Workshop on Multiagent Foundations of Social Computing*, 2014.

15. Baden, R., A. Bender, N. Spring, B. Bhattacharjee and D. Starin, "Persona: An Online Social Network with User-Defined Privacy", *Computer Communication Review (SIGCOMM)*, Vol. 39, No. 4, pp. 135–146, 2009.

16. Debatin, B., J. P. Lovejoy, A.-K. Horn and B. N. Hughes, "Facebook and Online Privacy: Attitudes, Behaviors, and Unintended Consequences", *Journal of Computer-Mediated Communication*, Vol. 15, No. 1, pp. 83–108, 2009.

17. Andrews, L., *I Know Who You are and I Saw What You Did: Social Networks and the Death of Privacy*, Simon and Schuster, 2012.

18. Kafalı, O., A. Günay and P. Yolum, "Detecting and Predicting Privacy Violations in Online Social Networks", *Distributed and Parallel Databases*, Vol. 32, No. 1, pp. 161–190, 2014.

19. Clarke, R., "Platform for Privacy Preferences: A Critique", *Privacy Law & Policy Reporter*, Vol. 5, No. 3, pp. 46–48, 1998.

20. Bennicke, M. and P. Langendorfer, "Towards Automatic Negotiation of Privacy Contracts for Internet Services", *The 11th IEEE International Conference on Networks (ICON)*, pp. 319–324, 2003.

21. Stewart, M. G., "How Giant Websites Design for You (and a Billion Others, too)", TED Talk, 2014.

22. Jennings, N. R., 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*, Vol. 10, No. 2, pp. 199–215, 2001.

23. Wishart, R., D. Corapi, S. Marinovic and M. Sloman, "Collaborative Privacy Policy Authoring in a Social Networking Context", *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, pp. 1–8, Washington, DC, USA, 2010.

24. Gruber, T. R., "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, Vol. 5, pp. 199–220, 1993.

25. McGuinness, D. L., F. Van Harmelen *et al.*, "OWL Web Ontology Language Overview", *W3C recommendation*, Vol. 10, No. 2004-03, p. 10, 2004.

26. Schmidt, A., M. Beigl and H.-W. Gellersen, "There is More to Context than Location", *Computers & Graphics*, Vol. 23, No. 6, pp. 893–901, 1999.

27. Horrocks, I., 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*, Vol. 21, p. 79, 2004.

28. Sirin, E., 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*, Vol. 5, No. 2, pp. 51–53, 2007.

29. "Resource Description Framework (RDF)", Website, 2014, `http://www.w3.org/RDF/`, [Accessed January 2015].

30. Shearer, R., B. Motik and I. Horrocks, "HermiT: A Highly-Efficient OWL Reasoner.", *OWLED*, Vol. 432, 2008.

31. Tsarkov, D. and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description", *Automated reasoning*, pp. 292–297, Springer, 2006.

32. Berners-Lee, T. *et al.*, "Cwm: A General Purpose Data Processor for the Semantic Web", Website, 2000, `http://www.w3.org/2000/10/swap/doc/cwm.html`, [Accessed January 2015].

33. "Drools", Website, 2006, `http://www.drools.org/`, [Accessed January 2015].

34. "Apache Jena", Website, 2011, `http://jena.apache.org/`, [Accessed January 2015].

35. "Protégé", Website, 2014, `http://protege.stanford.edu/`, [Accessed January 2015].

36. "Spring Framework", Website, 2015, `http://projects.spring.io/spring-framework/`, [Accessed January 2015].

37. Horridge, M. and S. Bechhofer, "The OWL API: A Java API for OWL Ontologies", *Semantic Web*, Vol. 2, No. 1, pp. 11–21, 2011.

38. "JUnit", Website, 2002, `http://junit.org/`, [Accessed January 2015].

39. Richardson, L. and S. Ruby, *RESTful Web Services*, " O'Reilly Media, Inc.", 2008.

40. "Introducing JSON", Website, 2013, `http://www.json.org/`, [Accessed January 2015].

41. Bellifemine, F., A. Poggi and G. Rimassa, "JADE–A FIPA-Compliant Agent Framework", *Proceedings of PAAM*, Vol. 99, p. 33, London, 1999.

42. "Spring's JUnit Capabilities", Website, 2009, `http://docs.spring.io/spring-batch/trunk/reference/html/testing.html`, [Accessed January 2015].

43. "Apache Tomcat", Website, 1999, `http://tomcat.apache.org/`, [Accessed January 2015].

44. "Apache Maven Project", Website, 2002, `http://maven.apache.org/`, [Accessed January 2015].

45. Squicciarini, A. C., H. Xu and X. L. Zhang, "CoPE: Enabling Collaborative Privacy Management in Online Social Networks", *J. Am. Soc. Inf. Sci. Technol.*, Vol. 62, No. 3, pp. 521–534, Mar. 2011.

46. Pappachan, P., R. Yus, P. K. Das, T. Finin, E. Mena and A. Joshi, "A Semantic Context-Aware Privacy Model for FaceBlock", *Second International Workshop on Society, Privacy and the Semantic Web - Policy and Technology (PrivOn), Riva del Garda (Italy)*, October 2014.

47. Aydogan, R. and P. Yolum, "Learning Opponent's Preferences for Effective Negotiation: An Approach Based on Concept Learning", *Autonomous Agents and Multi-Agent Systems*, Vol. 24, No. 1, pp. 104–140, 2012.

48. W3C, "Platform for Privacy Preferences (P3P) Project", Website, 2007, `http://www.w3.org/P3P/`, [Accessed January 2015].

49. Walker, D. D., E. G. Mercer and K. E. Seamons, "Or Best Offer: A Privacy Policy Negotiation Protocol", *IEEE Workshop on Policies for Distributed Systems and*

Networks (POLICY), pp. 173–180, 2008.

50. Carminati, B. and E. Ferrari, "Collaborative Access Control in On-line Social Networks", *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pp. 231–240, Oct 2011.