Modeling Social Networking Privacy

Carolina Dania*†, Manuel Clavel*
*IMDEA Software Institute
†Universidad Complutense de Madrid
carolina.dania@imdea.org, manuel.clavel@imdea.org

***Abstract*—In this paper we propose to use a formal language, called SecureUML, to model social networking privacy. SecureUML is a language for specifying role-based static and dynamic access control policies, the latter being policies that depend on the run-time satisfaction of specific constraints (e.g., the privacy settings in social networking sites). By using a formal language for modeling social networking privacy, we provide a rigorous, unambiguous description of the policies, and a solid, much-needed formal foundations for tools to validate them and to perform change impact analysis. To illustrate our proposal, as well as its benefits, we use Facebook as a case study; in particular, the latest two versions of Facebook's policy for posting and tagging.**

I. Introduction

Nowdays many people consider themselves as "Internet natives" (and many others are happy "Internet immigrants"): when they need information, they naturally open a browser and search for it; when they want to share information, they naturally post it on a social network. At the same time, privacy-related issues are a growing concern among users of social networking sites [10], [1], [21], [22], [13], [20], and among their regulators. On December 21, 2011, the Office of the Irish Data Protection Commissioner (DPC) announced the results of its "thorough and detailed audit of Facebook's practices and policies" [23], which includes, among many others, the following recommendations and findings [11] (see [12] for the latest DPC's follow-up review):

> Facebook must work towards:(i) simpler explanations of its privacy policies; (ii) easier accessibility and prominence of these policies during registration and subsequently; (iii) an enhanced ability for users to make their own informed choices based on the available information.
>
> Many policies and procedures that are in operation are not formally documented. This should be remedy.
>
> We recommend that Facebook introduce increased functionality to allow a poster to be informed prior to posting how broad an audience will be to view their post and that they be notified should the setting on that profile be subsequently change to make a post that was initially restricted available to a broader audience.

To Facebook's credit, over the past 4 years, users have been equipped with new tools and resources which are designed to give them more control over their so-called Facebook experience, including: an easier way to select your audience when making a new post; inline privacy control on all your existing posts; the ability to review tags made by others before they appear on your profile; a tool to view your profile as someone else would see it; and more privacy education resources. Despite all these efforts, many users are still concerned about how to maintain their privacy or —in Mark Zuckerberg's own words— "rightfully questions how their information was protected" [24]. In our opinion, there are at least three reasons for this:

- Facebook's privacy policy is hardly trivial to understand. For example, when default policies and privacy settings for posting and tagging conflict to each other (which happens very often) the solution will depend (sometimes in a convoluted way) on the existing relationships among all the users involved: the owner of the timeline, the creator of the post, the creators of the tags, and the reader of the post.
- Facebook's privacy policy has been in a constant state of flux over the past few years [17], and it is prompted to change again in the future.
- Facebook's privacy policy is only informally and partially described in a collection of privacy education resources and blogs, which cannot provide a coherent and complete account of this policy.

As a consequence, even advanced Facebook users may find difficult to understand the actual visibility of a post. To illustrate our point, we recall first the answers given in Facebook's 2013 Frequently Asked Questions (FAQ) [7] regarding the policy for posting and tagging:

- *If I post something on my friend's timeline, who gets to see it?*
  When you post something on a friend's timeline, who else gets to see it will depend on the privacy settings your friend has selected. If you want to write something to your friend privately, don't post it.
- *What does the 'Only Me' privacy setting mean?*
  Sometimes you might want certain posts visible only to you. Post with the 'Only Me' audience will appear on your timeline and in your news feed but won't be

visible to anyone else. If you tag someone in an 'Only Me' post, they will be able to see the post.

- *When I share something, how do I choose can see it?* Before you post, look at the audience selector. Use the dropdown menu to choose who want to share a post with.
  - Public
  - Friends (+ friends of anyone tagged)
  - Only Me
  - Custom (Includes specific groups, friends lists or people you've specified to include or exclude)

  Remember: anyone you tag in a post, along their friends may see the post. (...)
  Note: When you post to another person's timeline, that person controls what audience can view your post.
- *What is tagging and how does it work?* A tag is a special kind of link. When you tag someone, you create a link to their timeline. The post you tag the person in is also added to the person's timeline. For example, you can tag a photo to show who's in the photo or post status update and say who you're with. (..) When you tag someone, they'll be notified. Also, if you or a friend tags someone in your post and it's set to 'Friends' or more, the post could be visible to the audience you selected plus friends of the tagged person.
  When someone adds a tag of you to a post, your friends may be able to see this. The tagged post also goes on your timeline.

Now, suppose that Bob, Alice, Ted, and Peter have Facebook profiles: Bob is a friend of Alice and Ted; Ted is a friend of Peter; Ted is not a friend of Alice; Peter is not a friend of Alice or Bob; and none of them has blocked to another. To appreciate the challenge of understanding the actual visibility of a post, consider the scenarios S1–S4 below and try to justify (based on the previously recalled Facebook's policy) our answers to the given questions.[1]

S1 Alice posts a photo of herself, Bob and Ted in her timeline, and sets its audience to 'Friends'. Then, Alice tags Bob in this photo. *Question: Can Bob see the photo in Alice's timeline?* The answer is Yes.

S2 Alice has set to 'Only Me' the default audience for posts made by her friends in her timeline. Bob posts a photo in Alice's timeline. *Question: Can Bob see this photo in Alice's timeline?* The answer is Yes.

S3 Alice posts a photo of herself, Bob and Ted in her timeline, and set its audience to 'Friends'. Then, Bob tags Ted in this photo. *Question: Can Peter see this photo in Alice's timeline?* The answer is Yes.

S4 Bob posts a photo of himself, Ted and Alice in Alice's timeline. Alice has setting by default 'Only Me'. Then, Bob tags Ted in this photo. *Question: Can Peter see this photo in Alice's timeline?* The answer is No.

[1]These answers were obtained in 2013 on real Facebook scenarios.

Clearly, as was explicitly requested in the DPC audit, Facebook should provide simpler explanations of its privacy policies. Even better, it should formally document these policies. But, *which formal language is up to the task of modeling social networking privacy?* In this paper, we propose to use SecureUML [2] for this task. In a nutshell, SecureUML is a formal language for modeling role-based static and dynamic access control policies, the latter being policies that depend on the run-time satisfaction of specific constraints.

Organization.

In Section II we provide background information about SecureUML. Next, in Sections III and IV, we introduce, respectively, our data and SecureUML models for posting and tagging, as these actions are described in Facebook's 2013 FAQ. Afterwards, in Section V, we discuss how these models shall be changed to reflect the changes introduced in 2014 in Facebook's policy for posting and tagging. Then, in Section VI, we briefly discuss how to formally reason about SecureUML models, using as an example our model for Facebook posting and tagging. Finally, in Sections VII and VIII we discuss related work and draw conclusions.

II. SecureUML

SecureUML [2] is a modeling language for formalizing access control requirements that is based on RBAC [9]. In RBAC, permissions specify which roles are allowed to perform given operations. These roles typically represent job functions within an organization. Users are granted permissions by being assigned to the appropriate roles, based on their competencies and responsibilities in the organization. RBAC additionally allows one to organize the roles in a hierarchy, where roles can inherit permissions along the hierarchy. In this way, the security policy can be described in terms of the hierarchical structure of an organization.

It is not possible, however, to specify in RBAC policies that depend on dynamic properties of the system state. For example, to allow a user in a role to execute a method only if she satisfies specific constraints with respect to the actual resource on which the method is executed and/or the actual arguments that are passed to the method. To overcome this limitation, SecureUML extends RBAC with *authorization constraints*, which are formalized in SecureUML using the Object Constraint Language (OCL) [16].

OCL is a textual language for specifying constraints and queries. As part of the UML standard, it was originally intended for modeling properties that could not be easily expressed using graphical notation, such as class invariants in a UML class diagram. Every OCL expression is written in the context of a model (called the *contextual model*), and is evaluated on an object model (also called the *instance* or

scenario) of the contextual model. This evaluation returns a value but does not alter the given object model, since OCL's evaluation is side-effect free.

OCL is a strongly typed language. Expressions either have a primitive type, a class type, a tuple type, or a collection type. OCL provides standard operators on primitive data, tuples, and collections. For example, the operator $->$includes checks whether an object is part of a collection. OCL also provides a dot-operator to access the values of the objects' properties (e.g., the attributes and association-ends in UML class diagrams) in the given scenario. For example, suppose that the contextual model includes a class c with an attribute at and an association-end as. Then, if o is an object of the class c in the given scenario, the expression $o.at$ refers to the value of the attribute at for the object o in this scenario, and $o.as$ refers to the objects linked to the object o through the association-end as. Finally, OCL provides operators to iterate over collections. For example, $->$forAll, $->$exists, $->$select, $->$reject, $->$collect, and $->$iterate. In the following section we will provide examples of OCL expressions whose context is precisely our data model for Facebook.

III. Modeling Facebook's Data Structure

Facebook is a social network that "helps you connect and share with the people in your life." Each user has a *profile* that, basically, contains his/her personal information (name, surname, email, birthday, gender, and relationship status) and preferences (about music, television, movies, and games). Moreover, each user's *timeline* can displays posts or stories, status updates, tags on status updates, comments to posts or stories, photos, comments to photos, and tags on photos. In this section we introduce our data model for Facebook's profiles, timelines, posts, and tags. We do not intend to model these features in full, but rather those aspects that will play a role when modeling Facebook's policy for posting and tagging.

In Figure 1 we show how we can model, using a UML class diagram, profiles, timelines, posts, and tags. The following explanations highlight our main modeling decisions.

- Each profile, timeline, post, and tag, is modeled, respectively, as an instance of the classes `Profile`, `Timeline`, `Post`, and `Tag`.
- The method `addPost(@post)` adds the post `@post` to the given timeline.
- The method `removePost(@post)` removes the post `@post` from the given timeline.
- The method `readPost(@post)` read the post `@post` in the given timeline.
- Each photo is a type of post.
- Each profile is linked to exactly one timeline via `timeline`. This is the profile's timeline.
- Each profile is linked to those who are friends of him or her via `friends`.
- Each profile is linked to those that he or she has blocked via `blocks`.
- Each profile has two attributes, namely, `tagReview` and `contributors`. The attribute `tagReview` holds the setting chosen by the profile's owner for Tag Review. The attribute `contributors` holds the setting chosen by the profile's owner for posting on its timeline.
- The method `switchTagReview()` switches on/off the Tag Review on the given profile.
- The method `setContributors(@selection)` set to `@selection` the intended post contributors to a given profile's timeline.
- Each post is linked to exactly one timeline via `posted`. This is the timeline on which the post is posted.
- Each post has two attributes, namely, `creator` and `audience`. These attributes hold, respectively, the post's creator and the post's selected audience.
- The method `setAudience(@selection)` set to `@selection` the intended audience for a given post.
- The method `addTag(@profiling)` adds a tag of a profile `@profiling` to a given post.
- The method `removeTag(@tag)` removes a tag `@tag` from a given post.
- The method `forbidTag(@profiling)` adds a profile `@profiling` to the list of profiles that can not be tagged on a given post.
- Each tag is linked to exactly one post via `post`. This is the post on which the tag appears.
- Each tag is linked to exactly one profile via `profiling`. This is the tag's target.
- Each tag has one attribute, namely, `creator`, that holds the tag's creator.
- Each post is linked to those profiles that can not be tagged in the post via `forbidens`.

We show in Figure 2 an instance of our data model for Facebook. It represents the following Facebook scenario: Bob, Alice and Ted have Facebook profiles. Bob is a friend of Alice, and Ted is a friend of Bob but not a friend of Alice. Alice's timeline has a photo that was posted by Bob in her timeline. Bob has tagged Ted in this photo.

We conclude this section showing how we can formalize *queries* about the Facebook scenario represented in Figure 2 using OCL. In particular,

- To query about Bob's friends, we can use the OCL expression:

  Bob.friends.

  This expression evaluates to `Set{Alice, Ted}` in our sample scenario.
- To query about friends of Alice's friends, we can use the OCL expression:

  Alice.friends.friends$->$asSet().

  This expression evaluates to `Set{Ted, Alice}` in

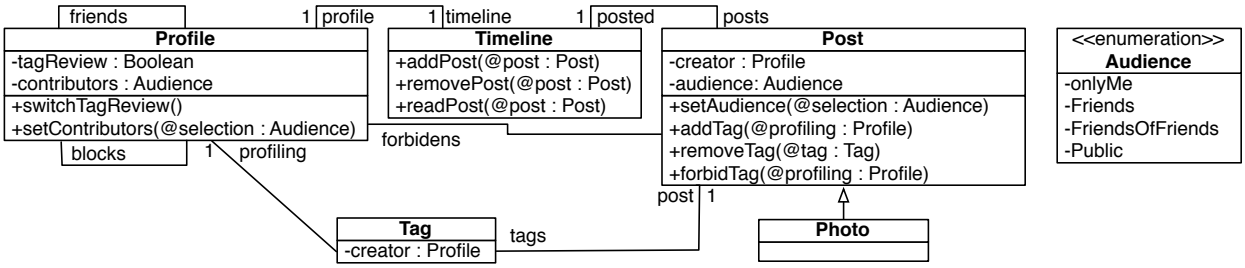

Figure 1. Modeling Facebook's data structure (partial).

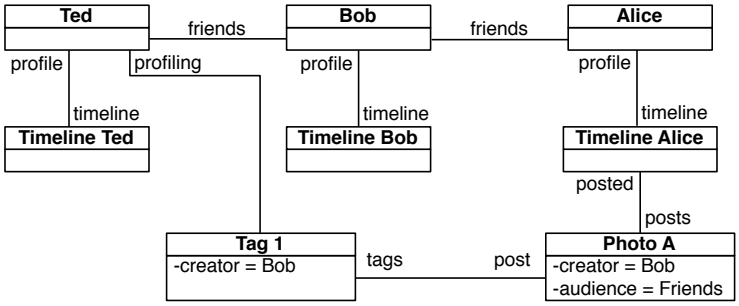

Figure 2. Modeling a Facebook scenario.

our sample scenario (Alice is certainly a friend of any of her friends).

- To query about friends of Alice and their friends, but not including Alice herself, we can use the OCL expression:

  Alice.friends−>union(Alice.friends.friends)
  −>excluding(Alice).

  This expression evaluates to `Set{Bob, Ted}` in our sample scenario.
- To query about whether Ted is tagged in any of the posts appearing on Alice's timeline, we can use the OCL expression:

  Alice.timeline.posts.tags.profiling−>includes(Ted).

  This expression evaluates to `true` in our sample scenario.

IV. Modeling Facebook's Privacy Policy

In this section we model, using SecureUML, the Facebook's 2013 policy for posting and tagging.[2] A word of caution: given the lack of a formal documentation, our understanding of this policy is based not only on the official information available at [7] (which is not always complete or coherent) but also on our own experiments using Facebook on "precooked" scenarios.

In what follows, for each method in our data model for Facebook, after describing the policy for executing this method, we will formally specified, using OCL, the corresponding authorization constraint. Please, be aware of the following SecureUML's conventions regarding variables in authorization constraints:

- The variable `@caller` refers to the user asking permission for executing a method. In our model, this variable has type `Profile` since we assume that the users participate in the social network through their profiles.
- The variable `@self` refers to the object on which the method will be executed, if permission is granted.
- The method's parameters can appear as variables in the method's authorization constraint.

Method: switchTagReview()

The following clause describes the policy for executing the method `switchTagReview`:

- anybody can turn on/off the option of reviewing any tag that anybody else wants to add to any post published in his or her timeline before they are actually published.

More formally, the permission to execute the method `switchTagReview()` has the following authorization constraint: @caller=@self.

[2]Understandably, for the sake of space limitation, we have to omit some interesting features, including: who is notified (and how) when a tag is added to a post; how (and by whom) a post's audience can be customized; how (and by whom) a post on which someone is tagged can be reviewed before it appears on his or her profile; how (and by whom) the maximum audience for posts appearing in someone's profile because he or she is tagged on them can be selected by default; how (and by whom) a tag can be added to a post different from a photo; how (and by whom) something different from a user can be tagged.

Method: setContributors(@audience)

The following clause describes the policy for executing the method `setContributors`:

- anybody can choose between not allowing anybody (except him or herself) to post on his or her timeline or allowing also their friends to post on his or her timeline.

More formally, the permission to execute the method `setContributors(@audience)` has the following authorization constraint: @caller=@self.

Method: setAudience(@audience)

The following clause describes the policy for executing the method `setAudience`:

- anybody can select the audience for any post that is posted on his or her timeline.

More formally, the permission to execute the method `setAudience(@audience)` has the following authorization constraint: @caller=@self.posted.profile.

Method: addPost(@post)

The following clauses describe the policy for executing the method `addPost`:

- anybody can add a post on his or her timeline.
- anybody can add a post on any of his or her friends' timelines, if the owner of this timeline has its preferences for posting set to 'Friends'.

More formally, the permission to execute the method `addPost(@post)` has the following authorization constraint:

@caller=@self.profile or
(@self.profile.contributors='Friends'
and @self.profile.friends−>includes(@caller)).

Method: removePost(@post)

The following clause describes the policy for executing the method `removePost`:

- anybody can remove a post that he or she has posted on a timeline.

More formally, the permission to execute the method `removePost(@post)` has the following authorization constraint: @caller=@post.creator.

Method: addTag(@profiling)

The following clauses describe the policy for executing the method `addTag`:

- anybody can add a tag of him or herself, or of any of his or her friends, on a post that is posted on his or her timeline, unless this friend has previously untagged him or herself from this post.
- anybody can add a tag of him or herself, or of any of his or her friends, on a post that is posted on a timeline, unless the owner of the timeline has switched 'On' the tag review preferences and he or she is not the owner of the timeline, or unless this friend has previously untagged him or herself from this post.

More formally, the permission to execute the method `addTag(@profiling)` has the following authorization constraint:

((@caller=@profiling or @caller.friends−>includes(@profiling))
and @caller=@self.posted.profile
and @self.forbidens−>excludes(@profiling))
or ((@caller=@profiling
or @caller.friends−>includes(@profiling))
and @self.posted.profile.tagReview=false
and @self.forbidens−>excludes(@profiling)).

Method: removeTag(@tag)

The following clauses describe the policy for executing the method `removeTag`:

- anybody can remove any tag of him or her on a post.
- anybody can remove any tag from a post that he or she has posted on a timeline.
- anybody can remove any tag that he or she has added to a post.

More formally, the permission to execute the method `removeTag(@tag)` has the following authorization constraint:

@caller=@tag.profiling or @caller=@tag.post.creator
or @caller=@tag.creator.

Method: forbidTag(@profiling)

The following clause describes the policy for executing the method `forbidTag`:

- anybody can forbid anybody else to tag him or her again on a post.

More formally, the permission to execute the method `forbidTag(@profiling)` has the following authorization constraint: @caller=@profiling.

Method: readPost(@post)

The following clauses describe the policy for executing the method `readPost`:

- anybody can read any post that is posted on his or her timeline.
- anybody can read any post that was posted by him or her on a timeline, unless he or she is blocked by the owner of the timeline.
- anybody can read any post that has its audience selected to 'Friends', if he or she is a friend of the owner of the timeline.
- anybody can read any post that has its audience selected to 'FriendsOfFriends', if he or she is a friend of the owner of the timeline, or a friend of a friend of the owner of the timeline, unless he or she is blocked by the owner of the timeline.

- anybody can read any post that has its audience selected to 'Public', unless he or she is blocked by the owner of the timeline.
- anybody can read any post, if he or she is tagged on this post, unless he or she is blocked by the owner of the timeline.
- anybody can read any post that has its audience selected to 'Friends' and was created by the owner of the timeline, if he or she is a friend of somebody tagged on the post, unless he or she is blocked by the owner of the timeline.

More formally, the permission to execute the method `readPost(@post)` has the following authorization constraint:

```
@caller=@self.profile
or (@caller=@post.creator
 and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Friends'
 and @self.profile.friends->includes(@caller))
or (@post.audience = 'FriendsOfFriends'
 and (@self.profile.friends->includes(@caller)
  or @self.profile.friends.friends->includes(@caller))
 and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Public'
 and @self.profile.blocks->excludes(@caller))
or (@post.tags.profiling->includes(@caller)
 and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Friends'
 and @post.creator=@self.profile
 and @post.tags.profiling.friends->includes(@caller)
 and @self.profile.blocks->excludes(@caller)).
```

To end this section, we can validate our modeling of the policy for executing the method `readPost`, using the scenarios S1–S4 that we introduced in Section I. Clearly, if our model is correct, the answers obtained in our real experiments about the visibility of the posts in these scenarios should correspond to the results of evaluating the method `readPost`'s authorization constraint on the corresponding instances of our data model for Facebook.

Recall that Bob is a friend of Alice and Ted; Ted is a friend of Peter; Ted is not a friend of Alice; Peter is not a friend of Alice or Bob; and none of them has blocked to another.

S1 Alice posts a photo of herself, Bob and Ted in her timeline, and sets its audience to 'Friends'. Then, Alice tags Bob in this photo. *Question: Can Bob see the photo in Alice's timeline?* The answer is Yes, because Alice has set her default audience to 'Friends' and Bob is a friend of Alice. Indeed, the `readPost`'s authorization constraint evaluates to true in this scenario, since

```
(@post.audience= 'Friends'
 and @self.profile.friends->includes(@caller))
```

evaluates to true when replacing @post by Alice's photo, @caller by Bob, and @self by Alice's timeline.

S2 Alice has set to 'Only Me' the default audience for posts made by her friends in her timeline. Bob posts a photo in Alice's timeline. *Question: Can Bob see this photo in Alice's timeline?* The answer is Yes, because Bob is the person who posted the photo and Bob is not blocked by Alice. Indeed, the `readPost`'s authorization constraint evaluates to true in this scenario, since

```
(@caller=@post.creator
 and @self.profile.blocks->excludes(@caller))
```

evaluates to true when replacing @post by Alice's photo, @caller by Bob, and @self by Alice's timeline.

S3 Alice posts a photo of herself, Bob and Ted in her timeline, and set its audience to 'Friends'. Then, Bob tags Ted in this photo. *Question: Can Peter see this photo in Alice's timeline?* The answer is Yes. because the audience selected by Alice is 'Friends' and, therefore, after Bob tags Ted the audience is extended to Ted and his friends. Indeed, the `readPost`'s authorization constraint evaluates to true in this scenario, since

```
(@post.audience = 'Friends'
 and @post.creator=@self.profile
 and @post.tags.profiling.friends->includes(@caller)
 and @self.profile.blocks->excludes(@caller)).
```

evaluates to true when replacing @post by Alice's photo, @caller by Peter, and @self by Alice's timeline.

S4 Bob posts a photo of himself, Ted and Alice in Alice's timeline. Alice has setting by default 'Only Me'. Then, Bob tags Ted in this photo *Question: Can Peter see this photo in Alice's timeline?* The answer is No, because the audience selected by Alice by default is 'Only Me', and Peter is neither the person who posted the photo, nor the person who is tagged in the photo. Indeed, the `readPost`'s authorization constraint evaluates to false in this scenario, when replacing @post by Alice's photo, @caller by Peter, and @self by Alice's timeline.

V. Adjusting to changes

Facebook's privacy policy has been in a constant state of flux over the past years [17]. This is certainly the case for Facebook's policy for tagging and posting, which is now explained in its 2014 FAQ [8] as follows:

- *When someone adds a tag to a photo or post I shared, who can see it?*
  When someone adds a tag to something you shared, it's visible to: 1. The audience you chose for the post or photo. 2. The person tagged in the post, and their friends. If you'd like, you can adjust this visibility. You can select Custom, and uncheck the Friends of those tagged box.

Clearly, the possibility of not sharing a post with friends of those tagged in the post was not an option in 2013. In fact, if we consider again the scenarios S1–S4, we notice that our answers to the questions about the visibility of the posts in these scenarios remain valid, except for scenario S3: according to the new Facebook's policy for tagging and posting, the question about whether Peter can see or not

the photo in Alice's timeline depends on whether Alice has checked or not the box 'Friends of those tagged' in her photo.

To adjust our SecureUML model of Facebook's privacy policy to this latest change, we need first to modify our data model for Facebook in order to represent whether or not a post will be visible also to the tagged profile's friends. We do so by adding to the class `Post` a new Boolean attribute `audExt`. Then, we modify accordingly the `readPost(@post)`'s authorization constraint. In particular, we need to replace the last clause in the previous description of the policy for executing the method `readPost` by the following clause:

- anybody can read any post that has its audience selected to 'Friends' and was created by the owner of the timeline, if he or she is a friend of somebody tagged on the post, unless he or she is blocked by the owner of the timeline or *the owner has unchecked the box 'Friends of those tagged'*.

More formally, the permission to execute the method `readPost(@post)` will now have the following authorization constraint:

```
@caller=@self.profile
or (@caller=@post.creator
 and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Friends'
 and @self.profile.friends->includes(@caller))
or (@post.audience = 'FriendsOfFriends'
 and (@self.profile.friends->includes(@caller)
  or @self.profile.friends.friends->includes(@caller))
  and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Public'
 and @self.profile.blocks->excludes(@caller))
or (@post.tags.profiling->includes(@caller)
 and @self.profile.blocks->excludes(@caller))
or (@post.audience = 'Friends'
 and @post.creator=@self.profile
 and @post.tags.profiling.friends->includes(@caller)
 and @self.profile.blocks->excludes(@caller)
 // the following conjunct is new
 and @post.audExt).
```

VI. Analyzing Facebook Privacy Policy

SecureUML has a well-defined semantics that supports formal reasoning about its models. In particular, given a SecureUML model M, we can check that nobody, *for which a given property P holds*, will be allowed to execute a certain method X. Notice that this corresponds to proving that there is no *valid* instance of the underlying data model for which both the method X's authorization constraint and the property P evaluate to true. Crucially, as explained in [4], [5], we can automatically transform this type of problems into first-order satisfiability problems, and then use automated theorem-proving tools to attempt to solve them.

We have applied this methodology to prove, as an example, that nobody will be allowed to read a post in a timeline if this person is blocked by the timeline's owner. First, we have formalized, using OCL, the properties that every valid instance of our data model for Facebook will have to satisfy, for example:

- If someone is blocked by someone else, then the former can not remain friend of the latter. Formally,
  Profile.allInstances()->forAll(p, q|
  p.blocks->includes(q) implies p.friends->excludes(q)).
- Nobody can be blocked by itself. Formally,
  Profile.allInstances()->forAll(p|p.blocks->excludes(p)).

Second, we have formalized, using OCL, the property of being blocked by the timeline's owner as follows:
@self.profile.blocks->includes(@caller),

where @caller refers to the person who wants to read the post and @self refers to the timeline where the posted is posted. Finally, after generating the corresponding satisfiability problem, we have used the SMT solver Z3 [6] to automatically prove the desired property, i.e., that nobody will be allowed to read a post in a timeline if this person is blocked by the timeline's owner.

VII. Related Work

To the best of our knowledge, no previous attempts have been made to rigorously formalize the Facebook privacy policy and, in particular, its policies for posting and tagging. There are, at least, two good reasons for this. First, as the DPC audit [11] has pointed out, "many policies and procedures that are in operation [in Facebook] are not formally documented." Second, the Facebook privacy policy has significantly changed over the past few years [17], in ways not always well-explained, as Zuckerberg has to admit in his blog [24]: "I'm the first to admit that we've made a bunch of mistakes. In particular (...) poor execution as we transitioned our privacy model two years ago."

Now, assuming that the Facebook privacy policy is formally documented, what will be the challenges for modeling this policy? Basically, as [18] discussed in detail, for modeling social networking privacy it is crucial to use a language able to formalize *fine-grained access control policies*. In other words, a basic role-based access control language, as proposed in [14], will only do part of the job. Thanks to its tight-integration with OCL, the language SecureUML [2] can deal with fine-grained access control policies, as we have shown in our case study. Of course, there are other options, but not many when having a formal semantics becomes a hard requirement. For example, XACML [15], which can be considered the standard choice for describing privacy policies, lacks of a formal semantics. In fact, due to this limitation, [18] uses the language Z [19] for specifying fine-grained access control policies. Although an interesting option, we prefer to use SecureUML instead of Z because SecureUML already has "built-in" the notions of role, permission, methods, resources, and authorization

constraints, which, would have to be "encoded" (more or less, naturally) along with the policies, if we were to use Z. Furthermore, SecureUML is designed to support model-driven security [3].

VIII. Conclusions and Future Work

In this paper, we have shown how to use SecureUML [2] to formally model social networking privacy. In particular, we have modeled, using SecureUML, the Facebook policies for posting and tagging. The key feature that makes SecureUML up to this task is its ability to formalize both static and dynamic access control policies, the latter being policies that depend on the run-time satisfaction of authorization constraints. We have also explained how to formally reason about SecureUML models, using as an example our model of the Facebook policy for posting and tagging. In fact, this is one of the main benefits of using a formal language, such as SecureUML, to model social networking privacy. Based on this formal foundation, we envision the design and development of new, more powerful privacy tools which, as requested by the DPC audit[11], will provide an "enhanced ability for users to make their own informed choices based on the available information." Furthermore, this formal foundation opens the path for more rigorous comparisons between privacy policies of different social networking sites.

Acknowledgements

This work is partially supported by the Spanish Ministry of Economy and Competitiveness Project "StrongSoft" (TIN2012-39391-C04-04) and by the EU FP7-ICT Project "NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems" (256980).

References

[1] A. Acquisti and R. Gross. Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook. In G. Danezis and P. Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *LNCS*, pages 36–58. Springer, 2006.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM ToSEM*, 15(1):39–91, 2006.

[3] D. A. Basin, M. Clavel, and M. Egea. A decade of model-driven security. In R. Breu, J. Crampton, and J. Lobo, editors, *SACMAT*, pages 1–10. ACM, 2011.

[4] M. Clavel, M. Egea, and M. A. G. de Dios. Checking unsatisfiability for OCL constraints. *Electronic Communications of the EASST*, 24, 2009.

[5] C. Dania and M. Clavel. OCL2FOL+: Coping with Undefinedness. In J. Cabot, M. Gogolla, I. Ráth, and E. D. Willink, editors, *OCL@MoDELS*, volume 1092 of *CEUR Workshop Proceedings*, pages 53–62. CEUR-WS.org, 2013.

[6] L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[7] Facebook. Facebook Help Center. 2013. http://www.facebook.com/help.

[8] Facebook. Facebook Help Center. 2014. http://www.facebook.com/help.

[9] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.

[10] R. Gross, A. Acquisti, and H. J. Heinz. Information Revelation and Privacy in Online Social Networks. In V. Atluri, S. D. C. di Vimercati, and R. Dingledine, editors, *WPES*, pages 71–80. ACM, 2005.

[11] Irish Data Protection Commissioner. Facebook Ireland Ltd. Report of Audit, December 2011. http://www.dataprotection.ie/documents/facebook%20report/final%20report/report.pdf.

[12] Irish Data Protection Commissioner. Facebook Ireland Re-Audit Report, September 2012. http://www.dataprotection.ie/documents/press/Facebook_Ireland_Audit_Review_Report_21_Sept_2012.pdf.

[13] M. L. Johnson, S. Egelman, and S. M. Bellovin. Facebook and privacy: it's complicated. In L. F. Cranor, editor, *SOUPS*, page 9. ACM, 2012.

[14] J. Li, Y. Tang, C. Mao, H. Lai, and J. Zhu. Role based access control for social network sites. In *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pages 389 –394, dec. 2009.

[15] OASIS. eXtensible Access Control Markup Language (XACML), 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[16] Object Management Group. Object constraint language specification version 2.3.1. Technical report, OMG, 2012. http://www.omg.org/spec/OCL/2.3.1.

[17] N. O'Neill. Infographic: The History of Facebook's Default Privacy Settings. http://www.allfacebook.com.

[18] A. Simpson. On the Need for User-Defined Fine-Grained Access Control Policies for Social Networking Applications. In *Proceedings of the workshop on Security in Opportunistic and SOCial networks*, SOSOC '08, pages 1:1–1:8, New York, NY, USA, 2008. ACM.

[19] J. M. Spivey. *The Z notation: a reference manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[20] Y. Wang, P. G. Leon, K. Scott, X. Chen, A. Acquisti, and L. F. Cranor. Privacy nudges for social media: an exploratory facebook study. In L. Carr, A. H. F. Laender, B. F. Lóscio, I. King, M. Fontoura, D. Vrandecic, L. Aroyo, J. P. M. de Oliveira, F. Lima, and E. Wilde, editors, *WWW (Companion Volume)*, pages 763–770. International World Wide Web Conferences Steering Committee / ACM, 2013.

[21] A. Young and A. Quan-Haase. Information Revelation and Internet Privacy Concerns on Social Network Sites: A Case Study of Facebook. In *Proceedings of the fourth international conference on Communities and technologies*, pages 265–274, New York, NY, USA, 2009. ACM.

[22] E. Zheleva and L. Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *WWW*, pages 531–540. ACM, 2009.

[23] M. Zuckerberg. Facebook and the Irish Data Protection Commission. The Facebook Blog, Dec. 2011. https://blog.facebook.com.

[24] M. Zuckerberg. Our Commitment to the Facebook Community. The Facebook Blog, Nov. 2011. https://blog.facebook.com.