Reactivity and Social Data: Keys to Drive Decisions in Social Network Applications[⋆]

Philipp Kärger[1,2], Emily Kigel[2], and Daniel Olmedilla[3]

[1] L3S Research Center, Hannover, Germany
[2] Leibniz University Hannover, Germany
[3] Telefónica Research & Development, Madrid, Spain

Abstract. Social Network applications are gaining momentum. However, equally important, privacy is being shown a crucial requirement. Nowadays, privacy preferences on Social Network applications consist only on allowing or restricting access to information based on attributes of users who are part in the very same network. This paper tries to enhance privacy and provide automatic reactions to events via a very flexible specification of privacy policies and the reasoning associated to them. In our approach it is possible to include Social Semantic data exposed on the Web into the policy definition and reasoning process. We introduce the notion of reactive Semantic Web policies offering higher control of the communications and interactions among Social Network applications and/or its users. We also present SPoX (Skype Policy Extension), which is an implementation that allows policy-driven behaviour control based on the Social Network and communication software Skype, including the capacity of automatically react in certain situations based on user-defined reactive policies such as, for instance, to automatically deny or let through Skype calls and messages based on existing online Social Web data.

1 Introduction

With the advent of the Social Web the possibilities to communicate via Internet increased dramatically. Among others, the way of communication via so-called Social Network applications[4] such as Twitter, Skype and Facebook plays an increasing role. Such applications typically offer plenty of ways to communicate among their members, examples are chat messages, wall posts, and micro-blogging posts. Moreover, in addition to such messages, a lot of communication happens via notifications. For example, in Skype the change of a contact's on-line status is shown in a small notification pop-up. Other kinds of notifications include friendship requests, notifications about being tagged in a picture, etc. In any case, independently of whether a message comes from a single user or if it is a notification generated by the Social Network application, users typically want to canalize the way such messages approach them. For that reason, Social Network applications allow their users to set up simple preferences or policies

[⋆] This work was partially supported by the FP7 EU Project OKKAM (contract no. ICT-215032)
[4] With this term we refer to any kind of application that is based on a Social Network.

about which notification they are interested in and which message or who is allowed to approach them in certain ways and contexts.

Such privacy preferences are typically limited in several respects. Stating for one Social Network application that only friends are allowed to leave wall posts in my profile does not cover people defined as friends on another application. Skype, as an example, is only aware of contacts explicitly listed in Skype. Consequently, it may happen that a call from someone who is a friend on Twitter or listed in one's FOAF[5] profile is blocked. This only happens because one did not explicitly (and redundantly) add the caller as a Skype contact. Generally speaking, any kind of social data that can be gathered from the Web shall be considered for such decisions. Recent studies for mobile phone usage actually show that disturbance of messages heavily depends on the social context of their origin [1, 2]. For example, one may not mind to be approached by people that regularly post in one's blog; or accept friendship requests from people one follows on Twitter. All these information about social contacts is available on the Web: with whom one discusses in forums (e.g., represented in SIOC[6]), whom people know (e.g., represented via a FOAF profile), with whom they are working, be it on publications (e.g., listed in DBLP[7]) or on software projects (e.g., DOAP[8]). Unfortunately, current solutions for privacy preferences do not make use of it [3] and suffer from the "walled garden" of Social Networks [4].

In this paper we propose to incorporate Social Web data into the evaluation of privacy preferences and, more generally speaking, into the definition of behaviour control of Social Network applications. We base our approach on a reactive extension of Semantic Web policies. We describe a language expressing such policies and an algorithm to evaluate them. With this language, policy decisions can be made based on concepts defined by different social aspects; that is, by using logic rules, new concepts can be built from formerly isolated Social Web concepts. We further describe an implementation of this language and our prototype SPoX (Skype Policy Extension)[9]. SPoX is an extension to Skype clients that allows users to easily specify policies that can, for example, express who is allowed to approach the user under which conditions. With SPoX a user can state that only friends on Twitter and Flickr or people listed in the user's FOAF profile are allowed to call and calls from all other users are automatically blocked and turned into a chat. For evaluating policies, SPoX is able to consider context information like time and online status of the user as well as social data about other users that is exposed on the Semantic Web or accessible via APIs of proprietary Social Network applications. Based on this information, SPoX automatically reacts to events in Skype's Social Network according to user-defined reactive policies.

In summary, the contributions of this paper are:

1. integration of social data into logic-based policies thus extending the expressiveness of privacy preferences and behaviour control in Social Network applications by crossing the borders of on-line Social Networks,

[5] Friend-Of-A-Friend, `http://www.foaf-project.org`
[6] Semantically-Interlinked Online Communities, `http://sioc-project.org`
[7] available for example in an Sparql Endpoint such as `dblp.L3S.de/d2r/`
[8] Description-Of-A-Project, `http://trac.usefulinc.com/doap`
[9] downloads and screencast at `www.L3S.de/∼kaerger/SPoX`

2. an extension of Semantic Web policies with events and triggers to adapt to the reactive nature of interactions and communication happening on nowadays Social Web,
3. an implementation of a flexible policy engine that handles such reactive policies; and SPoX, a prototypical extension for Skype clients, that allows a user to easily specify policies and evaluates and reason over them to automatically drive the behaviour of Skype accordingly.

This paper is structured as follows. We first motivate the usage of reactive policies in Section 2 by naming a set of potential useful policies. In Section 3 we provide background information on policies and negotiations. Section 4 introduces our proposed language and how it connects to the Social Web. There, we further discuss the possibilities to identify entities across the borders of Social Networks. An implementation is described in Section 5 and some related work is mentioned in Section 6. Finally, we conclude the paper with discussions about open issues and future work in Section 7.

2 Motivation scenarios

To showcase our approach throughout the paper, we list here a set of simple preferences controlling the flow of messages and the disclosure of information in the context of Social Network applications.

(A) Do not accept Skype calls unless the caller is either in my contact list or listed as friend in one of my Social Network profiles. For all other people calling me, cancel the call and open a chat.

(B) Show notifications about emails arriving and contacts going on-line on Skype only if the origin is either in my family group on flickr or in my family category on Skype. Never show such notifications if my computer is in presentation mode.

(C) Do not allow wall posts on my facebook profile that contain one of a specified list of bad words.

(D) Automatically accept friend requests on any platform if I ever wrote a paper with the requester or if the requester's website is a blog I regularly comment on.

(E) Only people that attended the conference ISWC2009 are allowed to comment on this blog and to see pictures tagged with `ISWC09`.

(F) Forward tweets from Twitter to my mobile phone if I am offline and the author of the tweet is listed in my FOAF profile.

(G) Do not accept calls by students unless it is in my consultation hour.

(H) Show this blog entry only to friends that work in my company.

3 Background

Semantic Web policies and negotiations: Semantic Web policies [5] are declarative statements with a well-defined semantics expressing the behaviour of a system. A typical use case for policies is the definition of access control rules protecting resources from being unintentionally accessed. For example, the policy stating that only my colleagues are allowed to access files tagged with *work* may look as follows[10]

$$allow(access(File, User)) \leftarrow isTagged(File, \texttt{'work'}), isColleague(User). \qquad (1)$$

Adding the facts "$isTagged(\texttt{'study.doc'}, \texttt{'work'})$." and "$isColleague(\texttt{'Bob'})$." will make the goal "$allow(access(\texttt{'study.doc'}, \texttt{'Bob'})$" succeed if posed against this policy—that is, Bob is allowed to access the document `study.doc`. In this paper, we follow [1], where the user's attention itself is considered a particular resource that is to be protected against the overload of undesired messages. A Semantic Web policy like Policy (1) may trigger negotiations while being evaluated (see [7]). In such negotiations, the decision about whether a certain policy condition holds or not is automatically negotiated among the requester and the policy owner. In Scenario (H), before a blog entry is shown to a user, the server storing the post and the requester may automatically negotiate in order to find out if the requester actually works for the company in question. In such a scenario, proving if a person has a certain property is typically done by means of digitally signed credentials. In this case, as a reply to the request for the blog entry, a request for a credential proving company membership is sent by the server; therefore setting up a negotiation.

PROTUNE*:* We base our approach on the PROTUNE[11] (PRovisional TrUst NEgotiation) framework. PROTUNE [6] aims at combining distributed trust management policies with provisional-style business rules and access control-related actions. PROTUNE features an advanced policy language for policy-driven negotiation and supports distributed credential management and flexible policy protection mechanisms. The PROTUNE policy framework offers a high flexibility for specifying any kind of policy, referring to external systems from within a policy and providing facilities for increasing user awareness, like automatic generation of natural language explanations of the result of a policy's evaluation. The PROTUNE policy language is based on logic programming and as such a PROTUNE policy has much in common with a Logic Program (see Policy (1)).

4 Reactive Semantic Web Policies on the Social Web

As motivated in previous sections, Social Network applications are purely reactive, that is, events occur and they typically require (semi-)automatically handling by the users (or the application doing it on behalf of the user). The approach presented in this paper extends classical access/deny policies as sketched in Section 3 by the notion of events and (re-)actions thus creating so-called reactive Semantic Web policies. This extension follows the event-driven nature of Social Networks [3] that includes the reaction to interactions. Here, the classical approach of allowing or denying access does not apply anymore since communication attempts, tagging people on pictures, or posting on forums require more advanced control that observe events and trigger reactions. In the following section we introduce a policy language which is able to express simple yet powerful reactive policies.

[10] Throughout this paper we will use this simplified logic programming syntax of policies that is also used in PROTUNE [6].

[11] `http://www.L3S.de/protune`

4.1 A reactive policy language

Our policy language is a straightforward extension with reactive rules to the PROTUNE language. Such reactive rules are expressed in the common form of Event-Condition-Action rules (ECA rules [8]) written in the form **ON** *Event* **IF** *Condition* **DO** *Action*. ECA rules are interpreted according to the following standard semantics: in case *Event* occurs and, at the same time, *Condition* is evaluated to true, the action *Action* is performed. In the following, we detail our language, which combines standard PROTUNE policies with reactive policies in the form of ECA rules.

Syntax: The language we developed for our approach is a superset of PROTUNE where policy rules of the following form are allowed:

$$\begin{aligned}
&\texttt{ON}\ \ \mathit{event}(A^e_1, \ldots, A^e_n)\\
&\texttt{IF}\ \ \mathit{conditionL}_1(A^c_{1,1}, \ldots, A^c_{1,m_1}),\\
&\qquad \ldots,\\
&\qquad \mathit{conditionL}_m(A^c_{m,1}, \ldots, A^c_{m,m_m})\\
&\texttt{DO}\ \ \mathit{action}_1(A^a_{1,1}, \ldots, A^a_{1,m_1}),\\
&\qquad \ldots,\\
&\qquad \mathit{action}_m(A^a_{m,1}, \ldots, A^a_{m,m_m}).
\end{aligned} \tag{2}$$

where A^e_i and $A^k_{(i,j)} (k \in \{c, a\})$ are terms[12] and $\mathit{conditionL}_i(\ldots)$ are literals, that is, they either represent $\mathit{condition}_i(\ldots)$ or $\neg\mathit{condition}_i(\ldots)$ for some logical atom $\mathit{condition}_i(\ldots)$.

Example 1. Recalling Scenario (A): a reactive policy that changes a call into a chat based on my local contact list or based on information gathered from a Social Network may look as follows:

$$\begin{aligned}
&\texttt{ON callComesIn(User, Call)}\\
&\texttt{IF}\ \neg\texttt{isInMyContactList(User)},\\
&\qquad \neg\texttt{isMySocialNetworkFriend(User)}\\
&\texttt{DO denyCall(Call), sendChatMessage('Hello ...', User)}.
\end{aligned} \tag{3}$$

By directly extending PROTUNE we are able to combine the definition of reactive behaviour with the declarative definition of policies as the following example shows.

Example 2. For example, the embedding of standard PROTUNE policies allows us to further define what is meant by the predicate `isMySocialNetworkFriend(_)` that was used in Policy (3):

$$\begin{aligned}
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{iAmFollowingOnTwitter(Person)}.\\
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{isMyDBLPCoAuthor(Person)}.\\
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{isFacebookFriend(Person)}.
\end{aligned} \tag{4}$$

[12] As it is common in logic programming, variables are denoted starting with a capital letter throughout the paper.

It is important to note that, in a similar way, one may not only define the concept "MySocialNetworkFriend" but also "FOAF friend of a FOAF friend of mine" or "person living in my city", etc.

Semantics: The evaluation of a policy described in this language is based on the Semantics of ECA rules and (non-reactive) PROTUNE programs (as defined in [6]). Due to space limitations we provide the semantics of our language here in a rather informal fashion: If an event occurs it is checked for every ECA-policy if its event predicate unifies with the detected event. Based on the set of policies that matches the event, a PROTUNE policy is constructed by replacing each matching ECA-policy r_i with a non-reactive PROTUNE policy of the following shape

$$\begin{aligned}\texttt{allow}(\texttt{r}_\texttt{i}) \leftarrow & \mathit{conditionL}_1(A^c_{1,1}, \ldots, A^c_{1,m_1}),\\ & \ldots,\\ & \mathit{conditionL}_m(A^c_{m,1}, \ldots, A^c_{m,m_m})\end{aligned} \tag{5}$$

The resulting policy is queried with goal $\texttt{allow}(\texttt{r}_1)$ (see Section 3). In order to evaluate this goal, the conditions in the body of Rule (5) are evaluated following the Stable Model semantics (as it is done for standard PROTUNE policies [6]). In case the goal succeeds, all the actions associated to r_1 are executed. After the evaluation for r_1 the goal $\texttt{allow}(\texttt{r}_2)$ is evaluated, and so on. It is important to note that we define the policy evaluation to follow the order of the reactive policy rules as they appear in the policy. Although the ordering of non-reactive PROTUNE policies does not matters (due to the declarative nature of Stable Models), the order of the reactive policies may matter since different orders of action executions triggered by different ECA rules may have different effects. This assumption follows the standard interpretation of ECA rules and also the intuitive behaviour one expects, for instance, from standard email filters like the Thunderbird Filter[13].

4.2 Retrieving and combining social data

Looking at the scenarios in Section 2, a reaction to a message or to an event in the Social Network may depend on social data about my network, be it if someone is a FOAF friend (as used in Scenario (A)), a student (Scenario (G)), attended an event (E) , or with whom I wrote a paper (D). Consequently, the condition part of a reactive policy needs to retrieve such data in order to contribute to the decision about whether to execute the action or not.

Semantic Web technologies provide standards and models to make social information easily accessible. Information about users is described in standard formats like RDF, which allows to deal with the data from different sources in a uniform way. In addition, due to the emergence of new Web technologies in the era of Web2.0, many Social Web applications open their platforms for external developers by providing an API in order to access the platform's data. In the following section we show how to technically retrieve such social data for policy reasoning. Subsequently, we elaborate on ways how to exploit this information.

[13] `http://kb.mozillazine.org/Filters_(Thunderbird)`

Connecting Protune to the Social Web. Looking back at Example 1, Policy (4), in order to evaluate the predicate $iAmFollowingOnTwitter(\cdot)$ it has to be connected to the actual Twitter API. For this, PROTUNE offers the so-called in-predicate [9, p.13], that allows to consider external data sources as given facts in the state of the knowledge base. Following this approach, Policy (4) has to be extended with the following PROTUNE rule.

$$\begin{aligned}&\texttt{iAmFollowingOnTwitter}(\texttt{Person}) \leftarrow \\ &\qquad \texttt{in}([\texttt{Friend}], \texttt{TwitterWrapper} : \texttt{getPeopleIAmFollowing}()), \\ &\qquad \texttt{Person} = \texttt{Friend}.\end{aligned} \tag{6}$$

For the evaluation of the in-predicate, PROTUNE requires a wrapper to be implemented that offers a method *getPeopleIAmFollowing()*. The results of that method are bound to the variable `Friend` and, internally, for each result r_i a fact is added to the policies: '`in([`r_i`],TwitterWrapper:getPeopleIAmFollowing()).`' . For our implementation the translation and replacement of such predicates is done automatically and the creation of logic programming-inspired policies is kept away from users as well and replaced by a graphical policy editor. We extended PROTUNE with wrappers that import knowledge from arbitrary Sparql endpoints, RDF files, Twitter, Skype and Flickr (for more details, see Section 5).

Exploiting social data for policy evaluation. Guiding and controlling the behaviour of social software requires conditions for policy decisions to be based on the information which is available on the social software applications. Moreover as most people use more than one Social Platform bearing their social data, the immense volume of data available, if combined, can lead to more fine-grained policies. This data consists of personal data and Social Network information on the one hand and, on the other hand, of user-generated content, like bookmarks, tags, reviews, photos, and blogposts (the so-called object-centered Social Network [4]). Furthermore, with the advent of the Semantic Web, and the Linked Data movement[14] in particular, this social data can be linked to more general concepts revealing information like what is a blog post about, in which country was a photo taken, etc.

In the policy evaluation process all this data, extracted from proprietary Social Platforms or exposed on the Semantic Web, is not isolated anymore. Once the data is retrieved, a policy reasoner can combine the retrieved data from multiple sources of the Social and Semantic Web to create advanced, fine-grained and user-adjusted policies. This is based on the observation that relationships among people are not only extracted from explicitly mentioned links in the network but "citizens form relationships and self-organize into communities around shared interests" [4] thus following the principle of *augmented Social Networks* [10]. Looking back to Scenario (D), I may, for example, define the concepts of "interest-sharers" similar to the concept "MySocialNetworkFriend" in Policy (4) in the following way: from the Skype profile of a caller, one can retrieve her website. Based on SIOC data exposed on this website one can find out if it is a blog I regularly comment on [11]. The second condition in Scenario (D) can be determined by a Sparql query to DBLP that delivers all published papers of a

[14] `http://linkeddata.org`

person. Based on this, it is possible to decide if a caller ever was a co-author of the person to be called.

The challenge of identification. Bridging the walled garden of the Social Web for policy reasoning requires a way of identifying the requester: for example, determining if a caller on Skype is a friend in one's FOAF profile or on Facebook needs information about the caller's identity on other platforms. Our current implementation solves this problem on the one hand by negotiation: a requester is automatically asked to disclose her identity for the required platform. On the other hand, a caller's website URL given in her Skype profile is matched against website URLs given in the FOAF profile. We leave further elaborations on this problem to future work and just name potential solutions here. OpenID[15] may be exploited to identify persons across Social Network boundaries. FOAF+SSL [12] can be used to authenticate requesters. Matching people by using further information about the requester is another option. For example, Okkam[16] may provide a unique id for the requester on the Semantic Web given known attributes, Sindice[17] may reveal other attributes about a person including other account ids. A query to the Google Social Graph API[18] may as well deliver information about a person's identity on other platforms.

5 SPoX (a Skype Policy Extension)

Our approach of controlling the behaviour of Social Network applications by exploiting social data seemed promising to be applied to the Social Network and communication tool Skype. First, Skype's privacy policies controlling who is allowed to do what are very limited: a contact of a Skype user can belong to three classes only: the user's friend, a foreigner, or a person that was blocked by the user. For example, one cannot set up filtering rules based on attributes of contacts such as to which category does a contact belong (see Scenario (B)). Moreover, the way a user is allowed to approach another cannot be filtered. For example, either both, chats and calls, are blocked or none of both (see Scenario (A)). Second, Skype offers an API which eases the retrieval of social data about other users on the one hand and, on the other hand, it allows to easily influence Skype's behaviour. Third, Skype offers a channel separated from chat messages and call streams, that allows to transfer data to other peers. This is particular helpful in order to exchange negotiation messages since no additional channel has to be set up between two peers.

SPoX[19][13] is a reactive policy engine that is influencing the behaviour of a Skype client. SPoX builds on the (non-reactive) policy engine Protune and accesses Skype via the open Skype Java API[20]. To develop SPoX we (a) implemented a reactive extension of classical Semantic Web policies. We further extended Protune in order

[15] `http://openid.net/`
[16] `www.okkam.org`
[17] `sindice.com`
[18] `code.google.com/apis/socialgraph/`
[19] `www.L3S.de/∼kaerger/SPoX`
[20] `developer.skype.com/wiki/Java_API`

to (b) access and reason on Semantic Web data (via Sparql endpoints or RDF files) and social data (via the proprietary Social Platform APIs of Flickr, Skype, and Twitter) and (c) to send negotiation messages over the Skype-inherent application channel[21]. When launching SPoX the user is presented with a small control panel for activating or de-activating SPoX and for opening the policy overview (see Figure 1). In the overview, policies can be activated and de-activated by means of the check boxes on the left, and they can be deleted or modified.

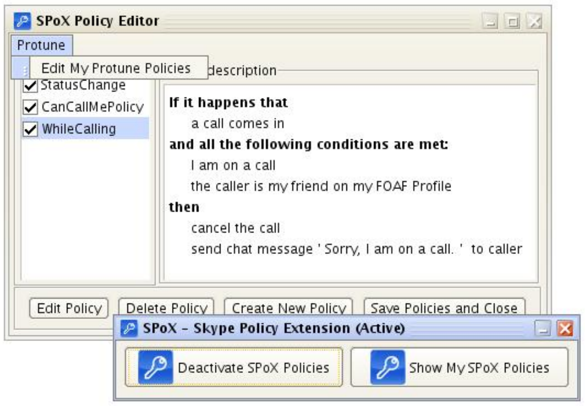
Gathering Social and Semantic Web data in SPoX. In order to incorporate Semantic Web data into the policy decision process of SPoX we extended Protune with a general wrapper that queries Sparql endpoints. In particular, SPoX allows to incorporate co-authorship data from DBLP's Semantic Web endpoint[22]. Another wrapper is able to access arbitrary RDF files, in particular to gather information about a user's contacts on Flickr[23]. Further Social Web data is made available to SPoX by a wrapper accessing FOAF profiles and by accessing the API of Twitter[24].

Defining reactive Semantic Web policies in SPoX. SPoX comes along with an easy-to-use reactive policy editor (see Fig. 2). It is inspired by the design of classical e-mail filters. On the left-hand side, events, conditions, and actions can be selected and added to the policy that is compiled on the right side. The underlined words can be changed in a pop-up that appears by clicking on them. Conditions and actions can be applied either to the initiator of the events (named "the caller" in Fig. 2) or to an arbitrary Skype user. The conditions can be changed between conjunctive and disjunctive connection (by clicking on "one of" resp. "all") and conditions can be negated (by clicking on "is not" resp. "is"). Not all conditions one may impose on a person approaching via Skype can be expressed by such a user interface. Therefore, SPoX allows to define as conditions arbitrary Protune concepts to be fulfilled . This way, new concepts can be defined and exploited for decisions SPoX has to take (as it is done in Policy (6)). SPoX allows to add such concept definitions to the user's Protune policy via the link "Edit my Protune policies" (see Fig. 1) that opens the Protune policy editor. In a SPoX policy's condition, one can refer to these concepts by means of a special condition that allows to freely enter Protune predicates.

6 Related Work

In [1] semantic technology is used to control disturbance by calls on mobile phones. There, among others, a user study is described revealing that social data influences decisions about whether to accept calls or not. However, our work focuses more on the retrieval of social data exposed on the Web (both, Semantic Web and proprietary Social Web Platforms) and is applicable not only to person-to-person communication but includes arbitrary messages. Furthermore, reactions to communication requests

[21] This channel is typically used for game information, see `skype.easybits.com/`.
[22] by means of the Sparql endpoint `dblp.L3S.de/d2r/`
[23] via the RDF exporter `apassant.net/home/2007/12/flickrdf`
[24] `apiwiki.twitter.com`



If it happens that
a call comes in
and all the following conditions are met:
I am on a call
the caller is my friend on my FOAF Profile
then
cancel the call
send chat message ' Sorry, I am on a call. ' to caller

Fig. 1. The control panel and the policy editor with an overview of the current SPoX policies.

in [1] are—due to the scenario addressed—limited to access and deny; whereas our policy language takes a more general approach that allows, for example, to turn calls into chats instead of only denying the call. Besides that, the concept in [1] to disclose the local context to a requester in order to help her decide if a call would disturb its recipient[25] is appealing and we plan to integrate it into our negotiation model (since the disclosure of context may again be protected by a policy). Besides that, a body of other work in the area of semantic reachability management for mobile phones has been carried out [15, 14]. Similar to [1], these approaches share our intention but do not describe how to include social data into privacy decisions, and do not allow to react to messages but limit automation to access and deny communication requests.

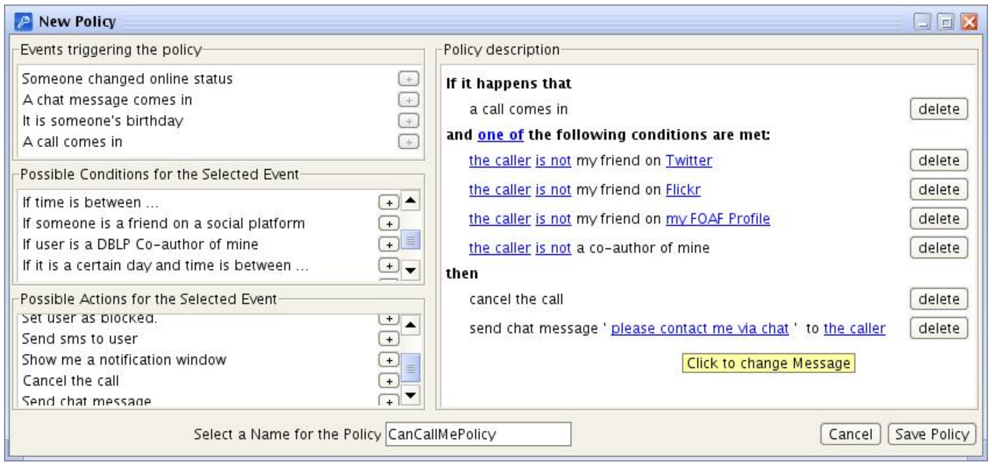
In [16], Golbeck et al. present an email filter based on semantic Social Network information (trust networks). In contrast to our approach, the authors do not allow to directly access social data (like "who is my friend") to decide about trustfulness but describe how to combine available social data into a single reputation score used to rank/filter emails.

Reactive Semantic Web policies have been mentioned already in our previous work [17]. There, reactive policies are not defined in a single language but in a hybrid language that requires two engines to be coupled in an interleaved fashion. In the present work, we describe a single reactive policy language that does not have the drawback of requiring an interleaved interpreter. However, other policy languages that allow for events or actions are available but either they do not allow for policy negotiations or they lack a well-defined semantics.

7 Conclusions, Open Issues and Outlook

In this paper we described an approach to remove the burden of message overload in Social Network applications. We described a reactive policy language that—based

[25] First described in [14].


New Policy

Events triggering the policy

Someone changed online status
A chat message comes in
It is someone's birthday
A call comes in

Possible Conditions for the Selected Event

If time is between ...
If someone is a friend on a social platform
If user is a DBLP Co-author of mine
If it is a certain day and time is between ...

Possible Actions for the Selected Event

Set user as blocked.
Send sms to user
Show me a notification window
Cancel the call
Send chat message

Policy description

If it happens that
a call comes in — delete
and one of the following conditions are met:
the caller is not my friend on Twitter — delete
the caller is not my friend on Flickr — delete
the caller is not my friend on my FOAF Profile — delete
the caller is not a co-author of mine — delete
then
cancel the call — delete
send chat message ' please contact me via chat ' to the caller — delete
Click to change Message

Select a Name for the Policy CanCallMePolicy — Cancel — Save Policy

Fig. 2. Setting up a new SPoX policy.

on social data on the Web—allows to describe what kind of messages are allowed to reach the user. Using our approach we improve the user experience in Social Network applications since (1) communication requests can be handled automatically based on advanced policies (including strong and weak evidences [18], policy negotiations, etc.), (2) messages generated from the Social Platform including news feeds, tweets, etc. can be channeled based on declarative policy statements, and (3) important changes in the Social Network are more visible since they can trigger reactions (e.g., notifications, forward messages).

To further extend this work, we are planning to perform a user study to understand how people actually use reactive policies and how easy it is to express them. We plan to extend our implementation to cater for more Social Platforms and more sources of Social Semantic Web data. Developing a more advanced policy language is on our agenda as well. A new version may include composite events, combinations of actions, and solutions for conflict detection among rules.

We are aware that open linked data is not valid for tough privacy decisions since creating a fake FOAF profile is easy (see Scenario (F)). Therefore, we plan to study how to improve the authentication part of our solution by FOAF+SSL [12] or OpenID[26]. However, our approach currently takes the assumption that messages are sent by co-operative entities.

References

1. Toninelli, A., Khushraj, D., Lassila, O., Montanari, R.: Towards socially aware mobile phones. In: First Workshop on Social Data on the Web (SDoW). (2008)
2. Campbell, S.W., Russo, T.C.: The social construction of mobile telephony: an application of the social influence model to perceptions and uses of mobile phones within personal communication networks. Communication Monographs 7 (4) (2003)

[26] `http://openid.net`

3. Grandison, T., Maximilien, E.M.: Towards privacy propagation in the social web. In: Workshop on Web 2.0 Security and Privacy at the 2008 IEEE Symposium on Security and Privacy. Oakland, California, USA, 18-21 May 2008
4. Breslin, J., Decker, S.: The future of social networks on the internet: The need for semantics. IEEE Internet Computing **11**(6) (2007) 86–90
5. Bonatti, P.A., Duma, C., Fuchs, N., Nejdl, W., Olmedilla, D., Peer, J., Shahmehri, N.: Semantic web policies - a discussion of requirements and research issues. In: 3rd European Semantic Web Conference (ESWC), Budva, Montenegro, Springer (June 2006)
6. Bonatti, P.A., Olmedilla, D.: Driving and monitoring provisional trust negotiation with metapolicies. In: 6th IEEE Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden, IEEE Computer Society (June 2005) 14–23
7. Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K.E., Winslett, M.: No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In: 1st European Semantic Web Symposium (ESWS 2004), Heraklion, Crete, Greece, Springer (May 2004) 342–356
8. Paton, N.W., ed.: Active Rules in Database Systems. Springer, New York (1999)
9. Bonatti, P.A., Olmedilla, D.: Policy language specification. Technical report, Working Group I2, EU NoE REWERSE (February 2005) http://rewerse.net/deliverables/m12/i2-d2.pdf.
10. Jordan, K., Hauser, J., Foster, S.: The augmented social network: Building identity and trust into the next-generation internet. First Monday **8** (2003)
11. Bojrs, U., Breslin, J.G., Finn, A., Decker, S.: Using the semantic web for linking and reusing data across web 2.0 communities. Web Semant. **6**(1) (2008) 21–28
12. Story, H., Harbulot, B., Jacobi, I., Jones, M.: FOAF+SSL: RESTful Authentication for the Social Web. In: Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)
13. Kärger, P., Kigel, E., Jaltar, V.Y.: Spox: combining reactive semantic web policies and social semantic data to control the behaviour of skype. In: ISWC, Poster and Demo Session, Washington, DC, USA. (October 2009)
14. Schmidt, A., Takaluoma, A., Mäntyjärvi, J.: Context-aware telephony over wap. Personal Ubiquitous Comput. **4**(4) (2000) 225–229
15. Stan, J., Egyed-Zsigmond, E., Maret, P., Daigremont, J.: Efficient Reachability Management With A Semantic User Profile Framework. In: Personalization in Mobile and Pervasive Computing (at UMAP2009). (June 2009)
16. James, J.G., Hendler, J.: Reputation network analysis for email filtering. In: In Proc. of the Conference on Email and Anti-Spam (CEAS), Mountain View. (2004)
17. Alferes, J.J., Amador, R., Kärger, P., Olmedilla, D.: Towards reactive semantic web policies: Advanced agent control for the semantic web. In: ISWC 2008 (Poster and Demo Session) Karlsruhe, Germany
18. Bonatti, P., Samarati, P.: Regulating service access and information release on the web. In: CCS '00: Proceedings of the 7th ACM conference on Computer and communications security, New York, NY, USA, ACM (2000) 134–143