

SPOX: combining reactive Semantic Web policies and Social Semantic Data to control the behaviour of Skype

Philipp Kärger
L3S Research Center
Hannover, Germany

Emily Kigel
Leibniz University Hannover
Hannover, Germany

VenkatRam Yadav Jaltar
L3S Research Center
Hannover, Germany

Abstract. In this demo paper we describe SPOX, a tool that allows to define the behaviour of Skype based on reactive Semantic Web policies. SPOX (Skype Policy Extension) enables users to define policies stating, for example, who is allowed to call and whose chat messages show up. Moreover, SPOX reacts to arbitrary events in Skype's Social Network as well, such as on-line status changes of users or the birthday of a friend. The decisions about how SPOX reacts are defined by means of Semantic Web policies that do not only consider the context of the user (such as time or on-line status) but include Social Semantic Web data into the policy reasoning process. By this means, users can state that, for instance, only people defined as friends in their FOAF profile, only friends on Twitter, or even only people they wrote a paper with are allowed to call. Further, SPOX exploits Semantic Web techniques for advanced negotiations by means of exchanging policies over the Skype application channel. This procedure allows two clients to negotiate trust based on their SPOX policies before a connection—for example a Skype call—is established.

Introduction

In the last years, the Semantic Web became a platform that contains more and more Social Data. This data is publicly available and represented in common formats. The most prominent among them are FOAF that allows to describe people and the relation among them, SIOC for online communities, and DOAP for software projects and people working in them. All this social data is out there not only to be simply queried but to be linked and combined in order to reveal implicit information. The work on SPOX is motivated by the fact that all this information is not used for filtering the information flood we are facing [2]. In Skype, for example, one cannot set up filtering rules based on attributes of contacts. Similar to other Social Network applications, 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. With SPOX¹ we present a tool that incorporates Social Semantic Web data into the evaluation and reasoning process of Semantic Web policies. We further use these policies for the definition of behaviour control for Skype. With SPOX policies a user can state, for example, 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 (see policy in Fig. 1). Another SPOX policy could be the filtering of status change notifications: during working hours I may only want to be notified about colleagues going on-line and not about family members or friends.

¹downloads and screencast at www.L3S.de/~kaerger/SPOX

Reactive Semantic Web policies in SPOX

SPOX is a reactive policy engine that is influencing the behaviour of a Skype client. SPOX builds on the (non-reactive) policy engine PROTUNE² and accesses Skype via the open Skype Java API³. To develop SPOX we (a) implemented the recently introduced reactive extension of classical Semantic Web policies [1]. We further extended PROTUNE in order to (b) access and reason on Semantic Web data (via Sparql endpoints or RDF files) and Social data (via the proprietary Social Platform APIs Flickr, Skype and Twitter)[2] and (c) to send negotiation messages over the Skype-inherent application channel⁴. Launching SPOX one is presented with a small control panel for activating or deactivating SPOX and for opening the policy overview (see Figure 2). In the overview, policies can be deleted or activated and de-activated by means of the check boxes on the left.

Reactive Semantic Web policies, in contrast to classical Semantic Web policies, are defined based on events and reactions to such events. SPOX policies are Event-Condition-Action rules (ECA rules) following the standard semantics “if an event occurs and the condition is true, execute the action”. The separation into these three parts is also reflected in the creation of SPOX policies with the graphical user interface (see Fig.1). According to this, the evaluation of SPOX policies follows three steps: in case a Skype Event is detected, all policies are checked if their event parts match the detected event. For each matching policy, PROTUNE is queried to evaluate its condition part. For this, variable bindings instantiated by the events (such as which user is calling, or what is the content of the chat message) are used to translate the SPOX conditions into a PROTUNE policy. If the PROTUNE engine, queried for this policy, returns true for the condition, SPOX calls the Skype API to initiate the actions specified in the reactive policy. The semantics implemented in SPOX evaluates the policies in the order they are listed, and, Skype behaves as if there were no policies unless a policy contradicts.

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

²www.L3S.de/protune

³developer.skype.com/wiki/Java_API

⁴This channel is typically used for game information, see <http://skype.easybits.com/>.

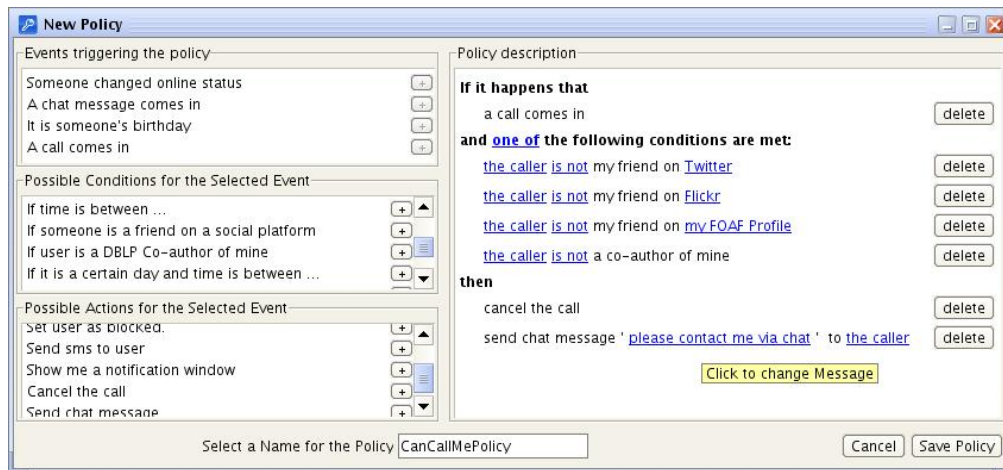


Figure 1: Setting up a new SPoX policy.

DBLP's Semantic Web endpoint⁵. Another wrapper is able to access arbitrary RDF files, in particular to gather information about a user's contacts on Flickr⁶. Further Social Web data is made available to SPoX by a wrapper accessing FOAF profiles and by accessing the API of Twitter⁷

Defining reactive Semantic Web policies in SPoX
 SPoX comes along with an easy-to-use reactive policy editor (see Fig. 1). It is inspired by the design of classical e-mail filters. On the left-hand side, events, conditions, or 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. 1) 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").

Extending SPoX with classical Protune policies
 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. I may, for example, define the concepts of "interest-sharers" based on SIOC data exposed on the Web as follows: if a person's website is a blog I regularly comment on, I may consider this person an "interest-sharer" who I may trust more than a foreigner. Similarly, I may define concepts like "FOAF friends of a FOAF friend of mine" or "people living in the same city". SPoX allows to add such concept definitions to one's PROTUNE policy via the link "Edit my PROTUNE policies" (see Fig. 2) that opens a PROTUNE 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.

Negotiating with Semantic Web policies in SPoX
 Trust Negotiation [3] is a process where two parties can mutually establish trust by a stepwise disclosure of information.

If, for example, a call to my Skype client requires the caller to prove (by means of a digitally signed credential) that she is an employee of my company, I may set up the following PROTUNE policy:

$isColleague \leftarrow sendCredential(C), isCompanyCredential(C).$

Given a request, the SPoX client replies to the requester that the performance of the $sendCredential$ predicate is required. If the requester's conditions to disclose the credential are fulfilled, the credential is sent (if not, a second step of the negotiation is triggered) and the SPoX client will let Skype accept the call. In SPoX, negotiation messages are transmitted over the Skype Game Channel so no separate communication channel is needed.

- [1] J. J. Alferes, R. Amador, P. Kärgler, and D. Olmedilla. Towards reactive semantic web policies: Advanced agent control for the semantic web. In *ISWC 2008 (Poster and Demo Session) Karlsruhe, Germany*.
- [2] P. Kärgler, E. Kigel, and D. Olmedilla. Reactivity and social data: Keys to drive decisions in social network applications. In *2nd ISWC Workshop on Social Data on the Web, CEUR Proceedings, Washington, October 2009*.
- [3] M. Winslett. An introduction to trust negotiation. In *iTrust*, pages 275–283, 2003.

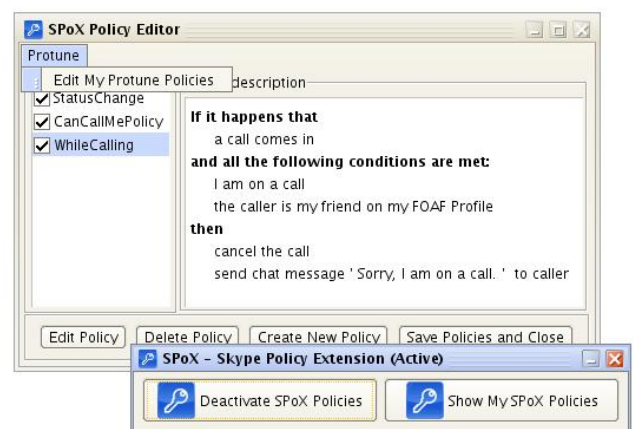


Figure 2: The control panel and the policy editor with an overview of the current SPoX policies.

⁵by means of the Sparql endpoint dblp.L3S.de/d2r/

⁶via the RDF exporter apassant.net/home/2007/12/flickrrdf

⁷apiwiki.twitter.com