Reactive Policies for the Semantic Web*

Piero A. Bonatti[1], Philipp Kärger[2], and Daniel Olmedilla[3]

[1] Università di Napoli Federico II, Italy
[2] L3S Research Center & Leibniz University of Hannover, Germany
[3] Telefónica Research & Development, Madrid, Spain

Abstract. Semantic Web policies are general statements defining the behavior of a system that acts on behalf of real users. These policies have various applications ranging from dynamic agent control to advanced access control policies. Although policies attracted a lot of research efforts in recent years, suitable representation and reasoning facilities allowing for *reactive* policies are not likewise developed. In this paper, we describe the concept of reactive Semantic Web policies. Reactive policies allow for the definition of events and actions, that is, they allow to define reactive behavior of a system acting on the Semantic Web. A reactive policy makes use of the tremendous amount of knowledge available on the Semantic Web in order to guide system behaviour while at the same time ensuring trusted and policy-compliant communication. We present a formal framework for expressing and enforcing such reactive policies in combination with advanced trust establishing techniques featuring an interplay between reactivity and agent negotiation. Finally, we explain how our approach was applied in a prototype which allows to define and enforce reactive Semantic Web policies on the Social Network and communication tool Skype.

1 Introduction

Semantic Web policies [11, 30] attracted a lot of research effort in the last years. Typically, policies are declarative behaviour descriptions used in the upper-most levels in the Semantic Web layer cake, thus establishing trust and enforcing privacy settings among Semantic Web systems. Current Semantic Web policy languages allow to define conditions that must be fulfilled by a peer in order to be allowed to perform a certain action on a system. These languages are based on the assumption, that certain actions (e.g., access to an RDF store [1] or to copy Web content [28]) are only allowed if certain conditions are fulfilled. In this paper, we extend this model to so-called *reactive policies*, which—in contrast to the current Semantic Web policies—allow to define (1) events triggering the evaluation of conditions and (2) actions which are taken as reactions for such an evaluation. Reactive policies make it possible to declaratively define what are the reactions to a specific situation or event within one single language.

* The authors' efforts were partly funded by the European COST Action IC0801 "Agreement Technologies".

This way, reactive policies reflect the dynamics of the Semantic Web: updates in the knowledge stored on some Semantic Web peer are turned into reactions in the real world, thus closing the gap between knowledge and behaviour on the Semantic Web while at the same time ensuring trusted and policy-compliant communication.

Making privacy decisions and establishing trust among systems or agents that do not know each other beforehand is a difficult task. One of the most prominent ways to address this challenge is policy-based trust negotiation [33], a protocol to exchange evidences (typically digitally signed credentials) and policies to mutually establish trust among strangers.

In this paper, we present a policy language to define declarative, exact, and detailed behaviour description of Semantic Web systems with Event-Condition-Action rules (ECA rules [32]). Our framework combines ideas from reactive languages on the Semantic Web [5] with advanced policy reasoning and trust negotiations such that reactivity is based on secure evidences. To achieve this, we formally define a policy language that features an extension to the formal trust negotiation process [13, 12] based on an interplay between negotiations and reactive rule evaluation.

In our language, policies have the form **ON** *Event* **IF** *Condition* **DO** *Action* and are interpreted according to the standard ECA semantics: in case *Event* occurs and, at the same time, *Condition* is evaluated to true, the action *Action* is performed. *Events* are any kind of change in the environment that is propagated by exploiting the infrastructure of the internet; examples are "a flight is rescheduled", "an auction will end in 10 minutes", or "a money transaction is completed". *Conditions*, on the one hand, make use of the mass of information exposed on the Semantic Web such as "is the requester listed in my FOAF[4] profile?" and, on the other hand, they allow for security checks that may, for example, only be carried out by a trust negotiation, such as "is the flight change notification coming from an authorized peer belonging to the airline?" or "is the requester a citizen of the city Hannover?" (to prove those attributes, the exchange of signed credentials is required). Finally, the *actions* will turn the Semantic Web knowledge into real world changes such as "rebook the flight according to my preferences", "redirect the call to my mobile phone", or "dispatch the goods".[5] It is worth noting that the requirement for *reactive* Semantic Web Policies, based on reactive rules, has already been sketched in several publications [12, 11, 3, 21], however, a unified language that combines reactive behaviour control with advanced trust establishing techniques was not given until now.

In summary, the framework for reactive Semantic Web policies presented in this paper has the following features:

- a declarative policy language for reactive policies with well-defined semantics
- seamless integration of Semantic Web sources into the reasoning process
- support for trust negotiations

[4] FOAF=Friend of a Friend, www.foaf-project.org

[5] We refer the reader to [4] where more examples are given for the use of reactive policies on the Semantic Web.

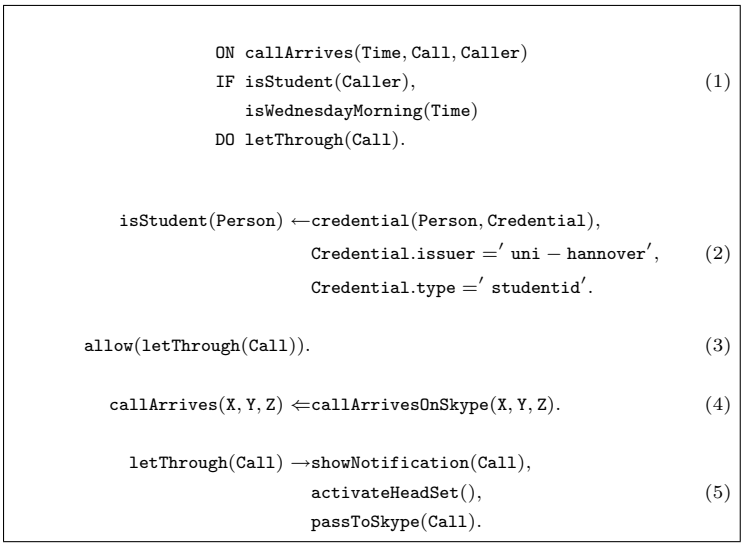

Fig. 1. An example policy accepting student's calls on Wednesday morning.

– support for strong and lightweight evidences

We further illustrate how the presented framework can be used in social systems and describe its use in our application SPoX [22] where reactive policies are enforced for the Social Network and communication tool Skype.

The remainder of this paper is structured as follows. In the next section we will first provide an overview of our policy framework and the principle of trust negotiation. Then, the syntax and the semantics of our policy language is provided. In Section 3 we describe how we used our approach to implement reactive policy control for Skype. In Section 4 we compare our approach to related work and we conclude the paper in Section 5.

2 A framework for reactive policies

Before we formally define the language and the negotiation protocol, we present an informal description of the procedure illustrating the basic principles of our approach.

An example policy is given in Figure 1 where it is stated that phone calls from students are automatically accepted on Wednesday morning. If a call comes in, in order to let Rule (1) apply, the predicate $isStudent(\_)$ has to be proven which requires the predicate $credential(\_, \_)$ to be true. The intuition behind this predicate is, that it is true if the peer given in the first argument provided a credential (which will then be bound to the second argument). In such a case, the fact $credential(peer, cred)$ would be added to the state of the system. Typically, in an initial state, when a call comes in, $credential(\_, \_)$ is not true.

Thus, the policy owner will initiate a request back for a credential. The goal of the procedure is to make $credential(\_, \_)$ true, i.e., adding it as a fact to the state. The reply to the initiator of the call comprises two parts: the request for proving $isStudent(\_)$ and the policy itself that tells the other peer how to prove $isStudent(\_)$. The idea behind this reply is that the policy owner asks the caller about evidences which help to prove $isStudent(\_)$. From the policy that is sent along with the response, the caller learns that a student credential is required. If possible, the caller sends back the credential and the negotiation is successful. Of course, the credential itself can be again protected by a policy (e.g., the calling peer may disclose a student credential only to university employees) which would lead to a request back forming a multi-step negotiation. Finally, if the negotiation is successful and it is Wednesday morning, the action part of Rule (1) can be executed. The execution of actions has to undergo a policy check again (automated re-actions have to be checked to ensure execution safety, as discussed later in the paper) so for each action a that is going to be executed, the predicate $allow(a)$ has to be proven. In our example, accepting the call is always allowed (see empty body in Rule (3)).

2.1 Evidences and filtered policies

Due to space limitations, we do not detail the negotiation model here and remain with the example given above. We only recall two important principles presented in [13, 12] which we adopt for our framework.

Strong and lightweight evidences. Evidences for properties provided by a peer in order allow another peer's decision may be strongly certified by cryptographic techniques, or may be reliable to some intermediate degree with lightweight evidence gathering and validation [11]. Our framework supports both, strong evidences and lightweight evidences. Examples for strong evidences are digitally signed credentials (such as the student ID in Figure 1). Lightweight evidences include so-called declarations, such as the signing of a license agreement or provision of a user name and password.

Policy filtering. As described in the example above, a request for evidences from another peer comprises the policies whose evaluation requires those evidences. This is needed in order to provide the peer receiving the request with reasons why the request is sent (otherwise a calling student would not be able to understand the benefit of providing her student credential). Another reason is that the policies implicitly contain all possible options (e.g., sets of credentials to disclose) to fulfill them. However, policies are sensitive resources and it may be necessary to hide parts of them [12] (see for example Policy (6) in Section 3: since the password itself is part of the policy, sending a non-filtered policy would unintentionally disclose the password). Moreover, not all parts of the policies are important for the receiver of a request which is another motivation for filtering. The policy filtering mechanism and how the filtered policy is used to select a "promising" set of credentials from a portfolio are not included in this paper: we refer to reader to [12] where this principle is described in detail.

2.2 Using external semantic data

Policy decisions may not only depend on credentials that are provided or not, but also on other sources such as if someone is a friend in a FOAF profile (see [23] for more examples) or if it is Wednesday morning (see Figure 1). For this, we support a specific predicate called `in`-predicate that allows calls to external methods to be integrated into the policy evaluation process. With this predicate, ground facts do not have to be present explicitly but can be retrieved on demand from external sources during the reasoning process. In the present work we use this feature to incorporate social data from the (Semantic) Web into the reasoning process as it was presented in [23]. For the `in`-predicate we adopt the syntax introduced in [29].

2.3 Syntax

In the following, we define the Syntax of our policy language. It is composed of three different types of rules: the reactive rules in ECA form (Rule (1) in Figure 1), definition rules to define complex events (Rule (4)) and complex actions (Rule (5)), and implication rules (Rules (2) and (3)) which form the declarative part of the policy and are processed like normal logic programming rules.

A policy P is a set of definition rules, reactive rules and inference rules. A definition rule is either an event definition or an action definition. Let E_a, E_c (atomic and complex events, resp.), A_a, A_c (atomic and complex actions, resp.) be sets of atoms and let e_a,e_c,a_a,a_c be respective elements of those sets.

The set of events E over E_a and E_c is the set of atoms e of the form $e ::= e_a \mid e_1 \bigtriangledown e_2 \mid e_c$ where e_1, e_2 are arbitrary elements of E. Given $e \in E$, an event definition is any expression of the form "$e_c \Leftarrow e$.".

A set of actions A over A_a and A_c is the set of atoms of the form $a ::= a_a \mid a_1, a_2 \mid a_c$ where a_1, a_2 are arbitrary elements of A. Given $a \in A$, an action definition is any expression of the form "$a_c \rightarrow a$.". [6]

Let $Cond$ be a set of atoms. A reactive rule r is any rule of the form "`ON` e `IF` $c_1, \ldots, c_m, not\ c_{m+1}, \ldots, not\ c_n$ `DO` a." with $c_1, \ldots, c_n \in Cond$.

An implication rule is of the form "$head \leftarrow c_1, \ldots, c_m, not\ c_{m+1}, \ldots, not\ c_n$." with $head ::= c_0 \mid allow(a)$ and $c_i (0 \leq i \leq n) \in Cond$, $a \in A_a \cup A_c$.

State predicates. A distinguished subset of predicates, called *state predicates*, may change their extension dynamically (as a consequence of message exchanges, updates, etc.). State predicates comprise `credential`, `declaration`, the `in` predicate (see Section 2.2), as well as predicates that define the attributes of compound objects like credentials and declarations (like the attribute `issuer` of a credential in Rule (2)). Conceptually, in the logical model, state predicates are defined by sets of ground facts, while their actual implementation may be ad

[6] It is worth mentioning that our approach can easily be extended to allow more complex action constructs, e.g., following the ideas presented in [8]. We leave such directions for future work.

hoc or cast into the system on evaluation time. For simplicity, we assume that policy rules do not change during negotiations. On the contrary, the dynamics of the system is reflected in the state predicates whose validity may change.

In contrast to traditional non-reactive trust negotiation approaches, in the presented framework, peer interactions are triggered by a variety of events instead of requests only. For example, in the model in [13, 12] the value of the meta-term *peer* is in general clearly identified by the requester in the negotiation's context. Since in our approach, sources for policy evaluations may not only be requests with a single requester peer, we use binary state predicates for representing credentials and declarations, where the first argument explicitly mentions the peer in charge of providing missing evidence and the second argument denotes the evidence itself. For example a peer may try to make a condition $credential(p, c)$ true by asking peer p for the credential c. Accordingly, we will call *peer variable* of a rule r any variable occurring as the first argument of `credential` or `declaration` in r's body.

Although our language implements negation as failure in general, negation of provisional predicates is prohibited. Provisional predicates are state predicates that are proven by another peer—typically declarations or credentials. As it has been stated in [12], this restriction ensures that policies are monotonic, that is, as more credentials are released the set of permissions does not decrease. Moreover, the restriction on negation makes the implication rules build a stratified program; therefore negation as failure has a clear, PTIME computable semantics that can be equivalently formulated as the perfect model semantics, the well-founded semantics or the stable model semantics [6].

2.4 Semantics

We will first give a procedural description of the policy evaluation process. Subsequently, we will detail this description by a formal definition of the semantics. Procedurally, the evaluation of a reactive policy happens as follows:

1. If an event occurs, the event expressions of all reactive rules are evaluated to determine if the event expression matches the event happened. All reactive rules whose event part applies are called *pending*.
2. For all pending reactive rules r all evidences from the condition part are collected that are not known from the current state but can be proven by other peers (for example, the evidence stating that the origin of a signal is a student and thus can provide a student ID credential).
3. Those requests together with the corresponding filtered policy are sent to the parties defined in the evidence's peer variable and thus initiate a negotiation.
4. If the condition of r is not true but there are remaining evidences which can be proven by other peers, the negotiation goes on.
5. If the condition of r evaluates to true, its variable results are bound to the action part, it is checked if the action execution is authorized (i.e., $allow(action)$ is true) and the actions in question are executed.

Events. An event triggers the evaluation of a rule if it matches the rule's event part as follows:

Definition 1 An event expression e is true for a given atomic event e_A and a reactive policy P, denoted $e \vdash_P e_A$, if one of the following conditions is true

1. e is an atomic event and there is a substitution σ such that $e\sigma = e_A$
2. e is an event expression of the form $e_1 \bigtriangledown e_2$ and there is a substitution σ such that $e_1\sigma \vdash_P e_A \vee e_2\sigma \vdash_P e_A$ (disjunctive connection of events)
3. there is a substitution σ, an event definition $e_c \Leftarrow e_{def}. \in P$, and $e_c\sigma = e$ as well as $e \vdash_P e_{def}$ holds.

We want to note here that this rather simple event algebra can be easily extended by more complex algebras (for example [18]). We leave further extensions in that direction for future work.

Condition evaluation. In the following we define what satisfies a condition part of a reactive rule and which message exchanges and negotiations are needed. We assume the reader to be familiar with the basics of Logic Programming, including SLD-derivations and *mgu* (most general unifier) [6].

Definition 2 [State atom, context, support] A *state atom* (resp. *state literal*) is an atom (resp. literal) whose predicate is a state predicate. A *context* is a set of ground state atoms. (Note that a context is a Herbrand model containing only state atoms.) A *support* of a goal G from a set of implication rules P is a goal G_n containing state literals only and occurring as the last step in a complete, finite SLD-derivation $G, G_1, G_2, \ldots, G_n$. With a slight abuse of notation, goals will be identified with the *set* of literals constituting the goal (instead of a sequence of literals).

We will use contexts to express the set of facts that are true for the current state of the negotiation. Supports will be used to denote the set of predicates that are collected from the policy and that are not true (yet). Those predicates are candidates for requests to other peers in order to collect evidences to prove the initial goal. For example, if a student Alice calls the peer from our running example on a Wednesday 11AM, the peer's context contains the fact $isWednesdayMorning('11AM')$, the support for the goal $isStudent(alice)$ contains $credential(alice, Credential)$. As soon as Alice provides a credential c, the fact $credential(alice, c)$ will be added to the context.

Note that a support is not necessarily a context because, in general, a support is not ground but may still contain unbound variables. Roughly speaking, the support of a goal is the set of literals that are to be verified in order to prove the goal. A goal can only be verified, if the literals in its support hold in the context. Hence, starting from the support of G, the problem of deriving G from the context is reduced to pattern matching against the current context. This is formalized by the following lemma describing the validity of a goal G given a policy and a context.

Lemma 1. *Let P be a set of implication rules of a policy and let C be a context. Then $P \cup C \models G$ iff there exists a support S of G from P and a substitution σ such that $S\sigma \subseteq C$.*

Resulting Actions. Now, since we expressed what satifies the event part and the condition part of a reactive rule, we define our language's semantics by stating which actions are executed and which messages are exchanged given a certain policy, a local state, and a history of events and exchanged messages. The formal framework consists of the following components:

- a (global) *history* (of message/signal exchanges) $\mathcal{H}$;
- a mapping *Pol* associating each peer p to a policy $Pol(p)$;
- a mapping *Ctx* that for each peer p and time t returns a context $Ctx(p,t)$ (called the *negotiation state*);
- a *multiset* of *pending rules* $Pending(p,t)$ for each peer p and time point t;
- a set of *executed actions* $Exe(p,t)$ for each peer p and time point t.

The history $\mathcal{H}$ is a finite set of *messages* $\langle p_1, p_2, M, t\rangle$, where p_1 is the sender, p_2 is the recipient, M is the message content, and $t \in \mathbb{N}$ represents a time point.

Beyond the agents potentially participating in negotiations, the set of peers also comprises some special peers called *environmental peers* that represent the system and the environment in which the system is situated. Environmental peers account for *signals*: events that are not requests or disclosures generated by an agent in the system. Examples for signals are notifications about environmental changes, updates in an RDF document, an arriving call, or the change of the on line status of a peer. The set of possible messages comprises:

- *signals* where M is a ground event and p_1 is an environmental peer;
- *requests* where $M = \langle G, FilteredPol\rangle$; intuitively, p_1 asks p_2 for evidence that can help in proving G from the filtered Policy *FilteredPol*;
- *disclosures* where M is a context consisting of a logical description of a set of credentials and declarations.

The system behavior is constrained as follows:

Pending and executing rules. At any time, a *signal* may activate a reactive rule thus making the rule pending; i.e., its condition part needs to be evaluated as a goal against the set of implication rules (see Lemma 1). This has several effects, including the start of a negotiation. Formally, for all signals $\langle p_1, p_2, e, t\rangle \in \mathcal{H}$ (where e is an event) and for all pending rules $R = (\texttt{ON}\ e'\ \texttt{IF}\ G\ \texttt{DO}\ a)$ in $Pol(p_2)$ such that $e \vdash_{Pol(p_2)} e_A$ and $mgu(e_A, e') = \sigma$, if $G\sigma$ has at least one support from $Pol(p_2)$ then it holds that:

- p_2 has to request for evidences that are to be provided by other peers. Or, more formally, there have to be requests for all peers p_3 occurring in some support of $G\sigma$ from $Pol(p_2)$. Further, $\mathcal{H}$ must contain a request $\langle p_2, p_3, \langle G\sigma, FilteredPol\rangle, t'\rangle$ (with $t' < t$), where *FilteredPol* is a filtering of $Pol(p_2)$ satisfying the faithfulness condition below. To identify correctly the

recipients p_3, the first argument of all `credential` and `declaration` atoms must be ground in all supports (guaranteed by the call-safeness conditions discussed later in the paper)
- the set of pending rules is cumulative, that is $\mathit{Pending}(p_2, t) := \mathit{Pending}(p_2, t-1) \uplus \{R\sigma\}$ for all R that are pending at time t.[7]

If $G\sigma$ does not have any support, that is, if the condition part of R is not fulfilled, then the event e does not cause the execution of the pending rule R.

Faithfulness. Filtered policies always correctly approximate the real policy in the following sense: for all requests $\langle p_1, p_2, \langle G, \mathit{FilteredPol}\rangle, t\rangle \in \mathcal{H}$, and for all contexts C, it holds that

$$\mathit{FilteredPol} \cup C \models G \text{ implies } \mathit{Pol}(p_1) \cup C \models G\,.$$

Note that the converse is not required to hold: not all conclusions that are valid in $\mathit{Pol}(p_1) \cup C$ also hold in the filtered version $\mathit{FilteredPol} \cup C$. This is due to the fact that policy filtering may remove relevant information from $\mathit{Pol}(p_1)$ during filtering either for confidentiality or efficiency (see Section 2.1).

Disclosure Safety. Each disclosure during a negotiation process must be authorized by the local policy. Formally, for all $\langle p_1, p_2, C, t\rangle \in \mathcal{H}$ (where C is a context), and for all atoms $P(p_1, e) \in C$ where $P \in \{\texttt{credential}, \texttt{declaration}\}$,

$$\mathit{Pol}(p_1) \cup \mathit{Ctx}(p_1, t) \models \mathit{allow}(\mathit{release}(e))\,.$$

Relevance. No evidence should be disclosed without need, that is, all disclosed evidence should be relevant to some previous request. Our notion of relevance is justified by Lemma 1: since all the proofs of a goal G depend on some support, we require all disclosed evidence to occur in some of those supports. Formally, for all disclosures $\langle p_1, p_2, C, t\rangle \in \mathcal{H}$ (where C is a context), and for all atoms $P(p_1, e) \in C$ where $P \in \{\texttt{credential}, \texttt{declaration}\}$, there exists a time point $t' < t$ and a request $\langle p_2, p_1, \langle G, \mathit{FilteredPol}\rangle, t'\rangle \in \mathcal{H}$ such that $P(p_1, e)$ belongs to a support of G from $\mathit{FilteredPol}$.

Evidence acquisition. The context is cumulative as well, that is, facts are never removed from the context.[8] Formally, $\mathit{Ctx}(p, t) := \mathit{Ctx}(p, t-1) \cup \bigcup\{C \mid \langle p_1, p, C, t\rangle \in \mathcal{H}\}$.

[7] In a real system, pending rules may not always be simply accumulated: they may be eliminated when the corresponding negotiations terminate or a timeout occurs. For the sake of simplicity we make this conceptual simplification.

[8] This is again a conceptual simplification: in real systems the elements of a context are first cryptographically verified, and—as a consequence—some may be discarded and thus removed from the set of facts in the context; moreover, context elements may be eliminated when the corresponding credential expires.

Execution Safety. $Exe(p,t)$ is the set of all actions that are executed as a result of a pending rule's successful evaluation. Formally, $Exe(p,t)$ is the set of all actions a such that for some $R = (\texttt{ON}\ e\ \texttt{IF}\ G\ \texttt{DO}\ a_1,\ldots,a_n)$ in $Pending(p,t)$, the following two conditions hold

1. $Pol(p) \cup Ctx(p,t) \models G$
2. $Pol(p) \cup Ctx(p,t) \models allow(execute(a))$.

All other reactive rules remain pending. Accordingly, $Pending(p,t+1)$ is the multiset of all rules of $Pending(p,t)$ such that either G or $allow(execute(a))$ is not a logical consequence of $Pol(p) \cup Ctx(p,t)$.

Call Safeness. As it has been stated before, the peer responsible for providing a state predicate has to be instantiated (grounded) in some argument of the atom. For defining this formally, we introduce call safeness as follows.

A *call typing* is a mapping $\tau : \mathsf{Pred} \to \wp(\mathbb{N})$ that associates each predicate symbol p to a set of argument indexes. Intuitively, if $\tau(p) = \{1,3\}$ then the first and third argument are going to be ground in each call to p. We require that $\tau(\texttt{credential}) = \tau(\texttt{declaration}) = \{1\}$.

We extend τ to atoms in the natural way: If $\tau(p) = \{i_1,\ldots,i_k\}$ then for all atoms $A = p(t_1,\ldots,t_n)$ let $\tau(A) = \{t_{i_1},\ldots,t_{i_k}\}$.

An implication rule R with head A is *call safe* iff for all atoms B occurring in the body of R and for all variables X occurring in B, $X \in \tau(B)$ implies $X \in \tau(A)$.

Moreover, a goal G is *call safe* iff for all atoms A occurring in G, $\tau(A)$ is ground.

Proposition 1. *If a goal G is call safe, then all of its supports from a call safe program are call safe, too. In particular, the first arguments of all* `credential` *and* `declaration` *predicates in the support are ground.*

A *reactive* rule $\texttt{ON}\ e\ \texttt{IF}\ G\ \texttt{DO}\ a$ is *call safe* iff for all atoms B occurring in G and for all variables $X \in \tau(B)$, X occurs in e.

This guarantees that the most general unifier of e and the occurred event makes G call safe and, consequently, all credential and declaration predicates in G are associated to specific agents. This condition ensures that at any stage during a negotiation, the receivers of request messages are determined.

3 Implementation

In this section we describe how the approach of reactive policies has been successfully used to provide advanced privacy control for the Social Network and communication tool Skype. Our policy framework allows to enforce policies on Skype such as the one given in Figure 1: calls, notifications, and chat messages are either accepted or initiated only if certain conditions are fulfilled.

Example policies and features exploited. The following example policies show how the features presented in the previous sections are exploited for our scenario. **(A)** Notifications about on line status changes are only shown for specific persons, who may be tagged as family members on some Social Platform. **(B)** Or such notifications are shown if people change the status who can prove that they belong to a certain company by disclosing a company member credential. **(C)** Chat messages may only be accepted if the local computer is not in presentation mode. **(D)** Another way of controlling incoming messages is to require the sender to provide a password before the message shows up.

The use of external Social Semantic Web data is exploited in Example A (retrieve person tags from Social Platforms). Example B requires credential exchange and trust negotiation to ensure a policy aware disclosure of company credentials. Example C integrates external information from the local client into the policy decision process and Example D exploits the declaration predicate for automatically retrieving a password. The policy rule from example D looks as follows:

```
ON chatArrives(Time, Chat, Sender)
IF declaration(Sender, password = A),                    (6)
   not (A =′ let_me_in′)
DO reject(Chat).
```

Those examples fit the scenario of Skype, however, our framework plugged into other systems offers other potentials as mentioned in Section 1, such as the automated rebooking of flights according to user preferences and time schedule (e.g., retrieved from a PIM[9]) [4].

Architecture. We implemented our framework on top of the policy engine Protune [10]. We further developed a Skype plugin that (1) observes events from Skype and passes them to the policy engine, (2) uses the Skype-inherent application channel[11] for transferring negotiation messages and (3) performs actions in Skype, such as cancelling a call, changing the mood message, or sending a chat message. The result was a tool called SPoX (Skype Policy Extension) [22], a reactive policy engine that is influencing the behaviour of a Skype client. For more details about this tool we refer the reader to [22] and to `www.L3S.de/~kaerger/SPoX`, due to space restrictions we will focus in this section on some particular features of this tool directly related to the interplay of reactivity and negotiation handling.

Credentials and declarations. In SPoX, the concept of declarations is used to link users across different Social Web platforms. One can state, for example, that only Flickr friends are allowed to call. In order to find out if a caller is a

[9] Personal Information Manager (e.g., Outlook)
[10] `www.L3S.de/protune`
[11] This channel is typically used for game information, see `http://skype.easybits.com/`.

Flickr friend, a state predicate $flickrName(Peer, Name)$ is used that, according to the semantics definition (see Lemma 1), leads to a request to the call initiator. The SPoX client on the caller's peer will receive the request for the declaration and, if provided by the caller, will send back the instantiated fact thus disclosing her Flickr name. Of course, disclosing the Flickr name can again be protected by a policy and would lead to a multistep negotiation. It has to be noted that for such a declaration there is no easy way to verify if a requester's Flickr name is actually the one provided. However, there are technologies such as OpenID that provide solutions in this case. Digitally signed credentials can be used used as well in SPoX as strong authentication in a similar way. Those credentials are as well transferred using the Skype application channel, e.g., for proving that someone is a student (see the example in Figure 1).

The Social Web as negotiation context. As described in Section 2.2, external sources can be easily integrated in the reasoning process and the gathered facts become part of the context as soon as they are required to make a goal true. In SPoX we exploited this concept for integrating arbitrary external Social Web sources into the negotiation process, such as DBLP (in order to prove co-authorship relations), FOAF, Twitter (to prove if someone is following me), Flickr, and Skype (e.g., prove if somebody is a contact or blocked). Also an extension for gathering social data from any OpenSocial[12] platform has been recently added (cf. [24]).

In conclusion, a combination of reactive behaviour description, Semantic Web data and trust negotiation, as it is needed for an advanced policy control in Skype, is provided by our interplay of triggering events and negotiation message exchange. Our formal model ensures that any action taken in Skype fulfills the given policies, either by adding facts gathered from the Semantic Web to the negotiation state or by exchanging evidences thus mutually and automatically establishing trust among Skype peers.

4 Related Work

The concept of reactive Semantic Web policies touches two research areas: advanced policy reasoning and trust management on the one side and reactive rules on the Web on the other. In the following, we will detail related approaches from both areas separately. As a third category, we compare our approach to general policy languages that support a certain notion of reactivity.

Reactivity on the Web. Adding declarative, reactive rules to enrich systems with reactivity features has been extensively studied in the 90s in the area of database systems, with the introduction of Active Databases [32]. Later, in order to specify the reactive behavior of Semantic Web agents, the concept of Reactivity on the (Semantic) Web has been introduced in several publications [19, 27, 26, 5]. Several frameworks have been developed for reactive rules on the Semantic

[12] see `www.opensocial.org`

Web, e.g., r^3 [2] and MARS [9]. Although these approaches share our strategy to provide reasoners for reactive rules, a general difference is that the present work adds a level of trust on top of reactive rule reasoning by an explicit handling of credentials, interactions among agents, contextual disclosure of information and the exchange of policies and evidences.

Semantic Web policies Among the most prominent Semantic Web policy languages there are KAoS [16], Rei [20], and Protune [12]. These policy languages allow for automated reasoning in order to enforce a policy and, with most of them, policies can also be exchanged between entities [7, 10, 12, 17] which is essential for automated agent interaction and negotiations on the Semantic Web. Therefore, they are typically based on languages with well-defined semantics and interoperable formats; that is in most of the cases description logics (e.g., KAoS and Rei) or rule-based semantics (e.g., Cassandra [7], Peertrust, and Protune).[13] However, the semantics of current Semantic Web policy languages is restricted to definitions of the conditions to be fulfilled in order to make a decision. In contrast, the present approach combines handling of general events and triggering of reactions with trust negotiation and credential exchange.

Policies and reactivity There are other policy languages that do not appear in the realm of the Semantic Web and some of them support a certain level of reactivity. Ponder [15] (and its successor Ponder2 [31]) is a policy language for pervasive systems that supports—beyond classical authorization policies—reactive obligation policies. Those obligation policies are specifically used to trigger internal auditing if a security violation happens and are not meant to react to external events. Ponder does also not feature agent interaction and misses a well-defined semantics. Contrarily, PDL (Policy Description Language) [25] is a policy language based on ECA rules that provides a well-defined semantics. PDL, similar to Ponder, does not feature agent interaction or other trust establishing techniques. It further restricts actions to contain only constants and thus does not allow for variable bindings crossing events, condition, and action expressions.

5 Conclusions

Reactive Semantic Web policies combine the features of reactive behaviour control on the Web with the trust and security features of advanced policies. Automatically guiding a system's behaviour on the Semantic Web does not only need an advanced reactive rules reasoner incorporating arbitrary, heterogeneous data sources, it also needs to consider the upper most layers in the Semantic Web stack: obeying as well as enforcing Semantic Web policies, automated agreement with other systems and trusted interactions with Semantic Web agents. In this paper, we describe a policy language that allows to define reactive behaviour

[13] For a more comprehensive overview and comparison of policy languages we refer the reader to [14].

control in a declarative way and that features advanced trust negotiation based on exchanges of evidences and policies. We gave a formal definition of our language's semantics and defined how the exchange of information may lead to an establishing of trust. We successfully exploited our approach to enforce reactive Semantic Web policies in the Social Network and communication tool Skype.

References

1. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *ISWC 2007 + ASWC 2007*, volume 4825, Busan, Korea, November 2007. Springer.
2. José Júlio Alferes and Ricardo Amador. r^3- A foundational ontology for reactive rules. In *ODBASE'07, LNCS 4803*. Springer.
3. José Júlio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. Towards reactive semantic web policies: Advanced agent control for the semantic web. In *ISWC2008, Poster and Demo Session, Karlsruhe, Germany*, October.
4. José Júlio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. Towards reactive semantic web policies—motivation scenario and implementation details. Technical report, L3S Research Center, www.L3S.de/∼kaerger/reports/reactive_policies.pdf, October 2008.
5. José Júlio Alferes and Wolfgang May. Evolution and reactivity for the web. In *Reasoning Web*, pages 134–172, 2005.
6. C. Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press, 2003.
7. Moritz Y. Becker and Peter Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *POLICY2004*.
8. Erik Behrends, Oliver Fritzen, Wolfgang May, and Franz Schenk. Combining eca rules with process algebras for the semantic web. In *RuleML*, pages 29–38, 2006.
9. Erik Behrends, Oliver Fritzen, Wolfgang May, and Daniel Schubert. An eca engine for deploying heterogeneous component languages in the semantic web. In *EDBT Workshops*, pages 887–898, 2006.
10. P.A. Bonatti and P. Samarati. Regulating service access and information release on the web. In *CCS 2000*, pages 134–143. ACM Press.
11. Piero A. Bonatti, Claudiu Duma, Norbert Fuchs, Wolfgang Nejdl, Daniel Olmedilla, Joachim Peer, and Nahid Shahmehri. Semantic web policies - a discussion of requirements and research issues. In *3rd European Semantic Web Conference (ESWC)*, volume 4011, Budva, Montenegro, 2006. Springer.
12. Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden, June 2005. IEEE Computer Society.
13. Juri L. De Coi and Daniel Olmedilla. A flexible policy-driven trust negotiation model. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Silicon Valley, CA, USA, November 2007.
14. Juri L. De Coi and Daniel Olmedilla. A review of trust management, security and privacy policy languages. In *International Conference on Security and Cryptography (SECRYPT 2008)*. INSTICC Press, July 2008.

15. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY*, pages 18–38, 2001.
16. A. Uszok et al. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *POLICY 2003*.
17. Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *ESWS 2004*, Heraklion, Crete, Greece. Springer.
18. N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In *18th VLDB Conference*, 1992.
19. Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, editors. *EDBT Workshops, Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers*. Springer.
20. Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In *ISWC 2003, Sanibel Island, USA*. Springer.
21. Philipp Kärger. Advanced semantic web policies: Evolution reactivities, and priorities. In *ISWC2008, Doctoral Consortium, Karlsruhe, Germany*, 2008.
22. Philipp Kärger, Emily Kigel, and VenkatRam Yadav Jaltar. Spox: combining reactive semantic web policies and social semantic data to control the behaviour of skype. In *ISWC, Demo Session, Washington, DC, USA*, October 2009.
23. Philipp Kärger, Emily Kigel, and Daniel Olmedilla. Reactivity and social data: Keys to drive decisions in social network applications. 2009.
24. Philipp Kärger and Wolf Siberski. Guarding a walled garden - semantic privacy preferences for the social web. In *Proceedings of the 7th Extended Semantic Web Conference*. Springer, June 2010.
25. Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *In Proc. of AAAI*, pages 291–298, 1999.
26. Wolfgang May, José Júlio Alferes, and François Bry. Towards generic query, update, and event languages for the semantic web. In Hans Jürgen Ohlbach and Sebastian Schaffert, editors, *PPSWR*, volume 3208, pages 19–33. Springer, 2004.
27. George Papamarkos, Alexandra Poulovassilis, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *SWDB*, pages 309–327, 2003.
28. Oshani Seneviratne, Lalana Kagal, and Tim Berners-Lee. Policy-aware content reuse on the web. In *8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009.*, pages 553–568.
29. V. S. Subrahmanian, Piero A. Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert B. Ross. *Heterogenous Active Agents*. MIT Press, 2000.
30. Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In *ISWC 2003*.
31. Kevin P. Twidle, Emil Lupu, Naranker Dulay, and Morris Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *POLICY*, 2008.
32. Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.
33. Marianne Winslett. An introduction to trust negotiation. In *iTrust*, pages 275–283, 2003.