

# Towards Reactive Semantic Web Policies— Motivation Scenario and Implementation Details

José Júlio Alferes<sup>1</sup>, Ricardo Amador<sup>1</sup>, Philipp Kärger<sup>2</sup>, and Daniel Olmedilla<sup>2</sup>

<sup>1</sup> CENTRIA, Universidade Nova de Lisboa, Portugal

<sup>2</sup> L3S Research Center & Leibniz University of Hannover, Germany

**Abstract.** The Semantic Web envisions a distributed environment with well-defined data that can be understood and used by machines in order to, among others, allow intelligent agents to automatically perform tasks on our behalf. In the past, different Semantic Web policy languages have also been described as a powerful means to describe a system’s behavior by defining statements about how the system must behave under different conditions and situations. With the growing dynamics of the Semantic Web the need for a reactive control based on changing and evolving situations arises. This paper presents a framework for the specification and enforcement of reactive semantic web policies, which can be used in order to allow agents to automatically perform advanced and powerful tasks, which can not be addressed neither by existing Semantic Web policy languages nor by recent efforts towards reactivity on the Semantic Web.

## 1 Introduction

The Semantic Web envisions a distributed environment with well-defined data that can be understood and used by machines in order to, among others, allow intelligent agents to automatically perform tasks on our behalf. In order to specify how such agents reason and make decisions under different conditions, different Semantic Web policy languages [1–4] have been developed. These Semantic Web policy languages vary in terms of the reasoning mechanisms they use (e.g., Description Logics or Logic Programming) as well as in the expressivity they provide, therefore being able to address different scenarios. Unfortunately, none of them provide means to model the reactive behavior of agents in order to make decisions or perform tasks triggered by external events. For example, an agent could automatically re-book a flight given that the schedule of a meeting has changed, or redirect an incoming VoIP call to a mobile phone in case the receptor is offline.

Recent efforts towards reactivity on the Semantic Web [5–7] focus on the evolution of data and how systems must react to changes, possibly triggering the execution of some actions. Some of these approaches provide the basic mechanisms required for a general-purpose Semantic Web reasoner for reactive rules, but they lack specific features required in order to be applied to behavior control for automated agents<sup>3</sup> such as, among others, specific modelling of the interactions between agents, delegation or contextual disclosure of information (including the policies themselves).

This paper presents a reactive semantic web policy framework that integrates both approaches, Semantic Web Policies and reactive systems, in order to ex-

plot their advantages. Just to mention a few, the benefits include separation between the application logic and the implementation, interoperability, compact representations (possibly to be exchanged among agents), verifiability, reasoning about agent behavior, reusability, and context sensitivity. Thanks to this integration, our framework allows agents to automatically perform advanced and powerful tasks, which can not be addressed alone neither by existing Semantic Web policy languages nor by recent efforts towards reactivity on the Semantic Web. Our framework has been implemented and is freely available.

The paper is organized as follows: Section 2 presents a fictitious but realistic Semantic Web application scenario in order to identify some of the crucial features required to provide the forthcoming Semantic Web systems, focusing on those not sufficiently addressed in the past. Section 3 presents an overview of the state of the art in both, reactive languages for the Semantic Web and Semantic Web policies, pointing out why each of these approaches taken separately, is not enough for dealing with all the identified features. In Section 4 we describe how a reactive framework ( $r^3$ ) and a policy framework (PROTUNE) have been integrated in order to realize reactive policies exhibiting the desired features. Section 5 concludes the paper by summarizing its main contributions and mentioning current and future work.

## 2 A Motivation Scenario

We start this section by presenting a fictitious application scenario based on a so-called SWYPE agent, and several use cases showing the benefits of such an automated and Semantic Web enabled agent. We continue by extracting

<sup>3</sup> In the same way as Description Logics or Logic Programming as general reasoning languages could not be directly used as Semantic Web policy languages, but were used as underlying reasoning mechanisms instead.

the requirements that need to be met in order to make such an agent a reality.

Susan and Tim work at a computer science research institute connected to a university. Therefore, both have to supervise and teach students on the one hand and on the other hand they are involved in international research projects. Both know that communication with project partners is sometimes difficult, e.g., the alignment of meeting dates and locations, the agreement on task distribution, etc. Similar problems apply to the supervision of students: appointments have to be arranged, rooms have to be reserved for the lectures, topics of theses have to be selected and distributed carefully, and so on. In order to cope with these communication issues, Susan and Tim as well as the research partners and the students are using a system called SWYPE.

SWYPE is a very flexible voice-enabled instant messaging solution, since it allows for automated adaptive behavior control. Based on knowledge available on the Semantic Web, on events/notifications, and also on information exchanged between communication peers, SWYPE behaves differently tailored to the user's needs. Moreover, a user's SWYPE client can act as his Semantic Web broker: it may initiate on-line transactions, automatically send notifications, redirect communication flow (e.g., calls or chat messages) and so on.

## 2.1 Use-Cases

*Example 1.* A major concern of Tim, when considering SWYPE, was that his availability to take incoming calls strongly depends on who is calling and at what time. He takes calls from people working at the university and research partners only during business hours. For people working at his institute he takes their calls also outside business hours (as long as he is still in the office). Tim keeps an up-to-date FOAF<sup>4</sup> profile establishing the research projects in which he is involved. Each of these projects provides a DOAP<sup>5</sup> profile containing a complete list of its members. His institute also maintains a FOAF profile listing all its members. Information about the university staff is available online (using some XML markup). Tim's SWYPE client uses all this information to decide how to proceed with an incoming call: should it go through, should a voice message be recorded, should a chat be opened instead, or should the call be routed to his mobile phone, and so on.

*Example 2.* Tim holds a lecture alone, and shares another one with Susan. Since many students are remotely located e-mail, chats and VoIP are commonly used for communication with students. Both Tim and Susan have established strict consultation hours for students to call. Outside those hours, Tim's students are invited to leave

a voice message. For their joint lecture, Tim and Susan decided that Susan will be the main contact point, but if Susan is not available Tim will take the calls. Information concerning students and lectures is made available online by the university. This information is available only to properly authenticated lecturers (each lecturer only has access to its lectures), though explicit delegation to lecture collaborators is possible.

*Example 3.* For Tim's lecture, SWYPE supports him with the coordination of his lecture's time and location. The university's room schedule is available online. Although the rooms are all booked in the beginning of the semester, it may happen that some rooms become unavailable for some unexpected reason (e.g., unscheduled maintenance interventions). As soon as such a conflicting modification is detected by Tim's SWYPE agent, it automatically negotiates a new location and date with (the SWYPE agent of) the administration of the university. According to Tim's personal schedule, the schedule of the students enrolled for the course, and the room restrictions of the university, a new location and time for the lecture is determined. Of course, the information about the personal schedule of the students (as kept in their favorite PIM<sup>6</sup>) is exposed strictly following the student's privacy preferences (still allowing non exposed schedule details to be negotiated). After Tim confirms the new location and time of the course, a notification about the change is automatically sent out to all the involved students.

*Example 4.* In a research project Susan is working for, it has been decided to have a project meeting in Japan in one month. Her SWYPE client informs her that her flight was canceled. Being aware of the meeting's date and location, the SWYPE client is able to automatically (query the online ticket service and) suggest an alternative flight. Once Susan confirms it her SWYPE agent proceeds to supply the payment information: it establishes the needed level of trust between itself and the ticket service and discloses her credit card details in order to purchase the new ticket. Unfortunately, Susan's credit card is not accepted because the ticket service only accepts European credit cards. Luckily SWYPE provides Susan with an explanation of the reason for the denial, therefore allowing her to adjust her SWYPE agent behavior so that a different credit card is disclosed instead. Hence, this time the transaction is finally successful.

## 2.2 Identified Requirements

A Semantic Web communication platform such as SWYPE requires that each node must be both: a communication capable peer (e.g., chats and voice calls), and a

<sup>4</sup> <http://xmlns.com/foaf/0.1/>

<sup>5</sup> <http://usefulinc.com/ns/doap#>

<sup>6</sup> Personal Information Manager (e.g., Outlook).

Semantic Web node (URI addressable, eventually discoverable, and capable of describing themselves on a semantic level). Besides these obvious requirements, to fully address the scenarios from above, such an agent must satisfy a much broader range of requirements, namely programmability, expressivity, and application specific requirements.

**Programmability Requirements.** A SWYPE agent acts as a broker for its user. As such it must reflect at all times the intended behavior of the user. Such behavior can be dynamically changed and personalized by the user herself and will also evolve. To achieve this, the behavior of a SWYPE client cannot be hard-coded, it must be programmable (i.e. user-defined) in a declarative and interoperable way, and at the same time it should allow for integration of external systems (e.g., legacy systems).

*Declarative.* Anyone should be able to use SWYPE. There is no middle man (no programmer or analyst in between), it is the user herself who must describe the behavior she expects from her SWYPE broker, and there is no room for ambiguities. This can only be achieved through declarative languages (with clear semantics) that closely relate to the way a user would express behavior using natural language (like we did in the examples of Sect. 2.1).

*Interoperable.* Such declarative languages must also be interoperable to accommodate, express and share all sort of domain level concepts either basic or derived (from other concepts). This allows to share information among different entities. One needs to be able to talk about concepts regardless of their actual meaning, being flexible enough to accommodate alternative meanings and to derive relations between concepts. For instance, Ex. 3 refers freely to the concept of a *student*. Nevertheless, it is assumed that the meaning of such concept is either shared and defined somewhere (viz. Ex. 2) or that the meaning will be communicated (e.g., sending its definition) during the interaction. Moreover, combining this with ontology-based reasoning mechanisms allows, for instance, for deriving that a member of a project, Tim is working for, is a research partner of Tim (see Ex. 1).

*Heterogeneous.* Unfortunately, a declarative and interoperable language has its limits. When it comes down to express actual behavior its expressivity is limited by the actual set of primitives it provides. A careful look at the examples in Sect. 2.1 will reveal a myriad of basic atomic constructs—e.g., someone is calling, answer a call, transfer a call, RDF and XML queries, and so on (notice also that, e.g., booking a flight might be an atomic construct of a travel domain specific language). It is not realistic to assume that all that information will be imported in a single knowledge base. Instead, interactions with external systems (e.g., interoperability with legacy systems such as a PIM in Ex. 2) need to be integrated in the language.

**Expressivity Requirements.** Behaviour descriptions must have enough expressivity to account for complex agent interactions, explanations and reactivity.

*Complex agent interactions.* An agent needs to interact with other systems or agents. Such interactions may be complex, such as for instance agent negotiations (cf. Ex. 3 and 4). A SWYPE user must be able to adequately describe the intended behaviour of a SWYPE broker during negotiations (among 2 or more peers).

*Reactivity.* A system providing a solution for our scenario should also provide reactive behaviour control. Events—such as the arrival of a call (Ex. 1), the postponement of a lecture (Ex. 3), or a flight cancellation (Ex. 4)—must be part of the system’s behavior description. Complex situations (i.e., events) that result from the composition of simpler events also need to be declaratively specified (e.g. [8, 9]). Reactivity also includes the handling of (re)actions. For example, triggering a ticket purchasing process at an on-line shop (Ex. 4) or the initiation of a negotiation in order to agree on a lecture’s new date and location (Ex. 3). Such (re)actions are, quite often, complex ones obtained by composition of simpler actions (and even conditions and events, e.g. [10]).

*Explanations.* Behaviour descriptions must also account for unexpected behaviors and behavior explanations. In case a SWYPE client behaves unexpectedly, an explanation needs to be provided to the user: stating the reason for such behavior and suggesting ways to avoid it in the future. For instance, in Ex. 4, SWYPE is able to detect an unexpected behavior (payment information rejected) and explains it to Susan (european credit card required). Even in case of proper behavior, an adequate explanation may help the user to understand it better (and improve it, if needed).

**Application Specific Requirements.** Last, but not least, in the background of most real-world scenarios there exist some basic security and privacy requirements. For example, the interaction between SWYPE agents requires some kind of trust establishing communication. For example the purchase of the flight tickets (Ex. 4) requires a payment (e.g., with a credit card). Therefore, information disclosure control is a requirement, therefore allowing agents to provide information only given that some pre-established conditions are fulfilled (e.g., certain level of trust is achieved). Moreover, the access control to any private resource—such as the schedule of a student (Ex. 3), or information about the students enrolled in a lecture (Ex. 2)—should be carefully enforced (e.g., through exchange of credentials). Furthermore, some other more specific requirements such as the modelling of interactions and delegation are typically required (Ex. 2) in most agent-based scenarios [2].

### 3 Where We Are - Policies vs. Reactivity

In the last years, two parallel approaches have been developed in order to address problems like the ones we describe in our scenario. In this section we detail the state of the art in both areas. We also point out the limitations of both approaches and why neither of them alone is able to provide a solution to our scenario.

#### 3.1 Reactivity on the (Semantic) Web

In order to specify the reactive behaviour of Semantic Web agents, the concept of *Reactivity on the (Semantic) Web* has been lately introduced [5–7]. However, adding declarative rules to enrich systems with reactivity features is not a completely new issue. In fact, reactive rules have been extensively studied in the 90s in the area of databases, with the introduction of Active Databases [11]. However, the differences in the requirements, and the new challenges posed by the Web, and in particular by the Semantic Web, justified the proposal of several specific new approaches. In fact, unlike in databases, reactivity in the Semantic Web must cope with a highly distributed and heterogeneous environment, where heterogeneity is present both in data and languages. So, actions and events are not restricted to the database domain (e.g., updates of a field) but far more general actions and events must be made available to the reactivity framework.

Most of the proposed approaches introduce reactivity by *Event-Condition-Action Rules* (ECA) of the generic form: *on Event if Condition do action*. These rules provide a suitable common model for reactivity, with modularisation into clean concepts, with a well-defined information flow, and a clear and well understood semantics: whenever the *Event* occurs, test if the *Condition* is true at the moment of evaluation and, if so, execute the *Action*.

In first approaches, ECA rule languages [12, 5, 13] resembled the triggers developed for active databases, where the events are always somehow related to the modification of data (e.g., XML or RDF). Soon it was realised that the Web required more general forms of events: in the Semantic Web, an event should be the (volatile) perception of an external occurrence, that may be a change of data, but may also be a manifestation of an executed action, or simply an incoming message. Moreover, events may be combined to form more complex events, as already mentioned in the requirements of Section 2.2. Specific languages for dealing with complex events have been defined in, e.g., [8, 9, 14–16]. It was also realised that, to deal with reactivity in the Web, heterogeneity of behaviour and of languages should be a central concern. A framework for reactive behaviour was proposed in [5–7] to deal with the heterogeneity of languages (us-

ing ontologies) within reactive ECA rules (implemented by MARS [17] and  $r^3$  [18]).

As it possibly became clear in this overview, some of these approaches provide general-purpose frameworks which meet some of basic requirements of our use cases, such as declarativity or reactivity. However, these are just mechanisms provided in order to enable a general-purpose Semantic Web reasoner for reactive rules. Unfortunately, they cannot be directly applied (alone) for behavior control for automated agents since they miss many features required such as specific modelling of the interactions between agents, delegation or contextual disclosure of information (see also Section 2.2).

#### 3.2 Semantic Web Policies

*Semantic Web Policies* [1–4] were born in order to address agent based behavior control and interaction modelling. Semantic Web Policies are well-defined statements that guide systems' or agents' behavior, that is, specify how they should take decisions. Although such decisions are typically bound to security and trust management, they also include many other scenarios such as, among many others, network configuration or agent negotiations. Access control frameworks like KeyNote [19] or PolicyMaker [20], which provide a separation between enforcement and decision mechanisms by means of policies, object-oriented languages such as Ponder [21] (which already include some reactive constructions), or privacy languages such as the Platform for Privacy Preferences (P3P) are not expressive enough to address Semantic Web scenarios.

On the other hand, there is a wide offer of policy languages that have been developed to date [1–4], addressing the general requirements for a Semantic Web policy language: expressiveness, simplicity, enforceability, scalability, and analyzability [22]. These policy languages allow for reasoning and can be exchanged between entities on the Semantic Web, therefore being described using languages with well-founded semantics (typically based on description logics or logic programming semantics).

Unfortunately, the semantics of current Semantic Web policy languages is restricted to definitions of the conditions to be fulfilled in order to make a decision. Current policy frameworks allow for the evaluation of complex conditions including external dependencies such as location of the requester, time, ontological definitions of concepts, etc. Typically, a request or requestor has to fulfill these conditions in order to let the request being executed (e.g., access is granted). However, a crucial requirement extracted from our scenario is that any solution has to allow for reactivity. This need for a broader notion of *Semantic Web Policies*, including reactive (ECA) rules, has already been stated in [23]. Unfortunately, among the existing powerful semantic web policy languages, none includes a semantics of reactivity. A reactive policy language requires a clearly defined semantics of an event and

should even allow for (possibly complex) composition of events (see Section 3.1). Beyond such events, definition of actions as *re*-actions to events is also required. Some policy languages already include actions (e.g., [4]), but they simply represent required statements in order to evaluate a condition, and not reactions to events.

## 4 Bridging Reactivity and Policies

Section 3 clearly shows that, currently, both policy and reactivity frameworks for the Semantic Web have limitations when trying to address the requirements stated in Section 2.2. Careful comparison of those limitations additionally shows that the two fields are complementary, and that together they should cover the full set of requirements. On the one hand, policy frameworks provide expressive and trustful condition languages, but they do not address reactivity requirements. On the other hand, reactive frameworks are very general, focus their expressiveness on reactivity issues and do not address other agent related concerns. In this section we briefly describe a reactive framework ( $r^3$ ) and a policy framework (PROTUNE) and how we have integrated both in order to realize reactive policies that are able to address the use cases presented previously.

### 4.1 $r^3$ - An Open Framework for Reactivity

$r^3$  (Resourceful Reactive Rules) is a Semantic Web framework to express reactive behavior with ECA rules [7, 24].  $r^3$  defines a foundational ontology [18] that provides an abstract vocabulary for expressing ECA rules and (on a meta-level) the languages used in those rules. In order to use a language in an  $r^3$  rule, a description of the language according to the  $r^3$  ontology has to be provided.  $r^3$  languages may range from general rule languages to domain specific languages, not excluding compositional and algebraic languages. Figure 1 illustrates how  $r^3$  globally distributed nodes fit into the Semantic Web.

The open network of  $r^3$  nodes cooperates towards realizing reactive behavior in the Semantic Web. Each node interfaces with the Semantic Web both on a service level and on an ontology level. Each  $r^3$  node provides services to load rules and evaluate expressions, and additionally exposes its description (kept internally as an RDF model) as a set of ontologies. This description must include (or import) all the  $r^3$  languages implemented by the engines that form the node. When an expression is submitted to an  $r^3$  node (either as part of a rule or of an external evaluation), the node translates it into an RDF sub-model, merges it into the node model, infers the language used and accordingly routes it to an appropriate engine (local to the current node or not). Through this open and

ontology-based architecture,  $r^3$  aims at achieving both language heterogeneity and Semantic Web transparency of reactive behaviour.

The communication between the three components of an  $r^3$  rule, namely events, conditions, and actions, is achieved by logical variables. When an ECA rule is submitted, its event expression is evaluated, and for each of its (asynchronous) results (i.e. substitution sets), the condition is evaluated filtering (or extending) the returned substitution set. If the resulting substitution set is not empty, it is then submitted as input for the execution of the action. For an example of an  $r^3$  ECA rule see Ex. 5.

*Example 5.* An  $r^3$  ECA rule to book a similar flight on the cancellation of the original flight (based on Ex. 4) is depicted in Figure 5.

The common functionality shared by different  $r^3$  nodes is abstracted using a Java library that includes the Servlet, Client and Core components which interact with abstract language engines (and markup translators), thus facilitating the integration of different languages (either by implementing new engines, or by wrapping old ones). More information concerning the  $r^3$  framework is available online<sup>7</sup>.

### 4.2 Protune - An Expressive Policy Framework

The PROvisional TRust NEgotiation framework PROTUNE [4] aims at combining distributed trust management policies with provisional-style business rules and access-control related actions. PROTUNE's rule language extends two previous languages: PAPL [25], which until 2002 was one of the most complete policy languages for policy-driven negotiation, and PeerTrust [3], which supports distributed credentials and a more flexible policy protection mechanism. In addition, the framework features a powerful declarative meta-language for driving some critical negotiation decisions, and integrity constraints for monitoring negotiations and credential disclosure. PROTUNE provides a framework with:

- A trust management language supporting general provisional-style<sup>8</sup> actions (possibly user-defined).
- An extensible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection.
- A parametrized negotiation procedure, that gives a semantics to the meta-language and provably satisfies some desirable properties for all possible meta-policies.
- Integrity constraints for negotiation monitoring and disclosure control.
- General, ontology-based techniques for importing and exporting meta-policies and for smoothly integrating language extensions.

<sup>7</sup> <http://reverse.net/I5/r3/>

<sup>8</sup> Authorizations involving actions and side effects are sometimes called provisional.

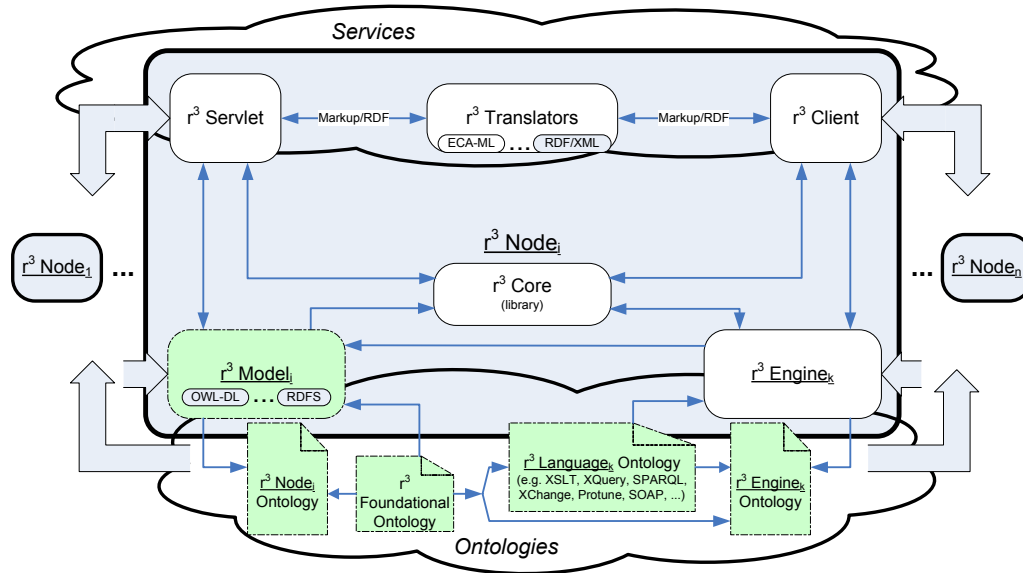


Fig. 1.  $r^3$  in the Semantic Web

Compact form	Expanded form (cf. the $r^3$ ontology)
<pre> ON travel: canceled-flight [   ] ^ OldFlight IF travel: flight-booking [   flight=OldFlight ] ^ Client &amp; travel: flight-similar [   flight=OldFlight,   new-flight=NewFlight ] DO travel: book-flight [   flight=NewFlight,   client=Client ] </pre>	<pre> [] a :Rule; :is eca:rule; :on [a :Component; :is eca:event; :equals [a :Expression; :is travel:canceled-flight; :bound-to [:name "OldFlight"]]]; :if [a :Component; :is eca:condition; :equals [a :Expression; :is log:and; :taking [a :Component; :is log:first; :equals [a :Expression; :is travel:flight-booking; :having [a :Parameter; :is travel:flight; :bound-to [:name "OldFlight"]]; :bound-to [:name "Client"] ]], [a :Component; :is log:next; :equals [a :Expression; :is travel:flight-similar; :having [a :Parameter; :is travel:flight; :bound-to [:name "OldFlight"]], [a :Parameter; :is travel:new-flight; :bound-to [:name "NewFlight"]] ] ] ]]; :then [a :Component; :is eca:action; :equals [a :Expression; :is travel:book-flight; :having [a :Parameter; :is travel:flight; :bound-to [:name "NewFlight"] ], [a :Parameter; :is travel:client; :bound-to [:name "Client"]]]]. </pre>

Fig. 2. An ECA rule that books a similar flight in case of a cancellation.

- Advanced policy explanations<sup>9</sup> in order to answer why, why-not, how-to, and what-if queries [26].

The PROTUNE policy framework offers a high flexibility for specifying any kind of policy, integrate external systems at the policy level and provide facilities for increasing user awareness, like for example, explanations of the policies in natural language. It is entirely developed in Java what permits its integration into web environments as an applet (without requiring the installation of any additional software). Figure 3 depicts the high level architecture of a PROTUNE Agent. More information concerning the PROTUNE prototype is available online<sup>10</sup>.

### 4.3 A Framework for Reactive Semantic Web Policies

$r^3$  and PROTUNE form the basis for our reactive Semantic Web policy framework. On the one hand,  $r^3$  provides general reactive rule reasoning for Semantic Web systems, on the other hand, PROTUNE is a powerful policy language that allows to define Semantic Web agents' behaviors in a declarative way. In this section we describe how we integrated both frameworks and show that the resulting framework is able to meet the requirements of our scenario, and, moreover, that it provides a powerful means for representing and evaluated reactive Semantic Web policies.

As already stated above,  $r^3$  offers the possibility to include other languages ranging from general rule frameworks to domain specific languages. Since PROTUNE offers a Java API, and both  $r^3$  and PROTUNE use logical variables for communication, a wrapper providing  $r^3$  with an implementation for the PROTUNE language was developed with the help of the  $r^3$  Java library. Hence, we enabled  $r^3$  rules to integrate PROTUNE capabilities, i.e. PROTUNE is exposed as an  $r^3$  language. Further details about this language and its implementation are included in Section 4.4. The resulting policy language is reactive:  $r^3$  takes care of triggering the actions, binding variables across the borders of events, conditions, and actions. It passes the bindings to PROTUNE and performs, if needed, the defined actions. PROTUNE provides all the needed means for a powerful automated agent behavior control.

Looking back to Ex. 5, the reactive behavior expressed there is of limited usefulness: it does not take the specific policies for payment and for travel clients personal preferences under consideration. The following example (which fits Ex. 4, where Susan's flight has to be rebooked) shows how our reactive policy language looks like and how it meets the required features extracted from the scenario.

*Example 6.* Policies for rebooking a flight upon cancellation are depicted in Figure 6. The travel agency's policy

includes a reactive rule: given the event that a flight is cancelled, a set of PROTUNE goals are evaluated and, finally, a new flight is booked. The remainder of the (travel agency and Susan) policies are defined by a set of PROTUNE rules (resp. rules T1-T4 and rules S1-S3) in order to declaratively define what has to be evaluated in the condition part of the reactive rule, namely: the definition of an acceptable alternative flight (e.g., favorite airline or airport) and payment information (securely expose credit card information and validate payment information).

We will now revisit the requirements from Section 2.2 and shortly show that our solution provides the desired features.

*Declarative.* PROTUNE is a declarative policy language. It allows a user to state what is to be done and not how. For example, Susan (see Ex. 4) can declaratively specify *what* an alternative flight should be like and not *how* an alternative flight is proven to be acceptable. Such declarative definitions (e.g., rule S1 in Ex. 6) are cleanly integrated into more operational reactive rules, given the clear and intuitive semantics of the latter.

*Explanations.* By combining  $r^3$  and PROTUNE we can provide natural language explanations for any decision taken in the condition part of a reactive policy. In Ex. 4 (where Susan is notified about the cancellation of her flight) a natural language explanation is created informing her that her credit card is not accepted and why. PROTUNE's explanation metapredicate allows us to define general explanation patterns like rules T3 and T4 in Ex. 6. Given those explanation patterns, PROTUNE generates explanations like: *I cannot find a Payment such that Payment is an accepted payment. Because: I cannot find a CreditCard such that CreditCard is a European credit card.*

*Heterogeneous.* PROTUNE also offers the possibility to connect external systems via the the predicate *in/2*. It can be used to plug in any call to external systems on the policy level. That allows, for example, to query a Credit Card validity check Web Service in order to evaluate a policy (see rule T2 in Ex. 6). Furthermore,  $r^3$  innate heterogeneity caters for the composition of PROTUNE conditions with other languages defining events and actions. This allows our framework to be triggered based upon application events and to perform application actions (e.g., a booking a flight). Such atomic events may even be rewritten into generic XML event messages by appropriate  $r^3$  rules, thus allowing the detection of more complex event patterns like those allowed by powerful and declarative event query languages.

*Complex agent interactions.* If the evaluation of a condition in an ECA rule requires a complex interaction, PROTUNE provides the needed infrastructure and protocols for the communication. A PROTUNE condition is evaluated to true if the required information has been

<sup>9</sup> A full demo is available at <http://cs.na.inf.it/reverse/demos/protune-x/demo-protune-x.html>.

<sup>10</sup> <http://www.L3S.de/web/PROTUNE>

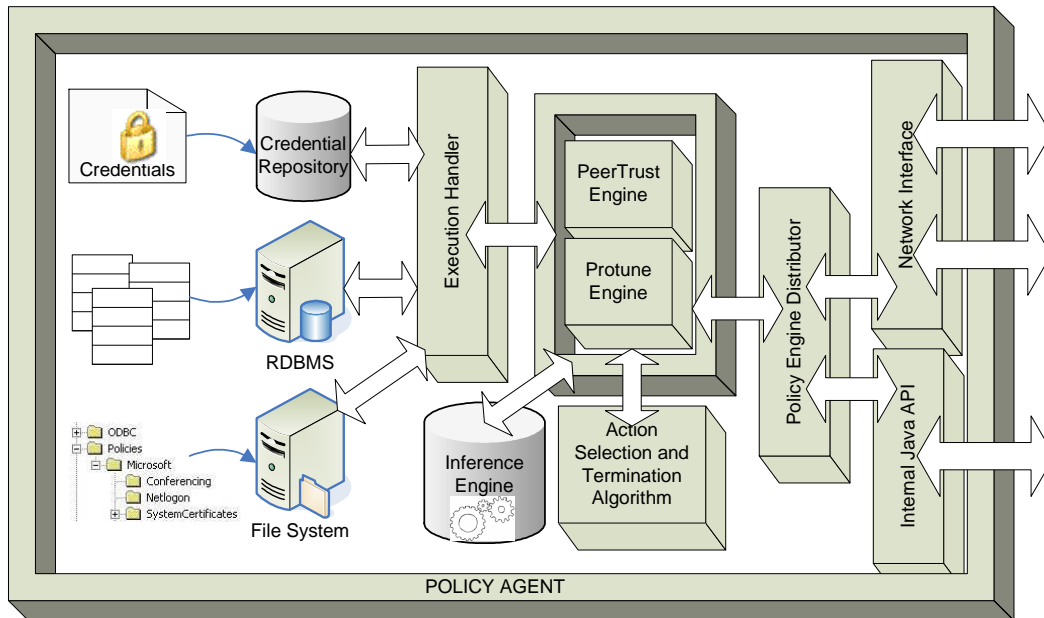


Fig. 3. Policy Framework Architecture

```

/* Travel agency's policy: */
ON travel:canceled-flight [] ^OldFlight
IF travel:flight-booking[flight=OldFlight]^Client &
  travel:flight-similar[flight=OldFlight, new-flight=NewFlight] &
  protune:opaque[peer=Client, literal="acceptAlternativeFlight(OldFlight, NewFlight)"] &
  protune:opaque[peer=Client, literal="sendCreditCard(CreditCard)"] &
  protune:opaque[peer="self", literal="isValid(CreditCard)"]
DO travel:book-flight[flight=NewFlight,client=Client,payment=CreditCard]

(T1) isValid(Payment) :- isEuropeanCreditCard(Payment).
(T2) isEuropeanCreditCard(CreditCard) :- in([], MasterCardWebService:isValid(CreditCard))
(T3) isValid(Payment)->explanation: Payment & " is an accepted payment.".
(T4) isEuropeanCreditCard(CreditCard)->explanation: CreditCard & " is a European credit card.".

/* Susan's policy: */
(S1) acceptAlternativeFlight(Flight, NewFlight) :-
  meetsMyPreferenceConditions(NewFlight).
(S2) sendCreditCard(my-visa-card-number) :- isATrustedService(requestor).
(S3) isATrustedService(Service) :- ...

```

Fig. 4. The travel agency's and Susan's policies for booking and re-booking flights.

exchanged. For example checking if a client’s payment is valid or not (rule T1 in Ex. 6) may require a policy-based trust negotiation because the client may request some prove that the receiver of the credit card is trusted (rule S2 in Ex. 6).

*Interoperable.* Since the entities we envision in our scenario act on the Semantic Web, languages defining their behavior should make use of concept definitions exposed in the Semantic Web. This allows entities to share a common understanding of the concepts they are exchanging information about. In our scenario (see Ex. 3), the concept *student* is shared between Tim’s SWYPE client and the client of the administration. Our solution provides this common definition of a concept since  $r^3$  domain languages are, quite often, fully abstract (non-opaque) languages whose functors and parameters are Semantic Web concepts, and since PROTUNE is also able to connect PROTUNE constants to Semantic Web URIs via the ontology-metapredicate (e.g. `student->ontology:<http://www.university.org/swype#student>`).

#### 4.4 Realizing the Integration of Protune and $r^3$

In order to use a language in an  $r^3$  rule, a description of the language according to the  $r^3$  ontology has to be provided. Such a description includes: all the items of the language, and the implementations available for it (as provided by alternative engine interfaces). Each language must follow a term-based structure (on an abstract level) and its functors may include both symbolic functors (and resp. parameters) and compositional constructs (and resp. parameters and components). A compositional construct, e.g. an algebraic operator, (recursively) builds complex language constructions by composing its—atomic or not—components (e.g., operator arguments).

The parameters of any language functor may have either a functional or a logical nature. The semantics of a language functor is undefined unless all its functional (i.e. input) parameters (e.g., `protune:peer`) are bound to concrete values. Among functional parameters, opaque parameters (e.g., `protune:literal`) may use textual (or markup) sub-languages whose syntax is opaque in what the Semantic Web and  $r^3$  are concerned (only a specific language engine is able to understand them). The PROTUNE language (as used in Ex. 6) was modelled as an  $r^3$  language containing an opaque construct that “digs” a literal PROTUNE goal (see Ex. 7). For more information on the  $r^3$  ontology, see [18].

*Example 7.* PROTUNE and other (partial) language descriptions according to the  $r^3$  ontology are depicted in Figure 5.

Operationally, the  $r^3$  prototype makes no distinction between evaluating a rule event (e.g., `travel:canceled-flight`), condition (e.g., `travel:flight-booking`) or action

(e.g., `travel:book-flight`) element. For any expression,  $r^3$  selects an available implementation for the language involved and submits the expression to the corresponding engine, possibly together with an input substitution set. This engine returns a set of results, each formed by an (optional) functional value and an output substitution set. Results may be returned either synchronously (as is usually the case for conditions and actions) or asynchronously (mandatory for events, with the exception of impossible, e.g. past, events, that return an empty result set). Functional values can only be used by compositional constructs or bound to variables.

The  $r^3$  Java library abstracts all this flow (and communication requirements) and further validates the submitted expressions according to a given  $r^3$  language. Moreover, an evaluation context is provided for, e.g.: fetching parameter values and variable bindings, and binding logical parameters and variables. Therefore, the task of implementing an  $r^3$  engine for a given language is focused on providing (or wrapping) an implementation of that specific language (PROTUNE in our case).

The integration between  $r^3$  and PROTUNE results in an  $r^3$  engine that implements the PROTUNE language included in Ex. 7 by wrapping a PROTUNE peer, and is available online at <http://reverse.net/I5/r3/TST/protune>.

## 5 Conclusions and Further Work

Semantic Web agents allow for automatically performing tasks on users behalf. They can perform tasks that users can not perform, perform worse (in terms of results or time required) or simply that users do not want to perform. However, in order to have intelligent agents that act as users would do, they have to be informed about how users would behave under different conditions and contextual situations. Semantic Web policy languages provide the means to specify statements for automated agent behavior control. Unfortunately, Semantic Web policy languages are not able to model reactivity, therefore allowing agents to react to changes in the environment nor to automatically trigger some actions given than a specific event occurs. On the other hand, existing reactive systems are general-purpose reasoners that do not meet many of the basic requirements needed to use them with agents.

This paper presents how both approaches, Semantic Web policy and reactive frameworks, being complementary, can be integrated in order to sum up the advantages they both provide alone. In such a way, very complex and advanced scenarios can be addressed, therefore giving users expressive and powerful agents able to help in situations that were not addressable nowadays. Our integration of the frameworks  $r^3$  and PROTUNE have been developed and tested, and the result can be downloaded from <http://reverse.net/I5/r3/TST/protune>.

```

protune:opaque a :OpaqueConstruct; :in protune:lang; :uses protune:peer; :digs protune:literal.
protune:peer a :FunctionalParameter; :in protune:lang.
protune:goal a :OpaqueParameter; :in protune:lang.
protune:lang a :Language; :implementation [a :Engine; :providing [a :Interface; ...]].

travel: canceled-flight a SymbolicFunction; :type travel:flight.
travel:flight-similar a SymbolicFunction; :uses travel:flight; :binds travel:new-flight.
travel:flight-booking a SymbolicFunction; :uses travel:flight; :type travel:client.
travel:book-flight a SymbolicFunction; :uses travel:flight, travel:client.
travel:new-flight a :LogicalParameter; :in travel:lang. ...

log:and a :ConjunctionConstruct; :in log:lang; :takes log:first, log:next.
log:lang a :Language; :defines log:first, log:next; :implementation [a :Engine;
...].

```

**Fig. 5.** Language description of PROTUNE according to the  $r^3$  ontology. The  $r^3$  ontology namespace is used as the default namespace.

We are currently working on a prototype application in order to demonstrate some of the scenarios described in the paper. In particular, we are extending some current messaging and VoIP applications in order to allow for automated behavior control. In addition, we also plan to develop appropriate tools for helping users to specify such reactive policies as well as to visualize and monitor them.

## References

1. Uszok, A., Bradshaw, J.M., Jeffers, R., Suri, N., Hayes, P.J., Breedy, M.R., Bunch, L., Johnson, M., Kulkarni, S., Lott, J.: KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In: POLICY. (2003) 93
2. Kagal, L., Finin, T.W., Joshi, A.: A policy based approach to security for the semantic web. In: ISWC 2003, Sanibel Island, FL, USA, 2003. Lecture Notes in Computer Science, Springer (2003)
3. Gavrioloaie, 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: ESWS 2004, Heraklion, Crete, Greece, Springer
4. Bonatti, P., Olmedilla, D.: Driving and monitoring provisional trust negotiation with metapolicies. In: POLICY 2005, Washington, DC, USA, IEEE Computer Society (2005)
5. Papamarkos, G., Poulouvasilis, A., Wood, P.T.: Event-condition-action rule languages for the semantic web. In Cruz, I.F., Kashyap, V., Decker, S., Eckstein, R., eds.: SWDB. (2003) 309–327
6. May, W., Alferes, J.J., Bry, F.: Towards generic query, update, and event languages for the semantic web. In Ohlbach, H.J., Schaffert, S., eds.: PPSWR. Volume 3208 of Lecture Notes in Computer Science., Springer (2004) 19–33
7. May, W., Alferes, J.J., Amador, R.: An ontology- and resources-based approach to evolution and reactivity in the semantic web. In: OTM Conferences (2). (2005)
8. Bry, F., Eckert, M.: Rule-based composite event queries: The language  $xchange^{eq}$  and its semantics. In Marchiori, M., Pan, J.Z., de Sainte Marie, C., eds.: RR. Volume 4524 of Lecture Notes in Computer Science., Springer (2007) 16–30
9. Adaikkalavan, R., Chakravarthy, S.: Snoopib: Interval-based event specification and detection for active databases. Data Knowl. Eng. **59**(1) (2006) 139–165
10. Behrends, E., Fritzen, O., May, W., Schenk, F.: Combining eca rules with process algebras for the semantic web. In Eiter, T., Franconi, E., Hodgson, R., Stephens, S., eds.: RuleML, IEEE Computer Society (2006) 29–38
11. Widom, J., Ceri, S., eds.: Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann (1996)
12. Bailey, J., Poulouvasilis, A., Wood, P.T.: An event-condition-action language for xml. In: WWW. (2002) 486–495
13. Bonifati, A., Braga, D., Campi, A., Ceri, S.: Active xquery. In: ICDE. (2002) 403–
14. Seiriö, M., Berndtsson, M.: Design and implementation of an eca rule markup language. [27] 98–112
15. Bailey, J., Bry, F., Eckert, M., Patranjan, P.L.: Flavours of  $xchange$ , a rule-based reactive language for the (semantic) web. [27] 187–192
16. Paschke, A., Kozlenkov, A., Boley, H., Tabet, S., Kifer, M., Dean, M.: Reaction RuleML. RuleML Initiative, <http://ibis.in.tum.de/research/ReactionRuleML/>. (2007)
17. Behrends, E., Fritzen, O., May, W., Schubert, D.: An eca engine for deploying heterogeneous component languages in the semantic web. In: EDBT Workshops. (2006) 887–898
18. Alferes, J.J., Amador, R.:  $r^3$ - a foundational ontology for reactive rules. In Meersman, R., Tari, Z., eds.: OTM Conferences (1). Volume 4803 of Lecture Notes in Computer Science., Springer (2007) 933–952
19. Blaze, M., Feigenbaum, J., Keromytis, A.D.: Keynote: Trust management for public-key infrastructures (position paper). In: Security Protocols, 6th International Workshop. Volume 1550 of Lecture Notes in Computer Science., Cambridge, April, Springer (1998) 59–63

20. Blaze, M., Feigenbaum, J., Strauss, M.: Compliance checking in the policymaker trust management system. In: Financial Cryptography, Second International Conference. Volume 1465 of Lecture Notes in Computer Science., Anguilla, British West Indies, Springer (February 1998) 254–274
21. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: POLICY 2001: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, London, UK, Springer-Verlag (2001) 18–38
22. Tonti, G., Bradshaw, J.M., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In: International Semantic Web Conference. (2003) 419–437
23. Bonatti, P.A., Duma, C., Fuchs, N.E., Nejdl, W., Olmedilla, D., Peer, J., Shahmehri, N.: Semantic web policies - a discussion of requirements and research issues. In Sure, Y., Domingue, J., eds.: ESWC. Volume 4011 of Lecture Notes in Computer Science., Springer (2006) 712–724
24. May, W., Alferes, J.J., Amador, R.: Active rules in the semantic web: Dealing with language heterogeneity. [27] 30–44
25. Bonatti, P.A., Samarati, P.: Regulating service access and information release on the web. In: ACM Conference on Computer and Communications Security. (2000) 134–143
26. Bonatti, P.A., Olmedilla, D., Peer, J.: Advanced policy explanations on the web. In: 17th European Conference on Artificial Intelligence (ECAI 2006), Riva del Garda, Italy, IOS Press (Aug-Sep 2006) 200–204
27. Adi, A., Stoutenburg, S., Tabet, S., eds.: Rules and Rule Markup Languages for the Semantic Web, First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005, Proceedings. In Adi, A., Stoutenburg, S., Tabet, S., eds.: RuleML. Volume 3791 of Lecture Notes in Computer Science., Springer (2005)