published as: Adrian Paschke, Rehab Alnemr, Christoph Meinel: Rule Responder Distributed Reputation Management System for the Semantic Web
Proceedings of the 4th International Rule Challenges (RuleML 2010), Washington, DC, USA, October, 2010, ISSN: 1613-0073

The Rule Responder Distributed Reputation Management System for the Semantic Web

Adrian Paschke, Rehab Alnemr, Christoph Meinel

Free University of Berlin
AG Corporate Semantic Web
Hasso Plattner Institute
Potsdam University
paschke@inf.fu-berlin.de, first.lastname@hpi.uni-potsdam.de

Abstract. Online Reputation management systems compute, manage, and provide reputation about entities which act on the Web. An important research question is how such a reputation management system can be build for the Semantic Web. In this paper we contribute with a reputation management system based on distributed rule agents, which uses Semantic Web rules for implementing the reputation management functionalities as rule agents and which uses Semantic Web ontologies for representing simple or complex multi-dimensional reputations. We demonstrate a Semantic Web reputation management system by means of an e-Commerce and a Social Semantic Web 3.0 use case, which is using an ontology based reputation object model and a rule-based inference service agent middleware, called Rule Responder. As contribution our solution ensures efficient automation, semantic interpretability and interaction, openness in ownership, fine-grained privacy and security protection and easy management of semantic reputation data on the Web.

1 Introduction

Online reputation management systems provide means for collecting reputation data about entities and computing reputations on the Web; performing actions based on reputations to determine trustworthiness or make automated decisions; allowing entities to manage and interchange their reputation; making sure the system is not abused and privacy of entities is respected so that reputation is only disclosed to authorized parties. Due to the open nature of reputation - where we can not anticipate beforehand what interactions, events, and opinions will finally formulate the reputation of an entity, nor we can know all the contexts in which en entity may gain reputation in the future - processing of reputation should be dynamically changeable by users in a declarative way, easily manageable with high levels of automation, and reputation information should be interchangeable in a well-defined machine-interpretable format. The Semantic Web, with formal languages for machine-readable data, ontologies and rules, is generally considered as a solution for declarative knowledge representation on the Web, enabling high levels of machine intelligence in distributed Web agent systems. In this paper we examine the Semantic Web approach for declarative reputation processing and reputation knowledge representation, and demonstrate a distributed online reputation management system.

Our online reputation management system is based on distributed semantic rule agents, which uses Semantic Web rules for implementing the reputation management functionalities as rule agents and which uses ontologies for representing simple or complex multi-dimensional reputations. We contribute with an ontology based reputation object (RO) model and an implementation based on a rule-based inference service agent middleware called Rule Responder. Because of the semantic structure of a given RO, a semantic agent is able to extract meaningful information from it and then use it directly (i.e. determine trust and access levels) or by using the reasoner for decision support. We describe how we integrate the RO model into Rule Responder agents' logic and how agents are benefiting from the new semantic structure of reputation objects, as opposed to using a single reputation value like in classical reputation approaches.

The paper is structured as follows: We start by explaining our model of reputation object in Section 2, followed by a section with our contribution - the Semantic Web reputation management system. In Section 4 we describe the implementation of this novel design artifact, based on our RuleResponder middleware and our reputation object ontology. Section 5 demonstrates our approach by means of a use case implementation. Finally, we conclude the paper with a summary of the findings of our work.

2 Reputation Management

In this section we briefly describe our previous work on the Reputation Object (RO) model. This model is the basis of reputation data usage and information exchange described in the architecture in section 3.

2.1 Reputation Object Model

Reputation systems are used as a way of establishing trust between unrelated parties, especially if enforcement methods like institutional policies are not implemented. A reputation model describes all of the reputation statements, events, and processes for a particular context. This context is the relevant category for a specific reputation. The way such systems query, collect, and represent reputation varies. Some systems use stars or scaling bars as the visual format of reputation, others use numbers and percentages like the one used to rate an e-markt participant. However, most of these systems use a single value to express an entity's reputation [8] [7]. In most reputation systems, the context of a reputation value is not embedded within the given reputation information. Mostly because it has the single value format. Reputation changes with time and is used within a context. Every domain has its own information sources as well as its own requirements. Therefore, the representation -not the calculation- of reputation should be unified between communities in order to facilitate knowledge exchange. In this paper, we continue our work on a data model for exchanging reputation information between different domains. Enabling reputation portability and linking it to its context eases the management of reputation data, mitigates risks in open environments, and enhances the decision making process. In our previous work [3], we presented a Reputation Object (RO) Model where the representation of an entity's reputation is replaced by an object instead of a single value. The model structure (Figure 1) contains a description of how this value is collected (e.g. by community ratings or monitoring service), the computation function (for this criterion) used to aggregate the values each time a new one is entered, and a history list (previous values dated back to a certain time slot). This

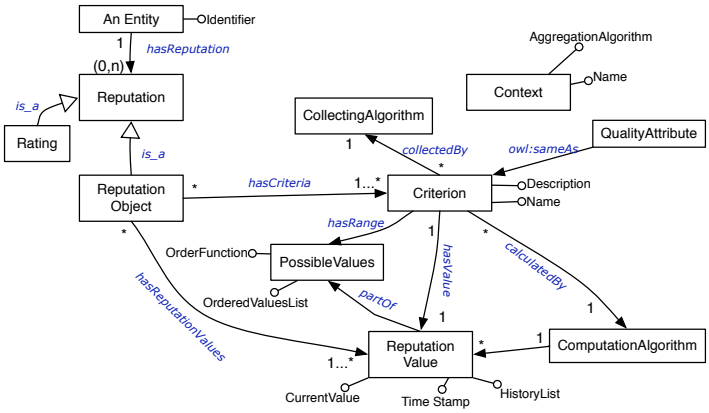

Fig. 1. Reputation Object Model

enables the destination system to map its perception (or its reputation computation function) to the one used in computing this value (i.e. a "very-good" in system A is "good" B). The reputation object in this case is seen as a profile of the entity's expected performance which is constructed using different information sources. Hence,the model achieves several goals:

- the reputation of an entity is more meaningful because it is associated with the context in which it was earned
- automation of criteria assignment is possible by declaring a relevant resource as a criterion (ex. `URI1 is_a _:criterion`)
- one can easily extend these criteria dynamically by adding to the list of contexts/criteria in the reputation objects

A reputation object wraps, into one semantic data structure, information about criteria describing this entity's performance in several context along with information about how they were obtained, what computation algorithms are used, and how to compare between values within each criterion. It is developed as an OWL ontology to ensure integration of data sources, connection of the data to its semantics, efficient information analysis as well as to provide common reputation representation framework. Details about the ontology is illustrated in Using descriptive terms from the ontology a [3]. To illustrate with an example from the e-market domain, a seller's reputation object reflects his expected performance and rating in several criteria such as product-quality, insurance, delivery. The data type of each criterion is dynamic. It can be either a numerical value, a string, or it can be a reference to an object value -maybe linking to another ontology in the linked data store- describing the evaluation of this particular criterion and whether it can be aggregated with others to form a context. Aggregating a set of criteria to a single context can be done to enhance the usability of the reputation object and also to ease the ontology matching process when comparing between two reputation objects. This way an agent using a reputation object, is able to draw more conclusions about the target entity within its correct context. As mentioned before, for each criterion in the object a description of how this value is collected (e.g. community ratings, monitoring), the computation function, and a history of its previous

values are included. This enables the destination system to map its perception (or its reputation computation function) to the one used in computing this value. [3][1][2] The reputation object in this case is seen as a profile of the entity's expected performance which is constructed using different information sources. The degree of visibility for these criteria to the community's users (i.e. how many criteria presented for users in a web site) depends on the community (i.e. a web site can limit the number of criteria for usability reasons).

A RO is constructed either offline or during negotiation process. It's a generic object that changes according to the domain and the user preference but in general it holds a profile (functionality, quality, ratings, etc.) about an entity (service or agent), which is collected from heterogeneous information sources. These information sources include the involved security settings (used within the platform) as well as quality attributes that may be obtained from the used Service Level Agreements (SLAs) such as robustness, availability, etc.. In [2] , we showed the advantages of using a trusted centralized party to manage reputation values within a specific domain or multiple domains. One of the tasks of the Trust Reputation Center (TRC), or a correspondent service, is to gather reports by participants, i.e. other users, services,etc. Due to user privacy concerns, reputation information should not be available for public data retrieval. Rather, the user should have control over which Trust Reputation Centers to use, and which services to allow access to his reputation data. From the perspective of services, the user should not be able to hide reputation information. Hence, a reputation search service is required returning a list of TRCs that store reputation information about the user, without automatically revealing this information. In [4] we introduced a protocol where reputation transfer between different Web sites and platforms is possible while at the same time, the user's privacy is respected and the user retains control over his reputation data without giving him the opportunity to hide valuable information.

3 Architecture: A Rule-based Open Reputation Systems

In this section we describe our new architectural design artifact for a reputation management system which is distributed on the (Semantic) Web. For the architecture we propose a distributed *Reputation Processing Network (RPN)* consisting of *Reputation Processing Agents (RPAs)* that have two different roles:

1. *Reputation Authority Agents (RAAs)*: act as reputation scoring services for the reputee entities whose Reputation Objects (ROs) are being considered or calculated in the agents' rule-based Reputation Computation Services (RCSs). A RCS runs a rule engine which accesses different sources of reputation (input) data from the reputors about an entity and evaluates a RO based on its declarative rule-based computational algorithms and contextual information available at the time of computation (described in the RO by the `Criterion` and its `PossibleValues` along with its `ComputationAlgorithm`, `orderedList`, and `OrderFunction`).
2. *Reputation Management Agents (RMAs)*: -aka reputation trust center- provide reputation management functionalities. A RMA manages the local RAAs providing control of their life cycle in particular and also ensuring goals such as fairness. It might act as a Reputation Service Provider (RSP) which aggregates reputations from the reputation scores of local RAAs. Based on the final calculated reputation, it might also perform actions, e.g. compute trust worthiness, make automated decisions, or trigger reactions. It also manages the communication with the reputors

collecting data about entities from them, generates reputation data inputs for the reputation scoring; and distributes the data to the RAAs. It might also act as central point of communication for the real reputee entities (e.g. persons) giving them legitimate control over their reputation and allowing entities to governance their reputations. RMAs can act as a single point of entry to the managed sets of local RAAs, which allows for efficient implementation of various mechanisms of making sure the RAAs functionalities are not abused (security mechanisms) and making sure privacy of entities, the reputation input data, and computed reputation objects is respected (privacy & information hiding mechanisms). For instance, an RMA can disclose abstracted aggregate reputation objects, such as trustworthiness levels of local entities, to authorized parties without revealing private reputation scores or local data about the entities.

Importantly to note, since RPAs can take different roles at the same time, the RPN can implement various distribution topologies from centralized to distributed reputation processing networks. For instance, a simple RPA can be a RAA for one or more entities and at the same time having the role of a RMA implementing the reputation object management. In more complex topologies RMAs might act as centralized nodes for networks of local RAA communities which can be accessed via communicating with the RMAs as a single point of entry to the local networks acting like a black box. Also mixed topologies are possible where RMAs reveal communication interface details about local RAAs to authorized consumers which then can start ad-hoc direct communications with the local RAAs. Another possibility is to interoperate and nest complete EPNs. This leads to different types of virtual organizations representing RPNs, where RMAs can communicate with each other, with their local RAAs, as well as with external reputation consumers. This gives high flexibility to implement different pragmatic coordination and negotiation protocols for the communication in EPNs, and supports the various organizational semiotics of how reputation object's information can be used in the context of organized activities and business domains.

4 Implementation

In this section we describe a concrete implementation of the conceptual architecture of a reputation processing network for the Semantic Web.

4.1 RuleResponder Agent Middleware

Rule Responder [1] is an enterprise service middleware which extends the Semantic Web towards a Pragmatic Web infrastructure for semantic agent networks. [12] Rule Responder utilizes industrial-strength enterprise service bus technology (Mule ESB) for distributed deployment of rule-based agent services and for conversations between agents. [6] The underlying ESB provides a highly scalable and efficient agent service broker and communication middleware which supports synchronous as well as asynchronous messaging using arbitrary transport protocols (more than 40 protocols such as SMTP, JMS, JDBC, TCP, HTTP, XMPP, SOAP etc.). For the communication the agents use (Reaction) RuleML [2] as a standardized platform-independent rule interchange format to interchange rules, events and information between agent services and

[1] Rule Responder: `http://www.responder.ruleml.org`
[2] Reaction RuleML: `http://reaction.ruleml.org`

other Semantic Web tools. RuleML [3] is an overarching rule standards family which comprises interchangeable rule languages including e.g. W3C SWRL (RuleML+OWL) and the new W3C RIF - RuleML [4] [5].

Reaction RuleML incorporates various kinds of production, action, reaction, and KR temporal/event/action logic rules as well as (complex) event/action messages into the native RuleML syntax. The general syntax of reaction rules is as follows:

```
<Rule style="active|messaging|reasoning" eval="strong|weak|defeasible|fuzzy">

      <oid>     <! object id -->                          </oid>
      <label>   <! meta data of the rule -->              </label>
      <scope><! scope of the rule e.g. a rule module --> </scope>
      <qualification> <! e.g. priorities, validity, fuzzy levels --></qualification>
      <quantification> <!- e.g. variable bindings-->      </quantification>
      <on>      <! event part -->                         </on>
      <if>      <! condition part -->                     </if>
      <then>    <! (logical) conclusion part -->          </then>
      <do>      <!--  action part -->                      </do>
      <after>   <! postcondition part after action, e.g.
                    to check effects -->                  </after>
</Rule>
```

Depending on which parts of this general rule syntax are used different types of reaction rules can be expressed, e.g. if-then (derivation rules), if-do (production rules), on-do (trigger rules), on-if-do (ECA rules). For communication between distributed rule-based (agent) systems Reaction RuleML provides a general message syntax:

```
<Message>
  <oid>          <!-- conversation ID-->                 </oid>
  <protocol>        <!-- used protocol -->               </protocol>
  <agent> <!-- sender/receiver agent/service -->        </agent>
  <directive> <!-- pragmatic primitive, i.e. context --> </directive>
  <content>         <!-- message payload -->             </content>
</Message>
```

Using messages agents can interchange events as well as complete rule bases (rule set modules), e.g. for remote parallel task processing. The protocol is used to defines the message passing and coordination protocol. The directive attribute corresponds to the pragmatic instruction, i.e. the pragmatic characterization of the message context broadly characterizing the meaning of the message. The typed logic approach of RuleML enables the integration of external type systems such as Semantic Web ontologies, XML vocabularies or Object Oriented class models (UML, Java, etc.). For instance, the following example defines a variable of type $ro : ReputationObject$ which is defined in our external reputation object ontology.

```
<var type="ro:ReputationObject">R</var>
```

Individuals of the reputation object can be bound to the typed rule variable and interchanged as part of the content of a message; sending the receiving agent not just the syntactical information about the payload data, but additionally the semantic information and a pragmatic context (directive) in which this data should be interpreted using the semantics. A standard nomenclature of pragmatic performatives is, e.g., the FIPA Agents Communication Language ACL [5], which can be used as external ontology

[3] RuleML: `http://ruleml.org`
[4] W3C RIF: `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`
[5] FIPA ACL: `http://www.fipa.org/repository/aclspecs.html`

in a similar fashion as the reputation object ontology. The semantic agent architecture in RuleResponder supports privacy and security implementations. In particular, local information used in the reputation authority agents becomes only accessible by authorized access via the public interfaces of the reputation management agents which act as an abstraction layer supporting security and information hiding. The Reaction RuleML IDL allows descriptions of the signatures of publicly accessibly rule functions together with their mode and type declarations and narrative human-oriented meta descriptions. External agents can access the local reputation processing networks only via these interfaces, which often only reveal abstracted information to authorized users, e.g. about public trustworthiness levels instead of the private entities' reputation details. Authorization of users in such hierarchical trust management agents might be implemented in combination with identity management, e.g. using FOAF+SSL [6] which is based on Public Key (PKI) standards. For instance, the following RuleML interface described the declaration of a public function *accessReputationObject* of a RMA which returns a ReputationObject ($mode = "-"$) and expects as an input ($mode = "+"$) a requesting agent of type *Agent*, defined in the FOAF ontology, and a composed security credential *ComposedCredential*, defined in the OWL-S credential ontology [7].

```
<Interface>
 <Expr>
   <Fun>accessReputationObject</Fun>
   <Var type="ro:ReputationObject" mode="-">RO</Var>
   <Var type="foaf:Person" mode="+">Person</Var>
   <Var type="owls:ComposedCredential" mode="+">ComposedCredential</Var>
</Expr>
</Interface>
```

While internal agents communicate using ESB as messaging middleware, external agents can communicate with the Reputation Processing Network via the configured endpoints of the ESB. Typically, this is a secure http endpoint for Internet connection, but might be also any of the other supported transport protocols such as JMS for communication from enterprise java beans (EJBs). User interfaces for human agents can be e.g. Web forms which translate into RuleML messages and post data to the RPN-ESB Internet endpoints (via http). For instance, an online Web user interface can be setup which allows issuing queries or submit reputation input data in a controlled natural language. RuleResponder supports arbitrary rule engines which provide a translator from the platform independent rule interchange format RuleML into the platform specific rule language of the engine such as Jess, Drools, OOjDrew, Prova, etc. We use Prova rule engine which provides the adequate expressiveness for the reputation processing agents.

4.2 Prova Rule Engine

Prova [8] is an enterprise strength highly expressive distributed Semantic Web logic programming (LP) rule engine. Prova follows the spirit and design of the W3C Semantic Web initiative and combines declarative rules, ontologies and inference with dynamic object-oriented programming and access to external data sources via query languages

[6] FOAF+SSL: `http://esw.w3.org/Foaf+ssl`

[7] OWL-S Security and Privacy: `http://www.ai.sri.com/daml/services/owl-s/security.html`

[8] Prova: `http://prova.ws`

such as SQL, SPARQL, and XQuery. One of the key advantages of Prova is its elegant separation of logic, data access, and computation as well as its tight integration of Java, Semantic Web technologies, and service-oriented computing and complex event processing technologies. It can be run in a plain Java environment or as a OSGI component enabling massive parallelization of Prova agent nodes in grid/cloud environments and deployment in e.g. ambient / smart components. From a logical perspective, Prova implements an extended (naf and neg), well-founded (linear 3-valued logic with goal memoization), defeasible (defeasible conflict handling with priorities), typed (external order sorted type systems such as Java or ontologies) semantics. [9]

The ability to embed Java calls in Prova makes the ISO Prolog-like rule programming style more suitable for integration of highly optimized Java computation workflows or (enterprise) java bean functions. The following simple rule creates a response sentence with the name and reputation score using Java string computation and displays it via to the Java system out. This is in particular useful for integrating domain-specific reputation algorithms implemented in Java for efficient computing reputation scores. Prova provides a rich library of built-ins include support for many query languages such as SQL, XQuery, SPARQL, File IO, so that external data sources can be dynamically accessed at runtime. For instance, the following rule uses a SPARQL query to access an RDF FOAF profile published on the web. The selected data is bound to variables and printed to the console.

```
exampleSPARQLQuery(URL,Type|X) :-
 QueryString = ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
                 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                 SELECT ?contributor ?url ?type
                 FROM <http://planetrdf.com/bloggers.rdf>
                 WHERE {
                         ?contributor foaf:name "Bob DuCharme" .
                         ?contributor foaf:weblog ?url .
                         ?contributor rdf:type ?type . } ',
 sparql_select(QueryString,url(URL),type(Type)|X),
 println([[url,URL],[type,Type]|X],",").
```

This dynamic integration of external data sources is useful to access the reputation (input) data upon which the reputation is computed. Prova supports external type systems such as e.g. Java class hierarchies or Semantic Web ontologies (RDFS, OWL) via its typed order-sorted logic. For instance, the typed RuleML variable $< vartype = "ro : ReputationObject" > R < /var >$ maps to the typed Prova variable $R : ro\_ReputationObject$. Prova supports reactive messaging via the built-ins

```
Send a message
sendMsg(XID,Protocol,Agent,Performative,[Predicate|Args]|Context)
Receive a message
rcvMsg(XID,Protocol,Agent,Performative,[Predicate|Args]|Context)
Receive multiple messages
rcvMult(XID,Protocol,Agent,Performative,[Predicate|Args]|Context)
```

This built-ins directly map to the message interchange format of RuleML and allow Prova agents to communicate with other agents via the RuleResponder middleware. The following rule snippet show how a query for a reputation object is send to an agent via the ESB and the received reputation object is then evaluated by the querying agent.

```
...
sendMsg(Sub_CID,esb,Agent,acl_query-ref, QueryRO),
rcvMsg(Sub_CID,esb,Agent,acl_inform-ref, ReceivedRO),
evaluateReputation(ReceivedRO),
...
```

Modularization of the knowledge base is another important expressive feature of Prova which is required to implement different agent roles in the same agent instance. It is possible to consult (load) distributed rule bases from local files, a Web address, or from incoming messages transporting a rule base.

```
:- eval(consult("reputation1.prova")).
:- eval(consult("http://ruleml.org/reputation2.prova")).
```

All rules in a rule base automatically have a special annotation $@src(FILENAME/URL/OID)$ with which they are managed as one module in the knowledge base. This label can be used for asserting or retracting complete modules from the knowledge base and for scoping queries / goals to apply only on the particular module. In the following example the subgoal $evaluateReputation$ applies on both modules $reputation1$ and $reputation2$.

```
reputation(X):-
  @src("reputation1.prova","http://ruleml.org/reputation2.prova")
  evaluateReputation(X).
```

This is quite similar to creating constructive views on data in a relational databases.

```
@label(rule1) r1(X):-q(X).
@label(rule2) r2(X):-q(X).
@label(rule3) r2(X):-q(X).
```

These rule labels can be, e.g., used for defining priorities in case of rule conflicts which are solved by defeasible reasoning. Meta annotations can be also used in literal guards, as the following example illustrates:

```
@author(dev22) r2(X):-q(X).
@author(dev32) r2(X):-s(X).
q(2).
s(-2).
trusted(dev22).
% Author dev22 is trusted but dev32 is not, so one solution is found: X1=2
p1(X):-
 @author(A)
 r2(X) [trusted(A)].
 :-solve(p1(X1)).
```

This example uses metadata annotations on rules for the head literals $r2/1$ and on the literal $r2(X)$ in the body of the rule for $p1(X)$. Since variable A in $@author(A)$ is initially free, it gets instantiated from the matching target rule(s). Once A is instantiated to the target rule's $@author$ annotation's value ($dev22$, for the first $r2$ rule), the body of the target rule is dynamically non-destructively modified to include all the literals in the guard $trusted(A)$ before the body start, after which the processing continues. Since $trusted(dev22)$ is true but $trusted(dev32)$ is not, only the first rule for predicate r2 is used and so one solution $X1 = 2$ is returned by $solve(p1(X1))$. After describing key features of Prova that are needed for implementing reputation processing agents, we now describe our semantic reputation ontology model which is used inside the agents for representing and interchanging reputation.

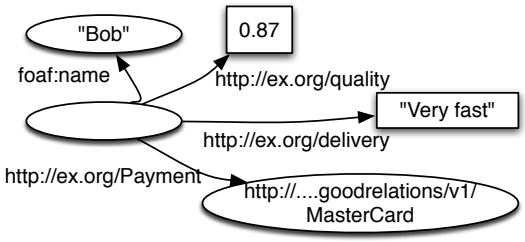

Fig. 2. A graph describing part of Bob's reputation

4.3 Reputation Object Model

A reputation statement usually describes a target, the topic of evaluation, and the value of this evaluation (i.e a judgment or a result of monitoring process). For instance, a rating of a seller in an e-market identified by `<foaf:Person rdf:nodeID="Bob">` then a simple description of his reputation can be viewed as declaring reputation statements. Bob's service-quality, delivery, and payment are identified by URIs as well as the literal values 0.87 and "very good". These statements corresponds to the RDF graph instance shown in figure 2 where the edges represent the `context`. This is a snippet of the reputation object instance that describes Bob's reputation:

```
RO={<Bob,quality,0.87>, <Bob,delivery,''very good'' >, <Bob,payment,gr:MasterCard>}
```

For the development of the Reputation Object Model Ontology, we used Protégé-OWL [9]. We developed a java library to facilitate the integration of the model within any system on the implementation layer. The implementation for reading, writing, and processing an RO along with the `selection` method (given a consumer priority list) was developed using Jena-API [10]. Using this library, a RO is attached to the rule-based agents and is exchanged and used in its logic for different kind of decisions, i.e. access policies, information dissemination, trust levels and the resulted, etc. An agent in Rule Responder also can manage reputation locally in its knowledge base, but can also communicate reputation objects to other agents. To illustrate, a simple example of an e-market seller's RO is showed in the following xml where reputation can be perceived as the evaluation of two criteria: `Review` and `DeliveryMethod`.

```
<gr:Reseller rdf:reference="http://www.example.org/John#">
 <ro:hasReputation >
 <ro:ReputationObject rdf:ID=''SellerRO1''>
 <ro:hasCriteria>
  <ro:Criterion rdf:resource=''http://purl.org/goodrelations/v1/DeliveryMethod''>
   <ro:hasReputationValue>standard</ro:hasReputationValue>
   <ro:collectedBy ro:CollectingAlgorithm=''#WebPortal''/>
  </ro:Criterion>
  <ro:Criterion>
    <review:Review>
      <review:rating>8</review:rating>
    </review:Review></ro:Criterion>
 </ro:ReputationObject></ro:hasReputation >
</gr:Reseller>
```

[9] Protege OWL: `http://protege.stanford.edu/overview/protege-owl.html`
[10] Jena framework: `http://jena.sourceforge.net/`

A customer's extract rating information from the structured objects because these two criteria are his priorities. The developed decision rule is:

```
buy(Name:gr_Reseller, Book:gr_Book):-
   reseller(Name:gr_Reseller, Book:gr_Book),
   hasDeliveryMethod("standard"),
   hasReviewRating("excellent").
```

5 Use Cases

The major benefit of using declarative rules, as opposed to standard procedural implementations e.g. directly in Java, is known to be in it its separation of concerns, which enables users to define the reputation processing logic independently from the underlying application and data models. Users can concentrate on defining the reputation processing network and the logic of the reputation processing agents - the responsibility to automatically interpret and execute these rules is delegated to a (Prova) rule engine. Distributed online reputation management systems can thus be much easier represented, managed and adapted while ensuring fine grained levels of modularity, security, privacy, and information abstraction and hiding. While this general benefits are well known properties of declarative rule programming, and have been demonstrated e.g. in other RuleResponder projects (e.g. [9, 11, 10, 6]), an important question in the evaluation of the quality of this work is the required expressiveness of the representation language for the declarative programming of distributed reputation processing networks consisting of rule-based agents. By means of a concrete use case implementation we demonstrate adequacy. A resource authorization agent, called *SocialActivities*, for computing the number of friends, implements a private rule $friend$ for deriving all friends of a person using a SPARQL query to access all friends from a local Foaf document in n3 format. A public rule $numberOfFriends$ computes the number of friends using this private rules. Other agents can query this public rule via a rcvMsg reaction rule waiting for incoming ACL $query-ref$ messages, which gives access only to authorized agents and only to all it's public rules. Privacy is ensured by not reveling persons' friends to other agents.

```
% private rule for collecting persons' friends
friend(Person, Friend) :-
 SparqlQuery = ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
                 PREFIX ex: <http://ns.example.org/#>
                 SELECT ?person ?friend
                 FROM <friends.n3>
                 WHERE {?person foaf:knows ?friend .} ',
 sparql_select(SparqlQuery,person(Person),friend(Friend)).
% public rule for computing number of friends
@public(numberOfFriends(?,?))
numberOfFriends(Person, Number) :-
  count(friend(Person,Friend),Number).
rcvMsg(CID,esb,Agent,acl_query-ref, Query) :-
 authorized(Agent), % check if requesting agent is authorized
 derive(Query) [public(Query)] % derive query if public guard is true
 sendMsg(CID,esb,Agent,acl_inform-ref,Query). % send back result
```

Another RAA, called *Driving*, computes driving ratings with respect to the driving skill level of a person. The shown rule *drivingSkill* is part of a set of the agent's rules, which derive personal driving ratings with respect the years of driving experience and years without accidents. The input data is coming from a local relational database which is queried by SQL in the private rule $yearsOfDrivingExperience$.

```
% public rule for computing number of friends
@public(drivingSkill(?,?))
drivingSkill(Person,Rating):-
   Rating = 8, % level is 8
   yearsOfDrivingExperience(Person,DrivingYears),
   DrivingYears > 5, % if driving experience is higher than 5 years
   yeasWithoutAccident(Person,AccidentFreeYears),
   AccidentFreeYears > 3. % and no accidents within 3 years
yearsOfDrivingExperience(Person,Years) :-
 ...,
 sql_select(DB,person,[years,Years]). % access data from local database
```

The following rule of a reputation management agent (RMA) allows external agents to ask for a person's reputation object. Both RAAs are queried by the RMA in two parallel running sub-conversations. The RMA then creates a (complex) reputation object for a person from the resulting two reputation criteria received from the RAAs. All details about the reputation computing algorithm and the reputation input data is not revealed to the external requesting agent, thus ensuring privacy of person's data.

```
rcvMsg(CID,esb,Agent,acl_query-ref, reputation(Person,ReputationObject) :-
  % query RAA Driving in new sub-conversation 1
  sendMsg(Sub-CID1,esb,"Driving",acl_query-ref,drivingSkill(Person,Rating)),
  % in parallel query RAA Rating in sub-cid 2
  sendMsg(Sub-CID2,esb,"SocialActivities",acl_query-ref,numberOfFriends(Person, Number)),
  % receive reputation criteria 1
  rcvMsg(Sub-CID1,esb,"Driving",acl_inform-ref, ReputationCriteria1),
  % receive reputation criteria 2
  rcvMsg(Sub-CID2,esb,"SocialActivities",acl_inform-ref, ReputationCriteria2),
  % create reputation object for person
  createReputationObject(ReputationObject,Person,ReputationCriteria1,ReputationCriteria2),
  % send back reputation object to requesting agent
  sendMsg(CID,esb,Agent, acl_inform-ref, reputation(Person, ReputationObject)).
```

This RMA can now be asked for the reputation of a person by sending it a query.

```
 sendMsg(CID,esb,"RMA1", acl_query-ref, reputation(foaf_person:Alice, RO:ro_ReputationObject))
```

The computed reputation `RO` of type `ro_ReputationObject` can be used to profile the reputation of a `foaf:Person` Alice by asserting it with a property `ReputationObject` describing her performance in the 2 computed criteria: `SocialActivities` and `Driving`.

```
<foaf:Person rdf:about="#Alice">
  <foaf:openid rdf:resource=http://alice.org''/>
  <ro:hasReputation >
  <ro:ReputationObject rdf:ID=''AliceRO1''>
    <ro:hasCriteria>
    <ro:Criterion rdf:resource=''#Driving''>
      <ro:hasReputationValue ro:value=8/>
      <ro:collectedBy ro:CollectingAlgorithm=''#drivingSkill''/>
    </ro:Criterion>
    <ro:Criterion rdf:resource=''#SocialActivities''>
      <ro:hasReputationValue ro:value=''active''/>
      < ro:collectedby >
        <ro:CollectingAlgorithm rdf:resource=''#numberOfFriends''/>
      </ro:collectedBy>
    </ro:Criterion>
  </ro:ReputationObject>
</ro:hasReputation >
</foaf:Person>
```

As shown by these examples from the implemented use case, we have demonstrated external data access, use of ontologies in decisioning and reaction rules, local autonomy of agents with support for data abstraction, information hiding and privacy, integration of external security system functionalities, parallel processing of reputation

within distributed communicating agent nodes, etc. Other properties which relate to the (worst case) complexity of the rule semantics and the robustness and scalability of the RuleResponder agent middleware have been evaluated in other industrial and research projects. The RuleResponder middleware based on an ESB has been proven to be highly scalable and robust and the underlying semantics of the Prova rule language is proven to be polynomial, with possible log-space reduction by defining constructive views using highly optimized query languages to access external data sources, which are then applied as fact base for the rule system's knowledge base. [9, 11] Moreover, massive high-performance parallelization of computation is possible by deploying Prova agents as OSGi components in computing cloud grids.

6 Conclusion

In this paper, we demonstrate a RuleResponder reputation management system based on a distributed reputation processing network architecture consisting of Pragmatic-Semantic Web agents which make use of an ontology based reputation object model and rules for declaratively programming the agent-based reputation processing and management logic. As a proof of concept we have implemented a Semantic Web reputation processing network based on ontologies and rules, in combination with highly efficient enterprise middleware. We also demonstrate adequacy in terms of expressiveness of our declarative knowledge representation approach by means of a real-world use case implementation with our approach. Deploying our reputation management systems on the Semantic Web helps opening the online reputation market to a context-aware competition between service providers, and customers who can select their providers according to their customized needs with respect to levels of reputation and trust.

Acknowledgements This work has been partially supported by the "InnoPorfile-Corporate Semantic Web" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

References

1. Rehab Alnemr, Stefan Koenig, T. Eymann, and C. Meinel. Enabling usage control through reputation objects: A discussion on e-commerce and the internet of services environments. In *in the special issue of Trust and Trust Management, Journal of Theoretical and Applied Electronic Commerce Research*, 2010.
2. Rehab Alnemr and Christoph Meinel. Getting more from reputation systems: A context-aware reputation framework based on trust centers and agent lists. *Computing in the Global Information Technology, International Multi-Conference*, 2008.
3. Rehab Alnemr, Adrian Paschke, and Christoph Meinel. Enabling reputation interoperability through semantic technologies. In *ACM International Conference on Semantic Systems*. ACM, 2010.
4. Rehab Alnemr, Matthias Quasthoff, and Christoph Meinel. *Taking Trust Management to the Next Level.* Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, 2009.

5. Harold Boley. RIF RuleML Rosetta Ring: Round-Tripping the Dlex Subset of Datalog RuleML and RIF-Core. In Guido Governatori, John Hall, and Adrian Paschke, editors, *RuleML*. Springer, 2009.
6. Lars Braubach, Alexander Pokahr, and Adrian Paschke.
7. Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *J. Web Sem.*, 2004.
8. Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004. ACM.
9. Adrian Paschke. *Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management*. IDEA Verlag, 2008.
10. Adrian Paschke. Rule responder hcls escience infrastructure. In *ICPW '08: Proceedings of the 3rd International Conference on the Pragmatic Web*, pages 59–67, New York, NY, USA, 2008. ACM.
11. Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated sla management. *Decis. Support Syst.*, 46(1):187–205, 2008.
12. Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web. In *2nd ACM Pragmatic Web Conference 2007*. ACM, 2007.