Reputation Objects for Interoperable Reputation Exchange: Implementation and Design Decisions

Rehab Alnemr
Hasso Plattner Institute
Potsdam University
Germany
rehab.alnemr@hpi.uni-potsdam.de

Christoph Meinel
Hasso Plattner Institute
Potsdam University
Germany
meinel@hpi.uni-potsdam.de

***Abstract*—Reputation systems aim to provide a mechanism for establishing trust for online interactions attempting to mimic their real-world counterparts. Reputation-based approaches depend on the users local experiences and feedback to create a soft measure for trust decision. They use various clues and past experience to decide on taking the risk of dealing with an entity. Reputation communities cannot exchange reputation or reputation information following an isolated-silos approach. The reasons for this varies from missing contexts in the reputation representations and heterogeneity in these representations, to technical reasons such as not being represented in an interoperable knowledge representation method. Developing interoperable reputation ontologies requires a technology that can provide means of *integrating data sources* and methods to *relate the data to its explicit semantics*. In this paper, we describe our design decisions in developing our reputation object (RO) ontology using semantic web technologies. The main motivation that leads these decision is to facilitate reputation information exchange or reputation interoperability along with model expressivity. Therefore, choosing the ontology language and specific APIs is explained in relation to this motivation.**

***Index Terms*—Reputation, Rating, Semantic Web Technologies, Collaboration, Trust Management**

I. Introduction

The population of users online is diverse and comprised of people from different backgrounds and cultures. Online systems have developed rating and reputation mechanisms for the sole purpose of establishing a measure of trust between these users and encouraging trustworthy behaviour. In current online communities, a user's reputation is viewed as the collection of ratings given by users in a community (global rating) about this user. However, how others see and perceive this reputation varies based on their views, tastes, opinions, backgrounds, needs, biases, and preferences, and on the context of interaction. The goal of reputation mechanisms is to help lower the risks of online interactions, thereby increasing the robustness and efficiency of internet-based applications. Reputation mechanisms are then used in finding experts, selecting compatible partners or service providers, and locating reliable recommendations and opinions. Because of this, these mechanisms also were extended to other environments such as Service Oriented Architecture (SOA) and cloud systems. This can only happen if system participants collaborate and share knowledge about their experience and ratings of the services provided. Reputation systems sometimes provide an incentive not only to collaborate but also to encourage constructive behavior. However, this open collaboration results in a challenge of what is being shared, how it is being shared, and the common versus the intended meaning of this information.

Virtual communities, especially online reputation communities, cannot exchange reputation or reputation information following an isolated-silos approach. Exchanging and transferring reputation from one community to the other increases trustworthy interactions, lowers the risk when dealing with new users, and eliminates the cold start problem (i.e. when registering in a new community, a user has to build up his reputation from scratch). By taking a closer look at the actual difficulties of reputation transfer, we can identify the crucial points of a working reputation transfer system and finally present a means of implementing such a framework. The first difficulty is that each reputation system has its own way of collecting, querying, calculating, and representing reputation. Most of these systems do not embed context related information within their reputation representation though most of reputation transactions are sensitive to the context of the reputation involved. A reputation without its context is meaningless and can sometimes be misleading. Second, there is heterogeneity in reputation representation and meaning. Third, even if it is possible to mine users' reputations from other domains (or to be presented by the user himself), the computation algorithm that computed these values remains as a black box. Thus, 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. Moreover, it is also important to reach with the collaborative communities (or services) a state that reflects the behaviour of their counterpart natural societies, hence making them more intuitive and easier to trust. Reflecting this goal, reputation transfer is a common phenomenon in our society and therefore should be facilitated in its computerized counterparts.

Our research is mainly concerned with creating a generic reputation representation model that is tractable enough to represent knowledge and enhance communicability between several domains. The first step was to understand the reputation concept from different perspectives and to use this

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250519

knowledge for providing trustworthy interactions between participants of virtual environments by studying and analysing reputation-related literature, as well as through online surveys, experiments, and user studies.[9][5][7] The second step was to determine a method to define such generic representation. Ontologies are used to enhance knowledge reuse by sharing common understanding of a domain.[14] They also facilitate interoperability and intelligent processing. Therefore, we have defined a generic ontology for representing an entity's reputation- called Reputation Object (RO) - given that our definition of reputation is the notion of profiling an entity's performance in several domains.[8] [6] [5] The third step was to develop our reputation object ontology using a technology which can provide means of integrating data sources and methods to relate data to its explicit semantics. Semantic web technologies were developed to provide a common data representation framework in order to facilitate the integration of heterogeneous sources. Therefore, it was our choice of technology for developing the ontology. Design decisions and implementation details of the model is the focus of this paper. The fourth step was to evaluate the generality of the ontology, by representing an entity's reputation in several use cases using our ontology which is discussed in [23][20][2].

The paper is organized as following: section II defines some relevant concepts to reputation systems and discusses problems in current systems. Section III explains our Reputation Object (RO) ontology and the proposed model. Section IV discusses the design decisions for the development of the model, its implementation and how to use it. In section V we conclude the paper and discuss future work.

II. Reputation Systems

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. In the literature, reputation is defined as an expectation about an entity's behavior based on information about or observations of its past behavior [1].

We distinguish between: reputation ontology, reputation system, model, or framework, reputation engine or mechanism, and reputation architecture. A *reputation ontology* describes the notion of reputation and the relations to the concepts that compose it, while a *reputation system, model, or framework* describes the collection, distribution, and aggregation of reputation information. A *reputation computation engine or mechanism* is one of the modules in a reputation system which shows how reputation value(s) are calculated. A *reputation architecture* is a set of protocols that determines how reputation values are communicated between the participants in a reputation system. A *reputation context* or criterion is defined as a characteristic, a property, or a measurement by which an entity is judged or evaluated in a certain context. Sometimes it is called *reputation attribute or quality attribute* and in general also it is called *reputation scope* which defines the general category in which a reputation was created i.e. for e-Market, for web services. [6]

There are extensive studies about reputation systems that discuss not only the current commercial ones but also proposed approaches from academia. For example, studies done by Jøsang in [13] and Sabater in [22], provide an exhaustive view of current reputation community. Commercial applications implementing trust and reputation mechanisms use relatively simple schemes than those proposed by research papers. However, online reputation systems are the biggest and most obvious examples of reputation systems. In [9] we categorized them by *characteristics* as: subject of rating, providers of the ratings (open to the public or restricted), business model (revenue derived from an associated online auction, advertising, or a public service) and relative reviews (whether users ratings are relative to the attributes of the rating). Another category is based on the *common features and properties* of the online communities as: e-market places, opinions, activity sharing, social and entertainment sites, news sites, the web and semantic web for anyone who publish anything in a decentralized way, and P2P networks where peer clients share opinions about other peers.

Each System use different kind of rating/raking to evaluate an entity's reputation. In online reputation communities, trust and reputation (or rating at this point) is represented numerically or graphically using bars and stars, karma, or in natural language (i.e. *good*, *bad*). The range of possible values for a trust level and the meaning of these values varies according to each system. [3] E-markets use reputation to manage trust between service providers and consumers. In this domain, users collaboration is essential i.e. If the users refrain from sharing their opinions and reviews of a product, the reputation system will seize to exist. Also, if the users share less, the reputation values will have insignificant meaning because there is not enough information to construct a credible and meaningful trust. The problem with these markets, however, is not sharing as *what is being shared*. In their book [11], Farmer and Price elaborated on the types of reputation system patterns that exist in current successful online systems.

Reputation mechanisms are used in finding experts, selecting compatible partners or service providers, and locating reliable recommendations and opinions. They are used also in other environments such as Service Oriented Architecture and cloud systems to facilitate service or service provider selection. Several approaches were presented to include QoS metrics (as reputation criteria or attributes) in the service descriptions and to help the consumer in distinguishing good services automatically. Examples of the work done in this area can be found in [16][18][17] and more. In [7], we explain the steps to construct a reputation system and elaborate on their types in [5][9][3].

A. Problems in the Current Representations and Interfaces

Vagueness of the rating is one of the problems in the online reputation systems. It is obvious when a user is rating a book: is the rating for the book itself (i.e. the user liked what he read) or the quality of the book (i.e. was new and good printing) or the service provided by Amazon for example (i.e. offering the book, price, delivery, payment method, etc.)? Rating should differentiate between *rating the service* and *rating the product*. Also it should take a different form if the domain is *"Multiple Products eMarket"* (i.e. when rating Amazon service) or *"One Provider eMarket"* (i.e. when the rating of the provider includes the quality of the product).

Such problem raised legal hassles. An interesting study about legal challenges that face online reputation systems was conducted in [10]. In this study, the authors explore legal cases against reputation systems as eBay (California, Grace vs.eBay) and Amazon (cases in UK and USA). Mostly, the main reason for these cases is rating ambiguity. Users misreport their ratings in a way that influences negatively the entity being rated and does not correspond to the rating attributes. For example, a seller of second hand books was given bad ratings because a book was not good or too long though the book was in a very good condition which is what matters for rating a used-books seller. Legally, systems like eBay holds no responsibility for users who are expressing their taste. What is important from the legal perspective is the distinction between "expressions of fact" and "opinion". In spite of the fact that eBay instituted limited assurance coverage- Standard Purchase Protection Program - the problem still exists and growing. What these systems need is specific rating attributes categorized and semantically defined. The less vague the rating, the fewer legal issues arise.

In general, reputation systems have also other issues some of which are:

- Excluding the context from the reputation value because most representation and exchange format has no embedded information about the context in which reputation was earned. Since context is not usually included in a reputation query, it is assumed that the implicit context is the domain of the reputation system (e.g. rate the seller for this purchase transaction) resulting in a too general query. In online markets, for instance, a consumer rates a seller generally for a trading/purchase transaction, leaving the details to be written in a natural language review.
- Difficulty in mapping between reputation values due to difference in perceptions
- Incorrect modeling and variance in calculations and interpretations because in spite of the wide variety of computation models, most of them do not reflect the real cognitive nature of reputation as they do not represent all the parameters that affect it.
- No portability or interoperability of reputation information because it is hard to exchange the knowledge when the semantics are not considered in the calculation or the representation of reputation. Reputation interoperability can solve problems such as the *cold start* problem.

These issues explain why it's hard to asses and exchange reputation especially between e-markets due to the difference in perception, calculation, interpretation, but most of all because the given reputation is *an overall one* that does not reflect the related contexts in which it is earned. These contexts can vary from the category it is earned (i.e. a selling transaction) to the quality aspects of one transaction (i.e. different quality criteria or attributes). [4]

III. Reputation Object Model

Most of the existing work on reputation systems focuses on improving the calculation of reputation values, preventing malicious actions, and the deployment into the business world where reputation is mostly represented in a singular value form. Our work focuses on how to represent reputation to reflect its real-world concept (i.e. non-general, context specific, and dynamic). The argument is that 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. Since reputation changes with time and is used within a context and every domain has its own information sources as well as its own requirements, the representation -not the calculation- of reputation should be unified between communities in order to facilitate knowledge exchange. In this ontology reputation is represented as a new form of reputation value: *Reputation Object (RO)*. This object holds information on the reputation of an entity in multiple contexts. The ontology's components are: a `ReputationObject hasCriteria` of one or multiple instances of class `Criterion` or `QualityAttribute` (for a service, the criterion describing service reputation is referred to as a quality attribute). The criterion is collected using a `CollectingAlgorithm` and `hasValue ReputationValue`. Each criterion instance has a `ReputationValue` (which includes the `currentValue`, its time stamp, and a simple list of its previous values called `historyList`) that in turn has the range of values defined in `PossibleValues`. It describes the data type that the criterion can have or a specific set of values (literals or resources URI) evaluating this criterion (e.g. a set of integers $\{1, 2, 3, 4\}$ describing 4 trust levels or a set of Strings $\{''good'','' bad'','' excellent''\}$ describing a user opinion). Each time a criterion is being evaluated (i.e. a new entry value for this criterion), a new `currentValue` is calculated using the `ComputationAlgorithm` which is the reputation computation function/engine used with this criterion such as *sum, avg, etc.*.

Since it is not always easy to identify intuitively what the highest reputation value is - among the defined possible value set -, the `PossibleValues` class has an `orderedList` that is ordered from the relatively highest reputation value to the lowest (e.g.$\{''excellent'','' good'','' bad''\}$). It also has the possibility to define a comparison and ordering function; `OrderFunction` to compare between values within each criterion and to be used by the reasoning engine. 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.

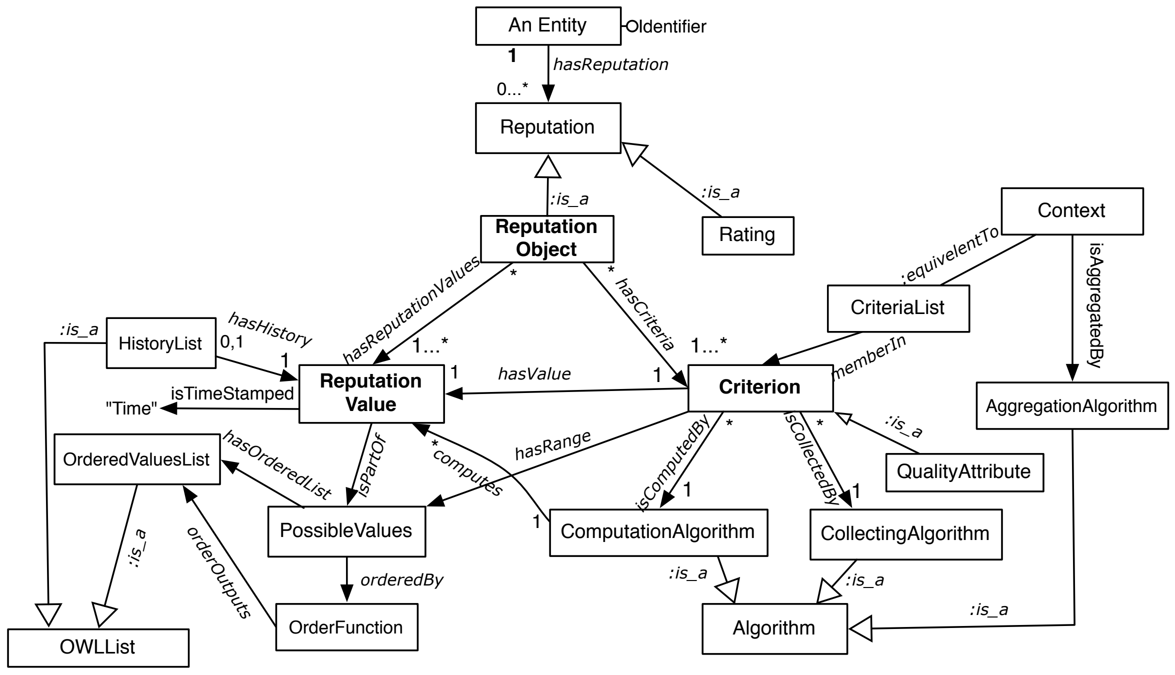

Fig. 1. The Reputation Object Ontology Model

The model is graphically represented in figure 1. A study and a comparison of current reputation ontologies is explained in [6] where we show why our ontology and model is more suitable for reputation interoperability.

IV. Model Development and Design Decision

In order to realize our model, a technology which provides a common data representation framework as well as a way to connect concepts with their definitions is needed. Semantic Web is developed with a main objective of facilitating data integration, enhancing information usage by connecting it to its definitions and context. Therefore, it was our choice of technology for implementing the model and to provide an integration possibility with the arising new semantic web services to be deployed on current semantic web infrastructure. In this section, we explain our design decisions, the languages and the APIs chosen for the implementation, and the implementation steps. The main motivation is to enable reputation interoperability within several domains and platforms. This motivation will lead the design decisions in every step.

A. Why Ontologies

In [24], an ontology is defined as: "A set of terms of interest in a particular information domain and the relationship among them". Ontologies describe domain-dependent as well as domain-independent knowledge. An Ontology defines common vocabulary for researchers who need to share information in a domain. In [19], the authors explain the reasons why one would want to develop an ontology. Some of them are:

- sharing common understanding of the structure of information,
- enabling information reuse,
- making domain assumptions explicit and clear,
- separating domain knowledge from operational knowledge,
- analysing the domain knowledge,
- and having the ability to integrate existing ontologies describing portions of the large domain.

Within the same community, ontologies enable mutual understanding among peers by providing precise semantics to concepts and relationships between these concepts. Therefore they are the most suitable knowledge representation of an interoperable object.

B. Choosing Ontology Language

Our goal is to have an ontology that can be of practical value and therefore the decision of which ontology language to use is critical. The choice was between using RDFS, and Web Ontology Languages (OWL-Lite, OWL-DL, or OWL-Full)[1]. Our requirements were to use a language that facilitates reasoning without being heavy in performance, that can describe complex relations between classes including the intersection and union relations, and that is compliant with both RDF and OWL elements. RDFS does not offer the same expressive capabilities of OWL, and therefore the choice was between the three families of OWL [12]. We decided to use OWL because it provides expressive modelling capabilities

[1]OWL-2: http://www.w3.org/TR/owl2-overview/

that help describe the restrictions and axioms of the model that should be incorporated in its description. OWL is a family of three languages: OWL-lite, OWL-DL, and OWL-full. From the three, OWL-DL was chosen because:

- OWL-Lite is not as expressive as OWL-DL and OWL-full. Low expressiveness prevents the developers from describing their ontology classes using complex axioms and restrictions. It does not support the use of quantifiers, and hence it is not suitable for developing a knowledge-base.
- OWL-Full is very expressive, yet in practice it does not support reasoning over the ontology.
- OWL-DL offers much expressiveness in describing the ontology and is also compatible with several reasoning engines.

A number of ontology editing tools are created to facilitate OWL ontology creation especially OWL-DL ontologies. One of the most popular editing tools is Protégé [2][3] which is the one used in developing our ontology.

C. Ontology Axioms and Knowledge-base

After choosing the language to use in describing our ontology, the model knowledge is represented in a knowledge-base. A knowledge-base is then used to produce new knowledge and to prove consistency of existing knowledge. An OWL ontology, from an abstract point of view, is a collection of axioms i.e. elementary pieces of knowledge or statements. They are *assertions* or logical statements about one or more objects in the class hierarchy. In a Description Logic (DL) knowledge-base, concepts and properties are separated from individuals in a *TBox* and an *ABox*. A*TBox* **T** is the terminology box which contains the axioms that make up the ontology. An *ABox* **A** is the assertion box where assertions are made about the terminologies, or in other words, facts are stated about individuals. The knowledge-base is described as:

$$KB = \langle T, A \rangle$$

where **T** is the union of the sets of concepts and properties along with their axioms, and **A** is the set of individuals in the domain with axioms that relate only to them. The development of such knowledge-base involves:

- The design of the TBox for the knowledge base. This is done by: classifying entities as classes and properties, declaring atomic and non-atomic concepts, creating the ontology taxonomy, partitioning the taxonomy, defining properties characteristics, and defining the axioms and restrictions to the TBox.
- The population of the ABox with individuals. This is done by enumerating and classifying each individual and relating them using properties.

Terminological axioms make statements about members of the TBox. The concepts and properties of the previously described reputation object model are added to the TBox by means of declaration and definition. Also, assertions about these concepts and properties are added in the form of *axioms*. For example, the restrictions and relations axioms for the concept*Reputation Object* and its cardinality are, respectively:

$$\begin{aligned} ReputationObject \sqsubseteq \exists (hasCriteria(Criterion \sqcap \\ (\exists hasReputationValue.ReputationValue)) \end{aligned} \tag{1}$$

$$ReputationObject \geqslant \mathbf{1}\, hasCriteria.Criterion \tag{2}$$

Every concept in the model is described by such axioms. We further construct the ABox by *populating* the ontology with *individuals creation* methods, either through the *Individual Plug-in* in Protégé or via the OWLFactory API implemented in our Java library (described in sub-section IV-E). In DL, populating a knowledge base ABox with individuals (i.e. adding individuals) is equivalent to adding concrete data to the knowledge contained in the knowledge-base as following:

$$\mathbf{ComputationAlgorithm}(Summation) \tag{3}$$

$$\mathbf{Criterion}(Delivery) \tag{4}$$

$$\mathbf{CollectingAlgorithm}(ReviewsEntry) \tag{5}$$

$$\mathbf{ReputationObject}(BobRep) \tag{6}$$

Relating these individuals to the relations or properties described in the TBox is relatively easy then:

$$hasCriteria(BobRep, Delivery) \tag{7}$$

$$computedBy(Delivery, Summation) \tag{8}$$

$$collectedBy(Delivery, ReviewsEntry) \tag{9}$$

Note that in OWL-DL or DL we cannot state, for example, relations for the delivery individual as `relates(Delivery,Summation,ReviewEntry)` for a property `relates`, since only binary relationships are allowed. That is why it is separated into two relations for the the individual *Delivery*. Figure 1 shows a graph of our ontology where the edges represent relations (or *properties*) between the classes and also their cardinality.

D. Choosing the Java API

As a part of developing a semantic web application, we developed a Java library to process the ontology and to give developers control over their domain logic implementation. An API is needed to communicate with the developed OWL ontology and to provide the same data model design of the ontology to the developers. Among the semantic web community, the most three famous Java APIs for dealing with ontologies and ontology graphs are: Jena [4], OWL-API [5], and Protégé-OWL API [6].

[2]Protégé: http://protege.stanford.edu/

[3]Protégé was developed in Stanford Center for Biomedical Informatics Research (BMIR) at Stanford University.

[4]Jena Ontology API:http://jena.sourceforge.net/ontology/

[5]The OWL API: http://owlapi.sourceforge.net/

[6]Protégé-OWL API: http://protege.stanford.edu/plugins/owl/api/

In order to develop our library, we tested some of the functionalities of each API and compared some of the features provided by them. All of these APIs are also integrated with reasoners to perform semantic queries on the ontology. OWL-API is the easiest to work with if one does not need to access the Protégé libraries. However, Jena is more mature and includes additional functionalities such as a database backend and transaction support. Nevertheless, as an RDF API, Jena represents OWL ontology in a lower-level of abstraction than the abstraction presented in the OWL functional syntax. On the other hand, since OWL-API does not support RDF, SPARQL[7], queries are not used. We chose Protege-OWL API over other APIs because:

- It supports both OWL and RDF. OWL-API bypasses RDF and provides services based on OWL only.
- Protege-OWL API can be used to edit OWL ontologies and to access description logic reasoners. As an extension of Protégé, the OWL Plugin benefits from the large user community and a library of reusable components.
- It supports direct creation and modification of SWRL rules from the API.
- It can be extended with a GUI using the integrated UI capabilities and libraries.
- It is the most complete of all of them.
- It includes most of the Jena properties including working with SPARQL queries.
- The mapping of OWL-Classes to Java interfaces reflects the intention of OWL.

E. Implementation

The goal of developing a semantic web application is to be able to use this application in an open infrastructure (such as the Semantic Web) that is based on formal models. Web services and software agents use these formal models - ontologies - as a way of understanding and communicating with each other. Developing a semantic web application consists of two stages: *formalizing domain knowledge using a standard ontology language*, and *coding the functionality and the logic of the desired system using conventional programming language*. According to [15], these two stages correspond to the two layers of any semantic web application where the *semantic web layer* publishes the ontologies and the interfaces on the web for discovery and service usage, and the *internal layer* controls the logic and the reasoning mechanisms and can be developed as a black box. Here, we describe our two stages of implementation that correlate with the previously mentioned semantic web application development.

For formalizing the domain knowledge, OWL-DL was chosen for the reasons mentioned in sub-section IV-B. Others who may decide to use this formalized ontology can define their own domain classes by the specialization of the default classes, and can add semantic restrictions to best define their characteristics (e.g. define a `WebServiceRO` as a subclass of `RO` class). For the logic and functionality layer, we use Java for implementing a library that can be used and integrated in other systems. The functionality is also exposed to software agents and web services through a service interface. The output is formally represented as an OWL object (i.e. the reputation object).

The application we developed access ontology objects (i.e. classes and properties) and the run-time objects (i.e. individuals). The structure of the package is as shown in figure 2. We developed it in such a structure to allow others the addition of their domain logic (i.e. adding new methods) and also to separate what has to be done from how it is done. For example, if a programmer wants to customize the methods used on his/her domain ontology, the new parameters and methods are added to the programmer defined part (e.g. interface `ROInterface`, class `RO`). The idea is that using Java (or via Java) we are able to fill the data model with data, *the data model* being the Reputation Object OWL ontology and *the data* being the individuals. This corresponds to the structure of the knowledge-base. The definition of the ontology itself will remain constant in the **TBox** and the change will be in the **ABox**. Changing the ontology itself is considered a major development cycle. Another functionality to the library is the access of the current individuals to query and update them. The package represents the internal data structure by which a programmer can perform actions on the ontology in an ”object-oriented” approach by means of parsing and serializing the ontology .

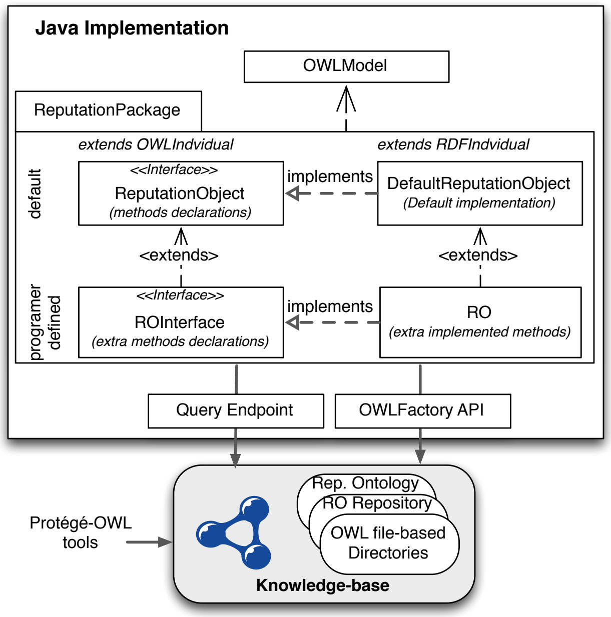

Fig. 2. Reputation Object Ontology Java Package

Parsing and Serializing the OWL ontology.: A parser is a method that takes an ontology serialization format such as RDF/XML, OWL/XML, or Turtle and converts it into an in-memory representation of the ontology that the serialization encodes. Renderers or serializers allow storage

[7]SPARQL Query Language: http://www.w3.org/TR/rdf-sparql-query/

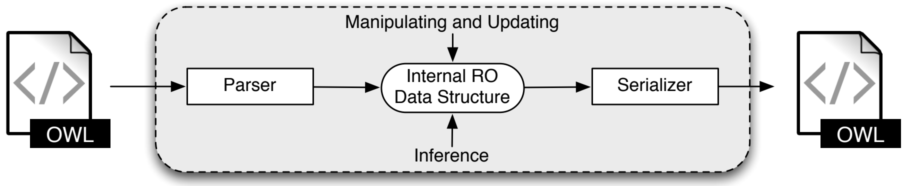

Fig. 3. Parsing and serializing

of ontologies using those serializations (figure 3). Protege-OWL API has two modes of loading and saving an ontology: OWL Files mode (class `JenaOWLModel`) and OWL Database mode (class `OWLDatabaseModel`). The static methods from the `ProtegeOWL` class can be used to load an existing OWL file from a stream or a URL. The calls will return a `JenaOWLModel`, and the save methods in the `JenaOWLModel` can be used to write the file back to disk. We use the file mode to load the ontology (called native repository), because it reduces the load and update time, as the following:

```
// create the corresponding Factory
   MyFactory fac = new MyFactory(owlModel);
String uri = "http://www.owl-ontologies.com/RepOntologyModel.
        owl";
owlModel= ProtegeOWL.createJenaOWLModelFromURI(uri);
```

, where the *uri* can be the actual URL of the ontology or a string with the local directory where the ontology file is stored. We then use the returned model to map the classes (from the core ontology file) and instances (from the instances file) to the internal data structure. After updating the ontology, the serializer writes it back to the file and saves it. We stored the ontology in several serialization formats. For example, the `OWLModelWriter` writes out a `TripleStore` in a specified OWLModel in RDF/XML format. However, this method takes lots of time (to create triples and a triplestore then serialize it). Another way is to serialize the model in an XMLAbbr. The method `owlModel.save()` can save an ontology in the format `FileUtils.langXMLAbbrev` as:

```
owlModel.save(new File(fileName).toURI(), FileUtils.langXMLAbbrev,
        errors);
```

or directly to the file stream:

```
owlModel.save((new File("RepOntologyModel.owl")).toURI() );
```

Once the ontology is loaded in an `OWLModel` it can be queried. The API also has methods to access resources (classes or properties) by their names (e.g. `OWLModel.getOWLObjectProperty(PropName)`). They were used to query specific properties and classes instances.

Usage and Integration: When using the library, a developer needs to extend both the interfaces and their default class implementation (figure 2) in the *programmer part* of the package. Any instances of the default classes has to be created via the `MyFactory` class. Here we show some examples on how to use the library. To create a new individual, for example, a new criterion:

```
//get a link to the factory
MyFactory fac = new MyFactory(owlModel);
//instantiate the individual and create its name
Criterion avail= (Criterion)fac.createCriterion("Availability");
ReputationValue v=(ReputationValue)fac.createReputationValue("80%")
        ;
ComputationAlgorithm comp=(ComputationAlgorithm)fac.
        createComputationAlgorithm("Actual-time-online/Lease-time"
        );
 // set the properties to the Criterion Individual
avail.addHasDescription("Measure the availability of the web service
        online");
avail.addHasValue(v);
avail.addIsComputedBy(comp);
```

To simplify, we used string values in the example. A developer who uses the library can extend the class methods to input data from external sources and hand them over to the *create individual* methods. One can add a new object property to class `Criterion` that says a *criterion* **can be** a *measurement*, for instance:

```
OWLObjectProperty canBeProperty = owlModel.
        createOWLObjectProperty("canBe");
canBeProperty.setRange(measurement);
canBeProperty.setDomain(criterion);
```

To add a new data property:

```
OWLDatatypeProperty property = owlModel.
        createOWLDatatypeProperty("name");
name.setRange(owlModel.getXSDstring());
```

Using Inference and Reasoning.: In order to make sure that the implemented classes confer with the ontology and its taxonomy, we instantiated a reasoner and queried it for the subclasses. This ensures that the hierarchy is correct and complete. Protégé-OWL API accesses only DIG and Pellet reasoners and has limited interface functionalities to their reasoning capabilities. Therefore, the reasoners in OWL-API were used instead. In order to use OWL-API reasoners, the `ProtegePelletOWLAPIReasoner` converts a Protégé-OWL model into an OWL-API model. The reasoner has methods to get inferred classes and to perform consistency checks on the ontology. We can also query the reasoner to get all the individuals of a certain class, to ask for a specific property value, and to get the class tree.

Output Data: Once a reputation object has been retrieved (in case of a query request) or updated (in case of a new entry), this object is returned as a response. If the operation being requested is sorting entities based on their reputation, a list of ordered ROs is returned as a response (`Collection<OWLNamedIndividual> ROs`). An example of the OWL file that represents an RO of a seller in an e-market is shown in listing 1. In this simplified example, a seller's reputation is described by the evaluation of two criteria: `Review` and `DeliveryMethod`. A seller or a business entity can be described by the vocabulary GoodRelations [8] which is an ontology for describing offerings and other aspects of e-commerce on the Web. The `WebPortal` specifies that the criterion `DeliveryMethod` has the reputation value *standard* if only one delivery method is available or has the value *several* otherwise. Review is a vocabulary for sharable reviews and simple ratings [9]. The final rating value - defined by the ontology - can only be a numeric value and expresses the reviewer's value judgement on the work.

```
<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>
```

Listing 1. Seller's RO

V. Conclusion

Reputation systems depend mainly on users sharing their experiences and opinions in a certain domain. The nature of the information being shared depends on the user perspective. Single format ratings tend to ignore the reasons and information behind the ratings. These ratings are used to construct the reputation of a service provider despite the lack of enough semantic information for the reasons behind them. For this and other reasons, reputation exchange between communities is not possible.

To enable such exchange of reputation, interoperable knowledge representation of reputation has to be used. We have developed an ontology -Reputation Object Ontology (RO)- to represent reputation in a format that is open, interoperable, and embedding reputation knowledge. Developing interoperable reputation ontologies requires a technology that can provide means of *integrating data sources* and methods to *relate the data to its explicit semantics*. We used semantic web technologies to develop the ontology and to describe the model. In this paper we discussed the steps of deciding on the design and implementation of the model putting *interoperability and explicit knowledge representation* as the motivation for each step. We started by describing the model briefly in section III, explaining in section why did we use ontologies for knowledge representation. The development was partitioned into two stages: (1) formalizing domain knowledge using a standard ontology language. In sub-sections IV-B, IV-C we explain why we chose OWL-DL as the ontology language and how the knowledge-base was constructed. (2) coding the functionality and the logic of the desired system using conventional programming language. In subsections IV-D, IV-E we explain why we chose Protege-OWL API over other Java APIs in the implementation and explain main parts of the implementation including: usage, integration, reasoning and querying the model. This all is done relating to the main goal of interoperability and expressive representation.

Any representation has its pros and cons, solving some problems and opening the possibility for others. As advised by [21], it is critical to find an equilibrium between usable representations and expressive ones. A highly expressive representation can add an unnecessary complexity to the reputation aggregation algorithm while a low expressive representation will not include sufficient information to produce meaningful reputation. By choosing to represent reputation in an OWL-ontology, we reach a degree of equilibrium. The representation itself is independent from the computation algorithms used, and the expressiveness of OWL-DL allows for usable decision-making processes. In future work, we plan to examine whether we can predict user buying decision based on our model.

References

[1] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Vol. 6*. IEEE Computer Society, 2000.

[2] Olga Streibel Rehab Alnemr. Trend-based and reputation-versed personalized news network. In *The 3rd Int. ACM Search and Mining User-generated Contents (SMUC) workshop associated with CIKM, UK.*, 2011.

[3] Rehab Alnemr, Justus Bross, and Christoph Meinel. Constructing a context-aware service-oriented reputation model using attention allocation points. *Proceedings of the IEEE International Conference on Service Computing*, 2009.

[4] Rehab Alnemr, Stefan König, T. Eymann, and C. Meinel. Enabling usage control through reputation objects: A discussion on e-commerce and the internet of services environments. In *Special issue of Trust and Trust Management, Journal of Theoretical and Applied Electronic Commerce Research*, 2010.

[5] Rehab Alnemr, Stefan König, Torsten Eymann, and C. Meinel. Enabling usage control through reputation objects: A discussion on e-commerce and the internet of services environments. *JTAER*, 5(2):59–76, 2010.

[6] Rehab Alnemr and Christoph Meinel. From reputation models and systems to reputation ontologies. In *Proceedings of the 5th IFIPTM, Springer IFIP, Copenhagen, Denmark*. Springer, 2011.

[7] Rehab Alnemr and Christoph Meinel. Why rating is not enough: A study on online reputation systems. In *The Collaborative Communities for Social Computing workshop (CCSocialComp), Florida, USA*, 2011.

[8]GoodRelation Vocabulary: http://purl.org/goodrelations/

[9]RDF Review Vocabulary: http://hyperdata.org/xmlns/rev/hReview

[8] Rehab Alnemr, Adrian Paschke, and Christoph Meinel. Enabling reputation interoperability through semantic technologies. In *International Conference on Semantic Systems*. ACM, 2010.

[9] 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.

[10] J. Chandler, K. El-Khatib, M. Benyoucef, G. Bochmann, and C Adams. Legal challenges of online reputation systems. In *In L. K. R. Song, Chapter in Trust in E-Services: Technologies, Practices and Challenges*, pages 84–111. Hershy: Idea Group Publishing, 2007.

[11] Randy Farmer and Bryce Glass. *Building Web Reputation Systems*. Yahoo Press, March 2010.

[12] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building OWL ontologies using Protégé 4 and co-ode tools, 2009.

[13] Audun Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, pages 618–644, 2007.

[14] Vipul Kashyap, Christoph Bussler, and Matthew Moran. *The Semantic Web, Semantics for Data and Services on the Web*. Springer-Verlag, 2008.

[15] Holger Knublauch. Ontology-driven software development in the context of the semantic web: An example scenario with protege/owl. *1st International Workshop on the Model-Driven Semantic Web (MDSW2004)*, 2004.

[16] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference*, pages 66–73. ACM, 2004.

[17] E. Michael Maximilien. Multiagent system for dynamic web services selection. In *In Proceedings of 1st Workshop on Service-Oriented Computing and Agent-Based Engineering*, page 25?29, 2005.

[18] E. Michael Maximilien and Munindar P. Singh. An ontology for web service ratings and reputations, 2003.

[19] Deborah L. McGuinness. Ontologies come of age. in spinning the semantic web: Bringing the world wide web to its full potential. In *MIT Press*, 2003.

[20] Adrian Paschke, Rehab Alnemr, and C. Meinel. The rule responder distributed reputation management system for the semantic web. In *RuleML-2010 Challenge, Washington DC, USA*. ACM, 2010.

[21] Jordi Sabater and Mario Paolucci. Representation and aggregation of social evaluations in computational trust and reputation models. In *International Journal of Approximate Reasoning*, 2007.

[22] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24:33–60, September 2005.

[23] Maxim Schnjakin, Rehab Alnemr, and Christoph Meinel. A security and high-availability layer for cloud storage. In *The 2nd International Workshop on Cloud Information System Engineering (Springer CISE 2010)*, 2010.

[24] Matthew Moran Vipul Kashyap and Christoph Bussler. The semantic web, semantics for data and services on the web. In *Springer-Verlag*, 2008.