2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11

Towards Context-aware Service-oriented Semantic Reputation Framework

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

Maxim Schnjakin
Hasso Plattner Institute
Potsdam University
Germany
maxim.schnjakin@hpi.uni-potsdam.de

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

***Abstract*—Reputation has been explored in diverse disciplines such as artificial intelligence, electronic commerce, peer-to-peer network, and multi-agent systems. Recently it has been a vital component for ensuring trust in web services and service oriented architecture domains. In this paper, we show details about our context-aware reputation framework. The framework is based on our semantic representation model for reputation called Reputation Object (RO) model. We discuss the advantages and propositions to construct such framework, its components, and how it is implemented. The importance of developing and using such generic reputation framework is highlighted within the emergence of the Semantic Web and service oriented architecture.**

I. Introduction

Reputation systems are used as a way of establishing trust between unrelated parties, especially if enforcement methods like institutional policies are not implemented. They may help lower the risks of online interactions and increasing the robustness and efficiency of internet-based applications. 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. Online reputation systems are the biggest and most obvious examples of these systems. It can be categorized based on the common features and properties of the web communities such as e-markets, activity sharing, social and entertainment, news sites, P2P systems and systems build upon the Semantic Web.[6] In service-oriented systems, quality attributes and ratings given by other services or service consumers are used to represent service reputation.

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 [24], provide an exhaustive view of what is out there in the reputation community. Commercial applications implementing trust and reputation mechanisms use relatively simpler schemes than those proposed by research papers. Most of the current reputation systems are built for a single context and only for the domain of their creation and also for a specific project or by a private company using proprietary schemas. Each has its own method to query, store, aggregate, infer, interpret and represent reputation information which limits the possibility of using these reputation information in other domains. In open environments, trust depends on reputation and how reputation is represented. In most reputation systems, the context of a reputation value is not embedded within the given reputation information. Reputation changes with time and is used within a context. Context-aware trust is complex but crucial for deriving meaningful trust. 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. This can be possible by embedding reputation information in its representation, especially information about the contexts involved in its creation, which is a feasible goal to achieve using ontologies. In information systems, ontologies promote and facilitate interoperability as well as intelligent processing. They are developed to enhance knowledge reuse by sharing a common understanding of a domain that can be communicated between people, and heterogeneous and widely spread application systems.

In this paper, we present some of the problems of current reputation systems, analyze why context-aware systems are needed, and show how we build a context-aware reputation framework that is based on our work of a generic reputation ontology. The framework adapts service oriented architecture (SOA) approaches. SOA focuses on encapsulating different functionality into services to be reused and composed with other services when needed. Each service represented in the paper has a functionality that enables context-awareness in our framework. We also discuss the implementation of the framework and the reasons for using both semantic web and SOA technologies.

The paper is organized as follows: section II briefly discusses reputation systems and definition of relevant terminologies, section II-A and III-A explicate the problems in current systems and the advantages of context-aware frameworks. In section III we explain the proposed framework that is based on our reputation object ontology, the

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.47

architecture, and the implementation. We finalize the paper with conclusion and future work.

II. Reputation Models, Frameworks, and Systems

In the literature, reputation is defined as an expectation about an entity's behavior based on information about or observations of its past behavior [2]. In the business world, Balmer [7] defines two characteristics for *corporate reputation*: it evolves through time and is based on what the organization has done and how it has behaved. Reputation definition evolves through the introduction of more complicated models. For example, Paolucci and Conte in [19] introduce a cognitive theory that clearly distinguishes between two concepts- *image* and *reputation*. The computation algorithm was defined in [22] as the Repage model. One of the major efforts that has been done to define a set of terms about reputation concepts can be seen in the european project eRep [1]. Based on the model described by [19] for multi-agent systems, the project explains in details the elements of social evaluations, the process of transmitting them, and agent decisions regarding reputation. The model distinguishs between two social artifacts that pertain to the evaluation of a target: *image* and *reputation*. *Image* is the the output of the evaluation process of other agents and assumed to be true by the agent who holds it. *Reputation* is the voice the agent is spreading which is not necessarily the truth.

Since there is a vast literature showing reputation systems from different perspectives (i.e. [15], [13], threats in [1], mechanisms in [10]), here we will briefly point out several definitions to distinguish between some terminologies. Resnick [20] defines a **reputation system** as: *"a system that collects, distributes, and aggregates feedback about participants' past behavior"*. It must have three properties to operate: long lived entities with an expectation for future interactions, ratings that can be captured and distributed, and past interactions' ratings used to guide the decision making. Conte and Paolucci [19] defines reputation-based systems as: *"a spontaneous and implicit norm-based system for social control"*. Jøsang [13] defines **reputation architecture** as a network architecture which determines how ratings and reputation scores are communicated between participants in a reputation system. He identifies two main types for the networks: *centralized* and *distributed* architectures. Both architectures have a reputation computation engine, but each has a different communication protocol (i.e. centralized protocol for centralized architecture). A **reputation computation engine** computes the reputation value based on plenty of factors (according to the model used) such as one's own experience, others referrals, a combination of both, etc. Some of the used algorithms or computation functions are: summation, average [21], bayesian systems [14][27], discrete trust models [2][8], belief models [12][28], fuzzy models [23][26], cognitive as [11], etc.. A **reputation model** describes all of the reputation statements (i.e. a source rating a target), events, and processes for a particular context. They were developed using different approaches and different semantics. A **reputation context** is the relevant category for a specific reputation. A reputation system should describe therefore:

- Computation functions/mechanisms i.e. how to calculate reputation?
- Communication model i.e. how to collect and disseminate reputation?
- Participants i.e. who use and/or is affected by reputation?
- Resources i.e. what is the information used to calculate reputation?
- Representation model i.e. how to represent, view, or visualize reputation?
- Storage i.e. where and how reputation is stored?
- Functionalities and applications i.e. what are the benefits of using reputation in the domain of its creation

It should also describe how these components are integrated into a given system. Here, 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.

Another definition that is vital to our work is reputation context. 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 e.g. for e-Market, for web services.

A. Issues in The Current Systems

Most of the existing reputation-based systems lack the connection between general reputation and the context of the given reputation. One can trust a physician to treat him but not to handle his financial transaction. Although reputation systems are used as the basis of trust in several online communities, there are some issues that arise in these systems due to the design and implementation of the current reputation models. These systems were designed as closed domains, where each one has its own data entry, enquiry,

[1] eRep Project: http://megatron.iiia.csic.es/eRep/?q=node/93

representation and interaction styles, interpretation, computation of reputation values, limited information sources, etc.. [4] This results in [5]:

- 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 is hard to assess and exchange reputation especially between e-markets due to the difference in perception, calculation, and 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. Context-aware Reputation Framework

In this section we describe our context-aware framework and its advantages. The framework is generic and can be integrated in several domains. Some of the services in this domain will remain the same -i.e. same implementation- while others will be more specialized to the domain i.e. specialization to the proposed interfaces. The framework adapts service oriented architecture (SOA) principles. SOAs have more open and collaborative environment that enable a timely manner response to user needs and business demands. The benefit of employing a service oriented approach in our framework is that in such collaborative environment knowledge extraction is easier, and each required functionality can be separated in a loosely coupled service; to be reused and composed with other services when needed.

The presented framework is based on a data model that represents reputation as an open, interoperable and comprehensible format. The *reputation object (RO)* model component is described here followed by the rest of the components explained in the architecture subsection.

A. Why Context-aware Reputation Frameworks

Current reputation frameworks either do not fall into the category of context-aware systems or have limited capabilities to implement context-awareness. Closed domains, by definition, do not attempt to semantically identify and define all the contexts related to their system processes since these are internal processes used only within the domain. In section II-A we discussed the problems in current systems. Here we elaborate on the advantages of developing context-aware reputation frameworks. They are:

- *Relevancy.* Most reputation systems are built for single context and only for the domain of their creation. A reputation context is the relevant category in which a specific reputation is earned. A context can be an objective (measurable) or a subjective one (non-measured but can have an approximated evaluation). Examples for reputation contexts are: a person's reputation in driving, a factory's reputation of producing quality goods, a service's reputation in its response-time, a post's reputation as likable to users, or a movies reputation in its storyline. A general reputation is not a form of judgment nor expectation for the target's performance in a specific topic. For example, one can be an excellent driver yet a bad runner. Also, one would not hire an excellent doctor to handle one's finances. This means that relevancy enhances the decision making process. Therefore, associating reputation with its context - and constructing context-aware reputation framework- is important for the relevant meaning of such reputation.
- *Portability/Interoperability.* Abstract or general reputation that is not connected to its context and is created for a particular domain, can not be assessed in another domain. Reputation values created by current systems simply can not be transmitted or used in another domain. This is due to the semantic heterogeneity of reputation models that makes it difficult to compare and evaluate them. Also, simplistic representation results in a lack of expressiveness and sparse reputation details, thus preventing reputation interoperability. Reputation interoperability is the possibility of transferring reputation information from one domain to the other and still can be understandable and comprehensible. Portable reputation requires awareness of the domain of its creation i.e. its context. The advantages of a portable reputation are: collaboration between online markets that can lead to building a knowledge base about multiple participants in the market, increased amount of information about a system participant that leads to more accurate expectation of his performance, and it also helps avoiding the *cold start* problem where, for example, a participant of one community does not need

to start building his reputation from scratch every time he signs up for a new community.

- *Trustability and credibility.* Reputation is subjective and since the human perception of trust varies from trust levels to the kind of trustworthiness (i.e. trust in a relationship, trust in an organization, trust in a person, or trust in credentials), it is critical to expose more information on the evaluation of the context of a given reputation to be able to make fair judgment. In other words, more information can lead to more trust. Moreover, context-aware systems will provide the means to know how each reputation value is computed, therefore increasing the possibility of reputation portability.
- *Providing customized selection.* In domains such as e-markets, internet of services, or the cloud, the problem of selecting a certain service provider based on a set of reputation criterion (that changes for each consumer) often arises. A context-aware framework can provide easier and customized way to select a seller according to what is most suitable to the user's requirements (i.e. the user needs faster delivery service and he/she dose not care about the price).
- *Eliminating Legal hassles.* Studies [9] on infractions and law suits against some e-markets because of reputation matters showed that the absence of rating context was the cause of these law suits. 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. To avoid such legal hassle, the rating must be less vague. Especially given the fact that laws and regulation vary by jurisdiction, which creates a challenge for online businesses and providers of online reputation systems operating internationally. For example, if the rating means provided is to give from one to five stars to a service provider, it should be clear what the consumer is rating. Is it the quality of the product, the service of the provider, customer service or something else entirely?

B. Reputation Object Model

Most of the existing work on reputation systems focus 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. This work [5] 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. The implementation of the ontology is described in section III-D. This ontology was used to represent an entity's reputation in several domains such as multi-agent based system (in [16], as the reputation of an agent and a way for decision making), for usage control in Internet-of-services (IoS) [4], and as an underlying ontology for a SOA reputation service in [3] that was later used in [25] for cloud service provider selection. It was designed mainly to facilitate reputation information exchange or reputation interoperability in any domain. Using this ontology, a dynamic ontology alignment is possible between two entities since reputation information (that helps in the alignment specially during runtime) are embedded in the reputation object. However, since the ontology focuses on representation, it does not address factors like transformation functions (mapping functions), ergo the separation into a standalone mapping service (discussed in III-C) which can

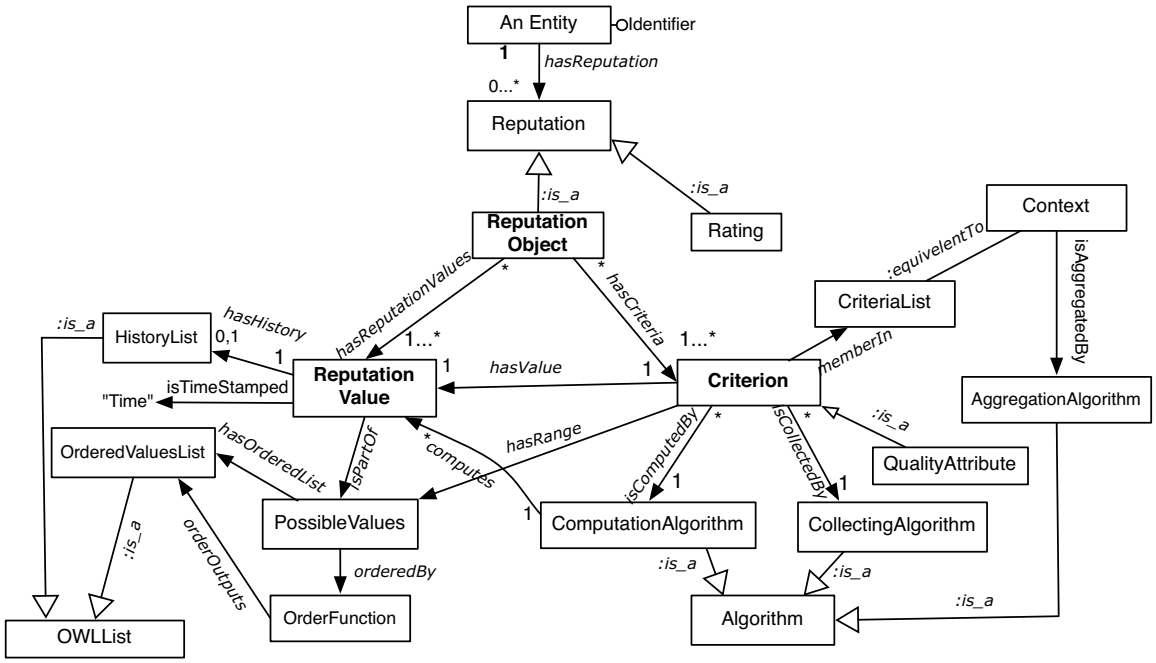

Figure 1. Reputation Object Ontology [5]

use mapping functions presented in [18] to enhance the use of the ontology.

Using reputation objects and taking advantage of the existing quality processes has the advantages of interoperable reputation and reusability of the embedded information. It also enforce context-awareness in the system where an entity's reputation is not generalized anymore. Each value in the reputation object can be related to a certain context. Therefore giving the benefit of easier storage and maintenance that leads to easier - and cleaner- information extraction process. It also facilitates context matching between two different platforms. The possibility of sharing reputation information increases if there are common reputation contexts- or quality attributes- that can be matched and then transferred to other domains (rather than transferring a single value that has no meaning to the destination system).

C. Architecture

The framework is a plug-in architecture to allow easy adaptation and generic use. For the purpose of reuse, interoperability and generic integration, it is loosely-coupled from the used identity systems. A system can adapt to the framework using its chosen identity model without forcing it to use a specific one. It needs to query the identity system for the user identity information. The framework is not dependent on a particular reputation computation function (or computation algorithm), the computation is done in a separate reputation computation engine or service. This allows each system to use its own computation function. The only needed requirement is either exposing the reputation engine as a service -to communicate with the rest of the framework components via messaging and service calls- or to pass a description file of the reputation engine (i.e. OWL ontology describing what is the computation function in use and a URI for the description). Figure 2 shows our general architecture along with possible communication points. The architecture is structured around our previously mentioned *reputation object semantic model* and it shows how to adapt to it.

The architecture is based on the principle of separation of concerns following a service oriented approach. The service-oriented architecture focuses on encapsulating different functionality into services to be reused and composed with other services when needed. In our work we focus on both the functionality and data layers where we provide generic services implementations or prototypes used in the integration of this framework with any domain. The presentation layer is dependent on the domain or the system, ergo, less generic.

*1) **Information Sources and Data Layer**:* Describes sources used to collect reputation information about an entity. These information are usually domain dependent because the perception of what constitutes an entity's reputation differs from one domain to another. For each domain, there are a number of specific criteria that reputation is affected by. In our architecture, each system has a set of these criteria described and stored in the *Domain-Specific Criteria* data storage. It can be configured by the system

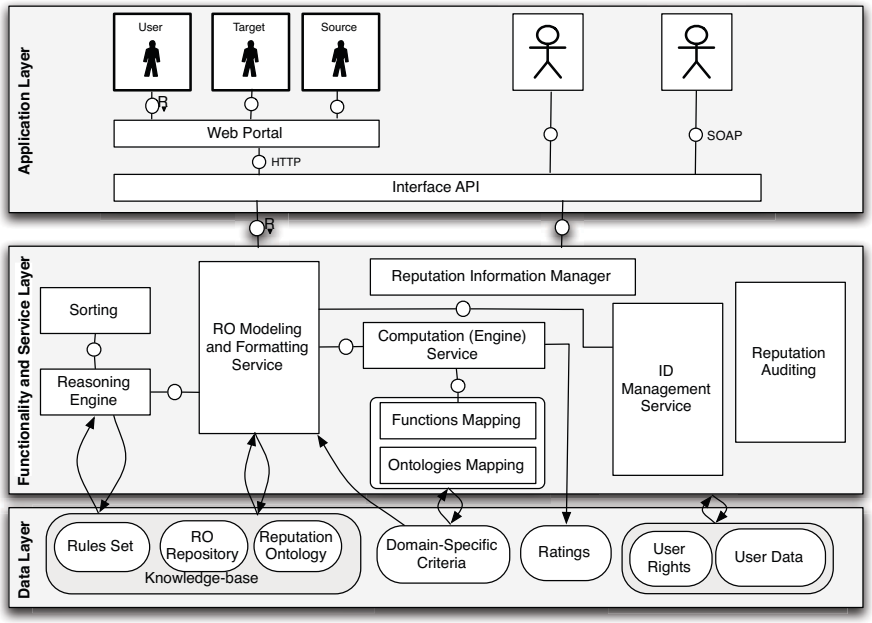

Figure 2. Framework

administrator or reused from a different system in a similar domain (e.g. service reputation is dependent on non-functional properties of the service such as its availability, response time, quality of service, consumers ratings). The reputation object modeling service communicates with this data storage to determine what are the criteria that will formulate a reputation object for this domain. Examples of different information sources are feedbacks from service consumers, direct experiences between software agents, ratings of an entity, service level agreements feedbacks between services, logs, system analysis, entity's performance, non-functional attributes of a service, etc. Each is described as a context or a criterion by which a reputation is obtained and evaluated. From observations such as these examples, a common abstraction form to an information source can be derived. This abstraction is a requirement to properly describe an information source- to ensure context awareness in the framework- and is used to construct the reputation object. The abstraction form consists of the following:

- Context name.
- Context description.
- The algorithm used to collect this data.
- A function that can be used in aggregating or computing multiple values of this context (e.g. average, sum, min, max, etc.). Also known as the reputation function or the reputation computation algorithm.
- Comparison function for each context (e.g. higher than, lower than, max of, etc.) that can be used to compare between two values (i.e. which reputation value is considered higher or better than the other).

A system administrator or a domain expert decides what are the criteria that construct the reputation of system participants and stores it in the *Domain-Specific Criteria* repository. A change in this repository affects how a reputation object is constructed (reflected by a *read* access by the *ROService*). The data layer consists of the used knowledge-base and information used by the identity system. Our *Knowledge-base* consists of three components:

(a) the *reputation object ontology*[2] described in section III-B and stored in an OWL repository,
(b) the user *reputation object repository* which contains ontology individuals (instances) i.e. objects that hold user reputation information, and
(c) the *rules set* used by the reasoning engine for decision making and is configured by domain expert to have direct control over their executing system.

*2) **Functionality and Service Layer**:* This layer has the services that are needed in a context-aware reputation framework. We have implemented some of these services (described in III-D). However, we also describe here the functionalities of the remaining external services (i.e. can be implemented by others) and integrated in the framework (which should be easy since the framework design adapts a SOA approach):

- *Reputation Object Modeling Service:* exposed using both SOAP and RESTful endpoints (results in accessibility to all types of clients), and takes as input a message that defines the operation to be executed, i.e.

[2]Reputation Object Ontology Vocabulary: http://purl.org/ralnemr/ro#

createRO, updateRO, queryRO. For example, when the operation is *updateRO* the service takes as input the entity's reputation criteria names - String or URIs - and values (i.e. `criteriaList`, `valuesList`), and returns as output an updated version of an entity's reputation object. In this operation, the service first requests the current version of the RO from the RO repository and the updated one is later stored in the same repository (read-write access to the repository). If the current RO does not have one or more of the criteria in the passed criteria-list (`criteriaList`), the service invokes the *ontology mapping service* to match the criteria on the `criteriaList` with the ones in the current RO. If there is no match found (unsuccessful mapping), the RO service simply adds the new criterion to the ones in the RO. In this case, the RO service requests from the *domain-specific repository* for each new criterion the following information: lists of value-types, order function, collecting algorithm, and computation algorithm (semantic description to the criterion). When the service needs to update a RO, it passes to the appropriate *computation service* a triple of (`newValue,oldValue,historyList`) which in return responds with updated criterion reputation score. The `historyList` is passed because some computation functions use the transaction history as a factor in their computations. In the end, the service stores the updated version in the *reputation object repository* (see algorithm 1).

- *Reasoning Engine:* which is used by the framework for reasoning and decision making. In section III-D we show the reasoning engine we used for our experiments.
- *Sorting Service:* One of the main goals of our framework is to enable customized service selection. This means that a consumer should be able to select a service based on his/her own preferred criterion. The sorting service takes as input two or more reputation objects i.e. `ROList` and a list of criteria that represents a consumer's *priorities list*. The list has sorted priorities or preferences (sorted reputation criteria) for the comparison between the objects. For example, the list has a service consumer's preferences for service providers based on their *service availability* first then *prices*, or has a customer's preferences for sellers based on their *prices* first and second by their *delivery time*. Since for each criterion an *order function* is defined, the comparison operator changes based on the criterion. The ontology has, for each criterion, its order function specified.
- *Computation Engine/Service:* deploying the reputation function or algorithm that is used to calculate a new reputation value whenever a new transaction takes place. The service can be a collection of computation functions that are used for each reputation criterion or a composed service that combines several functionalities (that corresponds to each computation algorithm). For example, the algorithm that calculates *service availability* is different from the one that calculates *service quality*. Service composition is - either by hard coding the composition or by using tools such as BPEL[3] or WS-Notification[4]. In our implementation, we used computation functions provided by eRep project and basic functions described in [18].
- *Mapping Services:* the *functions mapping* part is where two reputation computation functions are mapped. They are also called transformation functions. For example, in [18] the authors show a set of transformation functions between *Boolean, Real, Discrete Set and Probability Distribution* representations since they are the most common ones. They implemented an API interface of a set of common operations whose inputs and outputs are elements of the ontology, and must be implemented for each particular model. The *ontology mapping service* aligns two concepts -reputation criteria- to determine the correspondences between

Algorithm 1 General Description of the RO Modeling Service functionality

Require: List of criteria c, list of values v, entity i

if $operationType \leftarrow'' create''$ **then**
  $create(c, v, i)$
  $setRO(RO_i)$ in RO repository
else
  $RO_i \leftarrow getFromRepositry(i)$ *//requests RO from the RO repository*
  for each criteria c_j in criteria list c **do**
    if c_j is not in RO_i *//if the criterion is not in the RO* **then**
      if $mapService(j) \leftarrow NULL$ *//unsuccessful mapping* **then**
        $getCriterionDetails(c_j)$ from Domain-Specific repository
        $addCriterion(RO_i, c_j)$
      else
        $j \leftarrow mapService(c_j)$
      end if
    end if
    $newValue \leftarrow compFunction_j(oldVal, newVal, historyList)$
    $update(RO_i, newValue)$
  end for
end if
return RO_i

[3]BPEL OASIS Specification:http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm

[4]WS-Notification OASIS Specification http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

them or their semantic similarities. In multi-agents systems, it facilitates the interaction between agents within different communities by first discovering the semantic similarities between their related context/criterion, then it uses a *translator* to map the ontology between one agent to the other. The final mapping is wrapped in a representation that can be understood or interpreted by each agent. If the resulted common ontology is used only during run-time (for a specific interaction), it is a process known as *dynamic ontology alignment*. We used this external service implementation to realize the framework, adapting the core approach of service oriented architectures (i.e. service reuse).

As discussed earlier the identity system is independent from the reputation system, and the component is shown in figure 2 for illustration purposes. The *Reputation Auditing* service is not yet implemented -though defined in [4]- and is discussed in our future research. The information needed by the aforementioned services is collected and provided by the *reputation information manager service* which is system dependent (usually a set of services that handles knowledge extraction and knowledge feedback processes). We have implemented a reputation manager service prototype to pass the information in our experiments from pre-defined data sets.

3) ***Communication Protocols*** *:* The framework can be used for both centralized and decentralized reputation systems. A reputation authority can be the composition of the reputation services and the identity system in use. In this case, for *centralized systems*, the reputation request is a query about a specific individual for a specific criterion (or several criteria), the response from the reputation authority is the RO of the entity in question. For *decentralized system*, it's more like gossip where agents or representatives for the participants exchange reputation information. Each has its own repository of reputation objects of agents they were previously in contact with and each has its own reputation computation engine and a list of reputation criteria (albeit related).

On another level of communication, our framework uses the principles of service oriented architecture (SOA) [5] to implement each functionality as a service. W3C organization[6] defines a web service as:"*..a software application identified by a URI whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.*". This means that communication between services can be carried out using SOAP messages or REST operations. The advantage of adapting a SOA approach is access to shared resources, interoperability of data exchange, services are loosely coupled and can be reused, enabling service discovery, and service compositions. In our framework, services are exposed using both SOAP messages and RESTful operations for information exchange.

Querying the knowledge-base for the RO of a user is defined as *reputation queries* and is done via the *reputation object modeling service* (by specifying in the input message the operation name *queryRO*) and passing the query. In the service implementation, reputation queries are available through SPARQL [7] endpoints or the encapsulated reasoner API of `OWLModel` (explained in III-D).

D. Implementation Details

In order to realize such framework, a technology which provides 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, we used semantic technologies to develop our framework. The reputation object ontology itself was developed in OWL [8] using Protégé-OWL tools [9]. We decided to use OWL because it provides an expressive modeling capabilities that help describe restrictions and axioms of the model that should be incorporated in its description such as: how is the reputation value obtained, can a criterion refer to another concept (criterion matching) in other platforms, how to aggregate values of this concept if a new evaluation value is entered, can a set of criterion be aggregated in one context, how many reputation objects can an entity have, can the reputation object be extended, cardinality, inverse relationships, influencing factors, etc.. In such case, ontologies are used to provide such level of expressiveness.

Others who may decide to use this formalized ontology can define their own domain classes by the specialization of the default classes and add semantic restrictions to best define their characteristics (e.g. define a `WebServiceRO` as a subclass of `RO` class). We show examples of such specialization in internet of services domain in [4] and cloud service provider selection in [25]. 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 (i.e. SOAP endpoints). The output is formally represented as an OWL object (i.e. the reputation object). Part of the structure of the Java package is as shown in figure 3. We developed it in such 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.

[5]OASIS Reference Model for Service Oriented Architecture: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[6]World Wide Web Consortium: http://www.w3.org/

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

[8]OWL: http://www.w3.org/TR/owl-ref/

[9]Protege OWL: http://protege.stanford.edu/overview/protege-owl.html

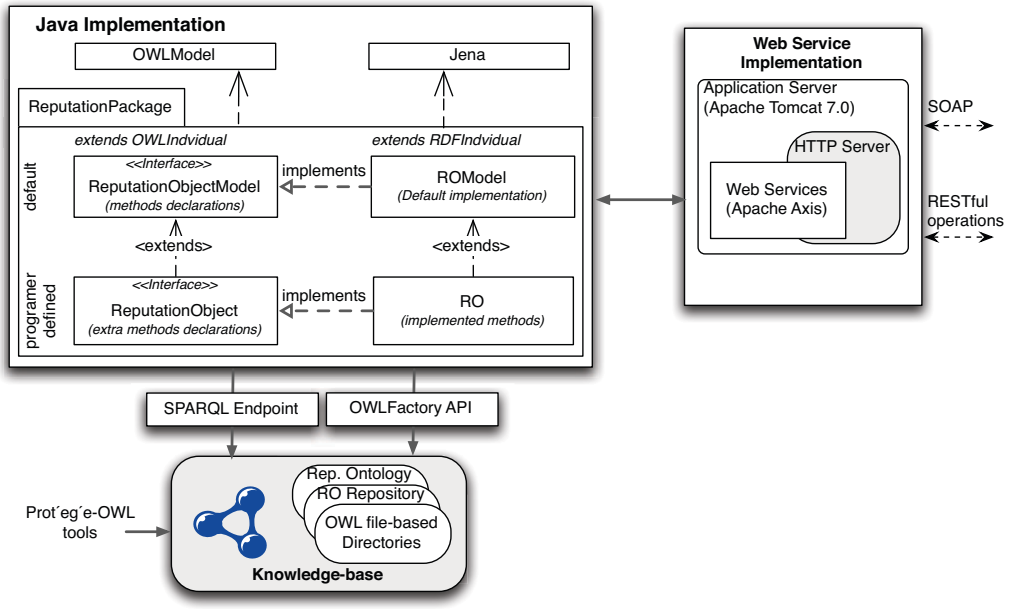

Figure 3. Technologies used in our framework

Implementation for reading, writing, and processing an RO along with the `selection` method (given a consumer priority list for the *sorting* service) was developed also in Java using `OWLModel`[10] library and Jena-API [11] which facilitates the integration of the model in any system on the implementation layer. In order to enforce the use of semantic storage of reputation objects, the only way to create a new object is through an OWL factory which interacts with the `OWLModel` which in return is used to create, modify, and query the ontological resources i.e. the reputation ontology repository. The `OWLModel` API has an encapsulated API for reasoning and provides access also to Pellet Reasoner. We used SPARQL for simpler queries over RDF data and the `OWLModel` reasoner for more complicated queries on the OWL ontology repository. This is implemented as part of *the RO modeling* service.

Part of our experiments was setup in an agent-based environment. In these experiments, the *rules set* is communicating with a rule-based service (agent) in the Rule Responder system[12] which allows the deployment of distributed rule inference services running a local rule engine such as Prova or Drools on an enterprise service bus [17]. The rule services, which can act as multi-agents, can communicate with each other using Reaction RuleML[13] as a standard rule interchange format. The separation of the rule engine from the *ROService* is intentional for integration purposes. Some might want to hard code their reasoning rules and decide not to use a fully developed middleware like Rule-Responder.

For the implementation of the web services, the used application server is Apache Tomcat 7.0[14] and Apache Axis [15](a SOAP-aware servlet) for deploying the services.

Output data: Once a reputation object is being retrieved (in case of a query request) or being 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. An example of the OWL file that represents a 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 [16] 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 [17]. The final rating value -defined by the ontology- can only be a numeric value and expresses the reviewer's value judgement on the work.

[10]OWLModel: http://protege-owl.sourceforge.net/javadoc/edu/stanford/smi/protegex/owl/model/OWLModel.html
[11]Jena framework: http://jena.sourceforge.net/
[12]Rule-Responder: http://responder.ruleml.org
[13]RuleML Initiative: http://ruleml.org/
[14]Apache Tomcat: http://tomcat.apache.org/index.html
[15]Apache Axis 2: http://axis.apache.org/axis/
[16]GoodRelation Vocabulary: http://purl.org/goodrelations/
[17]RDF Review Vocabulary: http://hyperdata.org/xmlns/rev/hReview

Listing 1. Seller's RO

```
<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>
```

IV. Discussion

In order to examine the current reputation systems and to discover problems that hinder the development of context-aware reputation systems, we are conducting several user studies. The first study focuses on how users perceive reputation and whether that perception is translated in current representations. The second study was an experiment on the online rating experience to see how users rate and compare between their ratings and what do they actually want to convey. During these study, we come to the conclusion that there is no such thing as a general reputation. Reputation is meaningful only in a context and a single value (or graphical representation) does not communicate what the users actually mean in their ratings. This led us to propose a framework that ensures context awarness not only to online communities but also to software systems in general.

In previous sections, we presented why context-aware systems are needed and proposed a framework that not only enables the construction of context-aware reputation systems but also ensures that it is easy to integrate it with other systems. Integration is possible via communication with the presented services, using the presented Java library, and by adapting our generic ontology for the representation of reputation information. The implementation conforms to two principles: the use of semantic technologies (i.e. using OWL for the reputation object ontology implementation) and the design of a service oriented architecture (i.e. separating functionality into services and using SOA technologies for their implementation). Semantic web technologies facilitate data integration and enhance information usage by connecting it to its definitions and context. Reputation ontologies create a common understanding of the notion as well as facilitate reputation exchange. A SOA design has many aspects and benefits. It provides a flexible infrastructure to allow independently developed software components to communicate in a seamless manner.

V. Conclusion and Future work

Reputation is used to establish trust between unrelated parties in web communities. Reputation systems are build for single context and only for the domain of their creation. Context-aware systems enables interoperability, establish trust through connecting information to its context, and allow customized service selection. In order to represent reputation information, ontologies are used to ensure that knowledge of the context related to this information is embedded within the representation. Ontologies are essential to make reputation systems meaningful in general contexts, and to support exchange of reputation information.

In this paper, we proposed a context-aware reputation system that is based on our reputation object ontology and designed to adapt to service oriented approaches. Advantages to context-aware reputation systems and the components that are needed to construct these systems were also illustrated in the paper.

In our future work, we plan to continue enhancing the framework and the implementation of the reputation auditing service. *Reputation auditing* process is an evaluation process that collects and monitors information regarding reputation quality then offer solutions and adjustments.

References

[1] *Reputation-based Systems: a security analysis*, 2007.

[2] 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.

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

[6] 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, pp. 796-816, 2009.

[7] John M. T. Balmer and Stephen A. Greyser, editors. *Revealing the corporation*. Routledge, London [u.a.], 2003.

[8] Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A formal model for trust in dynamic networks. In *In proc. of international conference on software engineering and formal methods*, pages 54–63, 2003.

[9] 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.

[10] Chrysanthos Dellarocas. Reputation mechanisms. 2005.

[11] Babak Esfandiari and Sanjay Chandrasekharan. On how agents make friends: Mechanisms for trust acquisition, 2001.

[12] Audun Jøsang. A logic for uncertain probabilities. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 9:279–311, June 2001.

[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] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation for e-businesses. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, HICSS '02. IEEE Computer Society, 2002.

[15] Lik Mui, Ari Halberstadt, and Mojdeh Mohtashemi. Evaluating reputation in multi-agents systems. In *Trust, Reputation, and Security: Theories and Practice*, Lecture Notes in Computer Science, pages 183–194. Springer, 2003.

[16] 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.

[17] 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.

[18] I. Pinyol, J. Sabater-Mir, and G. Cuni. How to talk about reputation using a common ontology: From definition to implementation. In *Proceedings of the Ninth Workshop on Trust in Agent Societies, Hawaii, USA*, pages 90–101, 2007.

[19] Conte R. and Paolucci M. *Reputation in Artificial Societies. Social Beliefs for Social Order*. 2002.

[20] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, pages 45–48, 2000.

[21] Paul Resnick and Richard Zeckhauser. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. In *The Economics of the Internet and E-Commerce*, pages 127–157. 2002.

[22] J. Sabater, M. Paolucci, and R. Conte. REPAGE: REPutation and imAGE among limited autonomous partners. *Journal of Artificial Societies and Social Simulation*, 2006.

[23] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA, 2001. ACM.

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

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

[26] Sandip Sen and Neelima Sajja. Robustness of reputation-based trust: boolean case. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, AAMAS '02, pages 288–293. ACM, 2002.

[27] Andrew Whitby, Audun Jøsang, and Jadwiga Indulska. Filtering out unfair ratings in bayesian reputation systems. 2004.

[28] Bin Yu and Munindar P. Singh. An evidential model of distributed reputation management. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, AAMAS '02, pages 294–301, USA, 2002. ACM.