Reputation Description and Interpretation

by

Ryan Chard

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2012

Abstract

Reputation is an opinion held by others about a particular person, group, organisation, or resource. As a tool, reputation can be used to forecast the reliability of others based on their previous actions, moreover, in some domains it can even be used to estimate trustworthiness. Due to the large scale of virtual communities it is impossible to maintain a meaningful relationship with every member. Reputation systems are designed explicitly to manufacture trust within a virtual community by recording and sharing information regarding past interactions. Reputation systems are becoming increasingly popular and widespread, with the information generated varying considerably between domains. Currently, no formal method to exchange reputation information exists. However, the OpenRep framework, currently under development, is designed to federate reputation information, enabling the transparent exchange of information between reputation systems. This thesis presents a reputation description and interpretation system, designed as a foundation for the OpenRep framework. The description and interpretation system focuses on enabling the consistent and reliable expression and interpretation of reputation information across heterogeneous reputation systems. The description and interpretation system includes a strongly typed language, a verification system to validate usage of the language, and a XML based exchange protocol. In addition to these contributions, three case studies are presented as a means of generating requirements for the description and interpretation system, and evaluating the use of the proposed system in a federated reputation environment. The case studies include an electronic auction, virtual community and social network based relationship management service.

Acknowledgments

I would like to thank my supervisor, Dr. Kris Bubendorfer, for his invaluable guidance and leadership, I am sincerely appreciative of all the time and direction. I am deeply grateful for the assistance my co-supervisor, Dr. Ian Welch, has provided. The opinions and feedback given by Dr. Lindsey Groves were extremely helpful and identified areas that required further work. I would also like to thank my examiners, Prof. John Hine and Dr. Ian Warren, for their in depth feedback, leading to a higher quality thesis being produced. Finally, I extremely thankful of the time and support given by my family and friends.

Contents

Chapter 1

Introduction

1.1 Introduction

Reputation is a tool used to estimate the reliability of others by associating historic actions to identities. The goal of reputation is to offer an indication of trustworthiness and forecast the behaviour of an entity by evaluating their actions. The basis for utilising reputation assumes that past actions are indicative of future behaviour. A reputation to act honestly provides confidence for others, suggesting an entity will perform honourably during potential interactions.

Reputation systems bootstrap interactions between entities in electronic communities, establishing a level of trust between unfamiliar users. A reputation system functions by collecting, aggregating and distributing the records of information regarding the actions of participants. Others within a community can then review this information and base decisions on the perceived reliability of an entity. Due to the population and vastness of many online communities, the ability to build long term, meaningful relationships with every member is not practical. The information supplied by reputation systems emulates the characteristics of having a previous relationship with other members. In essence, this approach mimics the informal ”word-of-mouth” reputation aggregation mechanisms inherent in social interactions.

Reputation systems have become prominent in virtual communities and are essential to the success of many online environments. The variety of domains that utilise reputation systems range from network routing applications [46] and spam filters [56], through to electronic auctions [61], peer-to-peer systems [67] and product recommender applications [42]. The diversity of domains is reflected in the variations of reputation systems and the information they generate.

In almost all examples of reputation systems, reputation is established, stored and utilised only within the domain. This creates reputation silos, where there is very little interaction or interoperability between systems.

Exchanging reputation information between systems can be used to increase the available perspective of entities within an open reputation environment. However, due to the proprietary nature of reputation systems, the information that is collected and distributed is often highly individualised to the domain in which it is generated. At a technical level there is currently no mechanism available to standardise reputation information and no protocols defined to transfer reputation between domains. This thesis addresses these two limitations with the aim of contributing to the development of a federated reputation system that can share reputation information between domains. OpenRep [27] is a distributed reputation framework designed to support the transparent exchange of global reputation information between reputation systems. Figure 1.1 depicts an environment in which OpenRep enables collaboration between distinct reputation systems. OpenRep creates a single framework for reputation systems to exchange information regarding entities within an open environment. This thesis presents the design and development of a reputation description and interpretation system that underpins the OpenRep framework.

The proposed reputation description and interpretation system includes the functionality to express diverse reputation information in a consistent

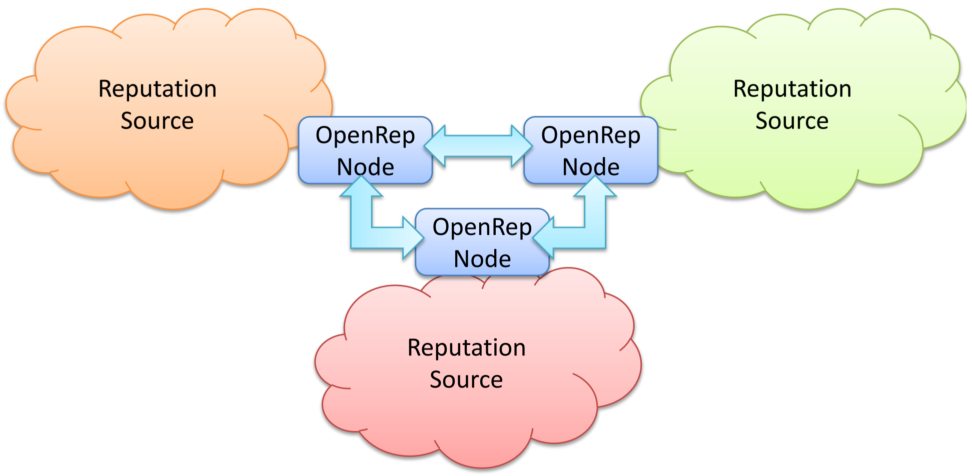


Figure 1.1: An open environment containing three unique reputation sources. The sources are able to communicate and exchange reputation information through their exposed OpenRep nodes.

manner. It defines a novel reputation language and accompanying verification architecture to ensure its correct use. Finally a unique reputation exchange protocol is defined to support the standardised transportation of reputation between systems. In order to demonstrate the functionality of being able to describe an entities reputation within one environment and transplant it, meaningfully, within another, case studies have been employed. Three diverse case studies have been defined and implemented, these case studies are used to drive the development of the reputation description and interpretation systems and are referred to throughout the thesis.

The remainder of this chapter defines common terminology used throughout the thesis, highlights the contributions made and describes the structure of this thesis.

1.2 Terminology

There are multiple components involved in the description of a reputation environment and its functionality. The following section briefly describes the assumptions and vocabulary used in this thesis when discussing reputation settings and the entities interacting with one another.

Within a reputation based environment, it is possible that many distinct reputation systems may coexist. Reputation systems are said to act as either *providers*, the sources of reputation information, or *consumers*, the recipients of information. In some cases a reputation system may be both a provider and a consumer. Each system is expected to maintain a personalised store of entities and reputation information relating to them. The reputation information is either collected through the monitoring of local interactions, or is gathered from other providers within the environment.

In real world scenarios, entities often utilise pseudonyms in different domains, for example individuals commonly use different aliases for different online services. In a global reputation environment an entity's global identity must first be established in order to aggregate reputation information from different domains. The focus of this thesis is the exchange and interpretation of diverse reputation information from different domains, rather than the ability to globally identify individuals. For this reason, establishing the global identity of an individual is considered outside the scope of this work, common identities across domains are assumed throughout the thesis.

The term *entity* can have many connotations within a reputation environment. Typically an entity refers to either an individual user, a group or an organisation. When describing the exchange of information, entities are described as agents and targets. Agents are those entities that request information, while a target is considered the entity for which reputation information is being investigated.

1.3 Contributions

This thesis makes multiple contributions to the development of a federated reputation system. The expression and interpretation of reputation information forms the basis of a framework to exchange information between reputation systems. The description of contextual information, differences between human and machine entities within reputation environments and standards regarding interoperability between reputation systems are also investigated. Specifically the main contributions of this thesis are:

- The design and prototype implementation of a reputation description and interpretation system. The description and interpretation system provides many features to facilitate the exchange of reputation information. The key elements of the system are:
  - The development of a reputation language, used to express information and provide a common form of communication between reputation sources. The language supports the description of reputation values and operations.
  - The definition of reputation types, capable of describing all identified forms of reputation information. The typing system includes the definition of domain specific types, such as the classification of interactions between users of a social network.
  - The design and implementation of a verification system for the reputation language. The verification system includes a static type enforcement system to ensure correct use of the language.
  - An exchange protocol, defined in XML Schema Definition schemas, standardises the encoding and transfer of reputation information. The protocol regulates the communication between reputation sources and embodies multiple features to reduce the overhead of communication.

  - The consistent expression of contextual reputation information allows reputation sources to encode information without loss of detail. Through the formalisation of the environment with regard to context, information can be interpreted with meaning.

- An investigation regarding the plausibility of integrating human and machine based reputation systems. Seven phases of a reputation system have been identified from the examination of a diverse set of human and machine based reputation systems. In addition, investigation of reputation systems with respect to the actions available to users under each of the identified phases provides a unique insight into the complexities associated with the integration of human and machine generated information.

- The design and implementation of a reputation environment, containing three uniquely constructed case studies. Each of the case studies represents a potential reputation source within an open environment. The case studies provide a practical basis to drive the development and evaluation of the description and interpretation system. The case studies include an electronic auction, virtual community and social network based relationship management service. The services are each independently implemented as a Jersey Web service utilising a Derby database, a Drupal client with a built in forum module and a Facebook application hosted in the Google App Engine.

1.4 Thesis Organisation

The thesis is organised as follows. Chapter 2 presents background information regarding the OpenRep architecture. Chapter 3 presents an overview of related work in which the role of reputation within different environments is discussed. Advancements in identity management and existing reputation exchange frameworks are also considered. Chapter 4 provides the design strategy of the research, explaining the motivation and role of case studies within the work before examining the requirements and design goals of the system. Questions raised in the design strategy are addressed in Chapter 5. This chapter describes the process of establishing solutions to these concerns, such as the definition of standards to facilitate interoperability between reputation systems, whether machines and human based systems can be integrated, methods of expressing contextual information and the selection of an appropriate language generation tool. Chapter 6 discusses the design and implementation of the Reputation Language. The design and implementation of the Verification System is given in Chapter 7. Chapter 8 investigates the process of implementing each of the case studies as functional services, capable of simulating a reputation environment. An evaluation of the work is given in Chapter 9, discussing and assessing the key aspects presented in the thesis. Finally, a conclusion to the work is given in Chapter 10.

Chapter 2

OpenRep Background

OpenRep is a distributed reputation system designed to federate disparate reputation systems in an open environment. OpenRep is designed to be easily incorporated into existing systems, such that users and providers can expose a node to capture, aggregate and consume reputation information. The OpenRep model consists of a set of standard, service based, interfaces and protocols that must be implemented by participating entities in order to join the OpenRep ecosystem. OpenRep leverages OpenID to authenticate users and associate diverse reputation information to a particular user. Through this extension of the OpenID architecture, reputation information can be consistently discovered, aggregated and shared in an open environment. This chapter presents the OpenRep architecture defined by Hendrix et al. [27]. The chapter presents background information relevant to the integration of the new components presented in this thesis into the OpenRep architecture.

2.1 OpenRep Architecture

The OpenRep architecture is designed to provide the capability to collect and distribute reputation information for participating entities. OpenRep is a distributed service-based framework that uses OpenID to authenticate users within their domain and provides an infrastructure to facilitate the exchange of reputation information between domains. Reputation information in OpenRep is distributed throughout the entities that compose the network.

The OpenRep architecture extends the OpenID system to integrate reputation information with a users digital address, this identifier uniquely represents an entities global identity. OpenRep maintains a globally consistent resource for each identity, providing a profile of reputation for each entity and node within the environment. Enabling reputation to be stored and retrieved from a predictable location establishes a foundation for the aggregation of information from multiple platforms. Associating reputation information with a globally resolvable identity, rather than localised pseudonyms, allows diverse reputation sources to contribute information about a specific entity and thereby provide a thorough history of an entities actions.

Although a single identifier is used to represent an entity, reputation information is replicated and distributed throughout the environment using a peer-to-peer network. The decentralised OpenRep architecture utilises a Distributed Hash Table (DHT) overlay network, such as Chord [69] or Tapestry [81], to establish supernodes. A supernode acts as a centralised authoritative node for a subset of nodes, and a peer to other supernodes, this model supports efficient routing of information through the overlay network [80]. In the OpenRep model each provider node is represented as a supernode. Supernodes are responsible for replicating reputation profiles throughout the network. Due to *node churn*, where nodes join and leave the network unexpectedly, information must be thoroughly distributed in order to improve reliability and fault tolerance. The DHT provides the ability to resolve the location of an entities reputation information at different providers. Figure 2.1 demonstrates the process of locating an entities reputation profile using hash functions on the OpenID address.

Applying a supernode architecture also increases the reliability of the

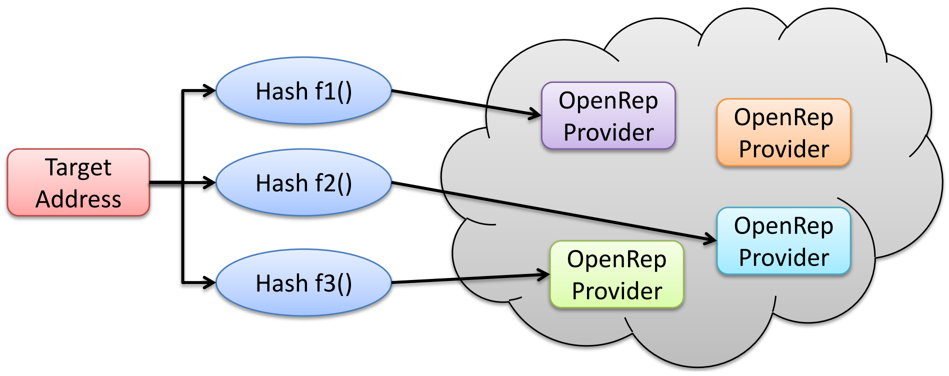


Figure 2.1: A Distributed Hash Table being used to locate supernodes which contain information regarding the target address. Hash functions process the target's identifier to determine nodes that store replicas of the their reputation profile.

information being collected. Because individuals in a P2P network are capable of acting as their own provider, and therefore store their own reputation information, the integrity of the information cannot always be trusted. Replicating reputation profiles over the network enables the use of threshold trust metrics, whereby a set of m from n supernodes must agree to verify a providers response.

2.2 Environment

OpenRep is composed of a collection of nodes that implement various components of the OpenRep stack and perform particular roles. Nodes can be categorised as providing three forms of functionality, acting as a *provider*, *source* and *agent*. A single node can provide the functionality to accomplish any combination of these roles. The flow of information between nodes can be seen in Figure 2.2. The figure shows reputation systems exposing an OpenRep node to participate within the environment. The figure also contains the depiction of a reputation system acting as both a *source* and *agent* at once. The following briefly describes the role of each of the nodes in the environment.

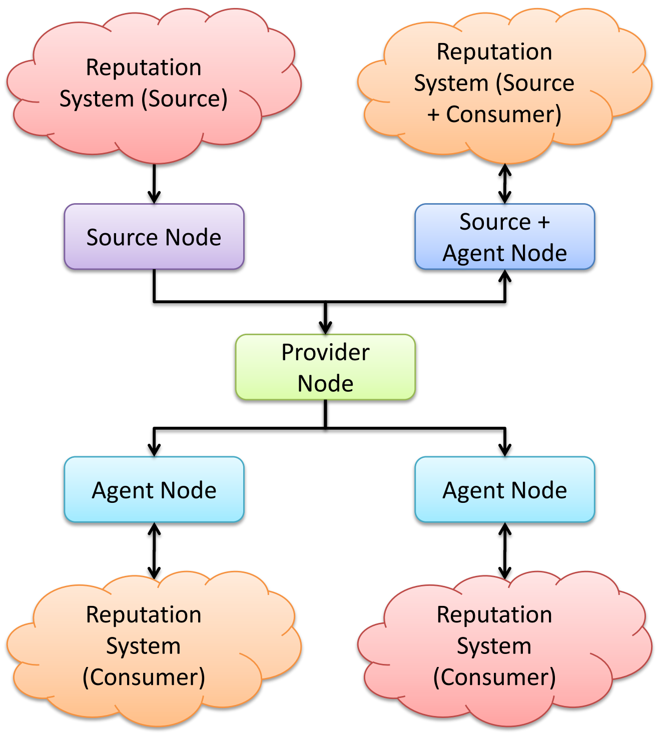

Reputation System (Source)
Reputation System (Source + Consumer)
Source Node
Source + Agent Node
Provider Node
Agent Node
Agent Node
Reputation System (Consumer)
Reputation System (Consumer)

Figure 2.2: The flow of information between reputation systems using OpenRep. Reputation information is generated in reputation systems and transferred to an OpenRep provider for distribution. Agent nodes operate on behalf of reputation systems to retrieve information from providers.

- **Provider:** The provider nodes are OpenID identity providers extended with OpenRep to store and distribute reputation information. Provider nodes supply both OpenID and OpenRep services which act as an identity provider and a reputation provider respectively. Within OpenID, providers allow users to join the environment and act as authentication points. The primary role of the provider nodes, with regard to OpenRep, is to store reputation profiles for the identities associated with the provider.

- **Source:** A source node represents the reputation generating services within an environment. Source nodes generally provide information to provider nodes regarding entities within their domain. For OpenRep to act as a global reputation mechanism, individual reputation generating services provide a source node to expose the collected information. Because nodes have accompanying reputation profiles, the trustworthiness of sources can be evaluated when aggregating information at the provider level.

- **Agent:** The agent nodes represent entities requesting reputation information from providers. Agent nodes act on behalf of individuals or reputation systems, with the goal of retrieving the reputation value associated with a target identity. Agent nodes are also capable of verifying information retrieved from a provider node by evaluating the response against that from other providers in the environment.

2.3 OpenRep Stack

The OpenRep stack comprises of three distinct layers, as shown in Figure 2.3. Information flows up the stack, originating at reputation sources and being expressed within the collection layer. Information is then interpreted at the composition layer and prepared for distribution throughout the network. The highest level provides the ability to base decisions on fine grained information. The following briefly describes the roles of each of these layers:

- **Collection:** The lowest layer of the stack, collection, provides the ability to receive information from heterogeneous reputation sources through a standardised API. The role of the collection layer is to process raw reputation information into a format that is interpretable by

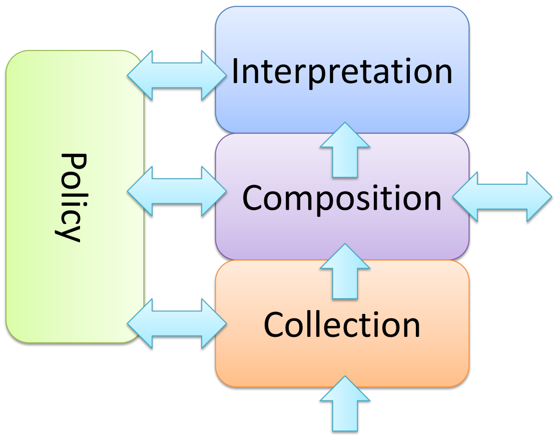


Figure 2.3: The OpenRep Stack. Information is collected through the Collection layer, from raw reputation sources, or through the Composition layer, from other OpenRep nodes. Information flows up the stack and is evaluated against policies at each level.

the composition layer. The reputation language, discussed in Chapter 6 is an essential element of the collection layer. OpenRep provider and source nodes are both required to implement the collection layer as they are responsible for gathering information from individual reputation sources.

- **Composition:** The composition layer is responsible for processing and managing information at a higher level than the collection layer. The role of translating the raw reputation from the collection layer into usable information is key to the OpenRep infrastructure. The verification system, presented in Chapter 7 defines a method to safely and reliably interpret the information from the collection layer and from other OpenRep nodes. Combining the composition and collection layers provides the ability to aggregate information. The composition layer also provides the functionality to communicate with other OpenRep nodes. The protocol to exchange reputation information between OpenRep nodes, described in Chapter 5, is also situated within this layer. In order to participate in the OpenRep environment, the composition layer must be integrated into the node.

- **Interpretation:** The interpretation layer contains the decision making functionality of the system. Capabilities associated with the extraction of specific information from a resource are included in this level. Generally, the interpretation layer is implemented when a node requires the ability to examine and base decisions on information regarding individual entities.

When implementing the OpenRep model for an existing reputation system, the functionality required by the node determines which layers of the OpenRep stack must be implemented. For example, a reputation provider node is generally responsible for storing, aggregating and distributing reputation information rather than collecting or consuming it. In general, providers do not need the ability to interpret and act on the reputation information they receive. Therefore, when integrating the OpenRep stack into a provider node, only the lower two layers are necessary.

Chapter 3

Related Work

Reputation is evident in many domains, from implicit word of mouth systems found within social communities to explicit reputation facilitating mechanisms in eBay. This chapter presents an overview of reputation information, from its conception in social settings through to its application in computer science. Reputation systems and a set of defining characteristics are then discussed. Finally the advancements of online identity and the existing work regarding reputation exchange is given.

3.1 Trust and Reputation

Trust is defined as the extent to which one can become reliant on another [22]. It is inherently difficult to responsibly trust others without a priori knowledge of their historic behaviour under certain circumstances. To advance online collaboration, a level of trust over another's ability to fulfil their obligations is required.

Reputation has been employed widely as a mechanism to institute trust within a domain. Reputation is the general consensus about ones character or standing [31]. Ones reputation can be used as a tool to bootstrap the process of building trust within others, allowing confidence to be placed on those of high repute. Reputation systems facilitate the ability to establish and build trust within an environment, contributing to the practicality of many online enterprises.

3.1.1 Reputation in Social Interactions

Through sociology, the study of human behaviour and society, it is apparent that social organisation is a fundamental, age old concept within human society. Instinctively, it is in the best interest of short term users to cheat one another. Dellarocas [17] presents word of mouth systems and their role in historic human societies. Word of mouth systems allow feedback and opinions to be propagated throughout an environment. Dellarocas explains that word of mouth systems were originally employed due to their ability to promote solidarity without the requirement of formal law being defined and enforced.

Within a social community, reputation is defined as the characteristics or standing associated with an individual by their peers. The foundation of an individual's reputation is established by comprehending previous behaviour. Word of mouth systems enable others to be informed of historic behaviour, effectively distributing opinions that represent ones reputation.

Raub and Wessie [58] state information collected through the examination of one's interaction with a peer can later be used to predict behaviour with other peers. The authors go on to explain that if one can anticipate the consequences of their current performance on future peers behaviour, they have an incentive for a trade-off between the short and long term effect of their decision. Raub and Wessie demonstrate that when present, reputation encourages individuals to consider both short and long term payoffs and promotes cooperation within the environment.

3.1.2 Reputation in Game Theory

Due to the significance of reputation in human society, the acts of generating and utilising reputation information have been comprehensively studied by economists using game theory. Wilson [75] describes reputation as a characteristic associated with players when there is doubt about the type of a player in the mind of another. The predictive power of reputation is based on the concept of past behaviour being indicative of future behaviour. When players have access to historic game outcomes, they are able to improve their long term gains by persuading others to believe they are of a certain type that best suits their interests.

Dellarocas [17] explains the requirement of initial uncertainty in other uninformed players is essential for reputation effects to take place. Consider repeated games between a long run player and multiple short run players. If no doubt over the long run player's type exists, the short run players will choose their Nash equilibrium [49] action. Dellarocas describes that building reputation enables the long run player to improve their payoffs. A long run player with a record of playing a cooperative action, is trusted by subsequent short run players to do so in the future.

Kreps and Wilson [35] and Milgrom and Roberts [47] describe the reason behind this behaviour through the use of long run players with locked behaviour. Locked behaviour implies a player is predestined to play a single action throughout their lifetime. With the example of an auction, the authors explain that when the advantages of having a reputation to perform well are sufficient, sellers overcome their temptation to cheat and try to acquire a reputation to deliver high quality goods. Based on the sellers past behaviour, buyers are able to place higher bids, increasing the long term payoffs of the seller.

3.1.3 Reputation in Computer Science

Computer based applications often aim to incorporate the advantages of trust that has been proven within social settings. Reputation systems have become increasingly wide spread and are prominent features in many online environments. The necessity of trust within pseudonymous online systems is evident when interactions have significance. The underlying concept of trust facilitating mechanisms is to provide an indication of an entities likelihood to act honestly. With the assumption of historic behaviour being indicative of future behaviour, entities are more likely to behave honestly due to the expectation that their current decision will have future consequences.

Within online environments, trust can be seen as either implicit or explicit. Implicit trust is often found within established organisations, such as online banking, where honesty is implied. Other settings, such as online auctions, require the explicit definition of a mechanism to manufacture trust. For reputation mechanisms to produce trust, entities need to be uniquely identifiable and actions must be associated with an individual. Methods of collecting, aggregating and distributing the information to others in the environment are necessary for the mechanism to function. Others can then interpret the information as a suggestion of the targets dependability. Along with the consideration of other quality of service metrics, reputation can be used to form opinions and enhance the accuracy of the decision making process.

Reputation mechanisms provide a significant motivation for entities to contribute beneficially to online environments. Resnick et al. [61] and Quill [57] demonstrate the value of one's reputation in eBay, showing that maintaining an honest and reliable reputation leads to more bids, a higher sale price and more sales in total.

3.1.4 Reputation Systems

Reputation systems are generally catered to best suit their target domain. Due to the proprietary nature of many of these domains, from network routing to peer-to-peer architectures, the differences between individual reputation systems can vary drastically. Through our work defining a taxonomy of reputation systems [28] and the examination of many systems from both academic and industrial settings, a set of categorical features have been identified. The following describes some of the categories presented in the taxonomy.

- **Governance:** The governance of a reputation system represents the method of control. Reputation systems are volatile, where users and information change frequently. The governance of a system relates to the method of organisation. Centralised governance, such as that utilised by Amazon and eBay, refers to systems with a focal point of authority. Although the underlying fabric is almost certainly of a distributed nature, the governance is managed by a singular body. Distributed governance describes multiple entities working in unison, often without an overseeing moderator, to establish a reputation system that can operate over the environment dynamically.

- **Entities:** An entity is the target of a reputation system, typically representing either people or resources [74]. Reputation systems can be categorised by the targets they cater for, be they individuals or groups. Individual entity based reputation systems are extremely prominent in online environments, with a focus on specific users and resources, such as books or films. Super node [77] utilising systems can also be considered as individually based. Group based reputation systems focus on servicing collections of individuals, whether they are long term alliances, or short term organisations. Reputation systems designed for groups allow new contexts of information to be considered, for example; group size, rate of growth and rate of churn [71]. Sabater and Sierra [64] introduced the concept of neighbourhood reputation in ReGreT, focusing on the relationships of neighbouring entities to the target.

- **Motive:** Reputation systems are often assumed to employ an incentive based motivational mechanism, in which trustworthy entities are rewarded and consequently, malicious entities are punished through a lack of reward. Rabahi et al. [59] with CONFIDANT and Aberer et al. with P-Grid [3] both reverse this approach. Instead, they each employ a disincentive based mechanism in which entities are assumed to be cooperative until identified as malicious and are in turn addressed.

- **Collection:** A reputation system's collection method describes the techniques used to capture information regarding an entities performance. Reputation systems often rely on direct collection methods such as experience and observation. Direct collection requires one entity to either interact with another entity, or monitor an interaction between other entities to form a personalised opinion on how well entities behaved [71]. Reputation systems will typically employ some degree of indirect collection as well. An indirect approach involves information being obtained from other entities, be they individuals, groups [40] or external repositories [41]. Sepandar et al. [32] explain that utilising both personal experience and others experience provides a better basis to form decisions. Reputation systems can further be classified as to whether they incorporate derived information gathering techniques, allowing information from diverse sources, often not explicitly designed to be used as a reputation source, to be utilised during the decision making process.

- **Context:** Contextual information refers to the different types of transactional information collected and distributed within reputation systems. Reputation information can be generated in a variety of ways, each associated with contextual information that is often not collected. Single context reputation systems focus on one concept of the transactions being monitored. Typically these systems support an encompassing review of an interaction, for instance, recording whether it was a positive or negative experience. Information is then aggregated and presented in a simplistic manner where reputations are often consumed as a single value. These systems support an individual perception of interactions and utilise the collected information in a structured, predefined way. Other perspectives of reputation are also important. When trading goods, not only the price, but the quality, delivery time and after sales services all play significant roles when assessing an item [65]. The degree in which reputation systems support multiple contexts differs substantially. PeerTrust [79] presents two innovative context factors for trading, taking into account the value of transactions and the level of participation within the community. The contextual factors provide a greater understanding of the trade and measures the benefit of entities to the environment. ReGreT [63] includes fine grained reputation information representing the frequency of overcharging, late delivery and quality of historic transactions. Sabater and Sierra [62] have extended ReGreT to feature a social contextual factor, allowing trust to be extracted from groups and communities associated with the target.

- **Representation:** The format in which reputation information is described, exchanged and interpreted varies between reputation systems. Through the investigation of existing reputation systems and their symbolisation of information, a set of common types can been determined. Binary types represent entities as either trustworthy or untrustworthy, for example how P-Grid [4] defines global trust. Discrete values are easily interpreted and are classified as either bounded or unbounded. Bounded discrete values describe fixed scores, such as a zero to five ratings, or Slashdot's[1] rated set of terrible to excellent. Unbounded discrete values show accumulative reputation scores, for instance within eBay, allowing the score to reflect the growing amount of feedback received. Continuous information is shown as a floating point number, ranging from zero to one, such as within

[1]http://slashdot.org/

EigenTrust [32]. String based feedback allows the expression of information in much greater detail than an individual numerical value. Text based feedback comments enable contextual information and other factors describing the transaction to be exchanged. Due to the complexity associated with autonomously processing text, comments are typically intended for human consumption. Collections, such as vectors, facilitate the description of multiple pieces of information while maintaining relationships. PeerTrust [79] employs collections to express separate values regarding individual contexts for each transaction.

3.2 Identity

With advancements in identity management, reputation systems are capable of associating actions with entities across sessions. The proprietary nature of most reputation environments has limited the ability to openly exchange reputation information. Sources typically maintain a private set of user identities and pseudonyms which is often incompatible with other sources. The ability to effectively associate pseudonyms from various environments with a unique individual is a requirement for sharing information between sources.

Single sign-on (SSO) solutions have been developed as a method of reducing the authentication overhead for both services and users. SSO allows users to authenticate once and then move seamlessly between participating services without the need to log in at each. Services that provide SSO simplify the log in process by allowing users to authenticate a global identity.

Multiple SSO frameworks have been created to provide global identity management systems, such as Microsoft Passport[2], Shibboleth [9], Kerberos [50], Liberty Alliance [72] and OpenID [60].

[2]http://www.passport.net/

Liberty Alliance extends basic SSO by incorporating an identity mapping service. The service allows users to request identity tokens that allow the distinction of users in a privacy preserving manner [72].

OpenID is a widely used standard for authentication and identity, simplifying the authentication process across platforms by providing a distributed SSO framework. Users are able to authenticate themselves with a third party by proving the ownership of an identifier. Identifiers are resolvable digital addresses, either in the form of a URL or XRI. The location of the identifier typically provides an XRDS document, describing the identity provider. Third party services can then contact the provider, prompting the provider to authenticate the user. Because the identity is simply a resolvable address, the customisation of the content is possible.

3.3 Reputation Exchange

The exchange of reputation information is reliant on many factors. The following section presents the existing work in the area along with the OpenRep system currently being developed.

Pingel et al. [54] have investigated the concept of *Cross-Community Reputation*, in which the advantages and issues of combining reputation information from different communities are discussed. The authors have designed and created a multilateral, secure, reputation system allowing reputation to be exchanged between interoperable communities. The work is demonstrated across communities using phpBB[3], a bulletin board software package.

Grinshpoun et al. [24] have focused on the issues regarding the exchange of reputation information. They discuss the advantages of sharing information and present a set of problems and policies related to the exchange between environments. Further work examining the privacy concerns and tradeoffs when sharing information has also been presented by

[3]http://www.phpbb.com/

the group [21].

The Organization for the Advancement of Structured Information Standards[4] (OASIS) are working to create an Open Reputation Management System (ORMS). The goal of the ORMS Technical Committee is to create a specification for representation, calculation and exchange of reputation information. Although not focused on the performance of calculation, ORMS intends to describe the relevance of the result within a transaction.

OpenRep is designed to leverage the OpenID architecture to facilitate reputation information. From the extendibility of the OpenID framework, as discussed in Section 3.2, reputation information can be integrated into the identifier's documents. The ability to provide and associate reputation values with a globally unique identifier, allows reputation information to be openly described and exchanged. OpenRep is designed to extend the functionality presented by Pingel et al. [54] and Gal-Oz et al. [21] by providing a solution to federate reputation systems from diverse domains and without intrinsic compatibility.

3.4 Summary

A historic account of word of mouth and reputation systems has been presented with the advantages of reputation within social settings being evident. The concept of historic actions being indicative of future behaviour is vital for cooperation within communities. Analysing reputation systems through game theory demonstrates the potential reward of being considered reputable. The integration of reputation systems into the computer science domain attempts to apply the benefits of reputation within functional applications. Through the evaluation of many existing reputation systems, a set of characteristics capable of describing reputation systems has been constructed. Some of these features have been presented to provide insight into the potential variety of diverse systems. The advance-

[4]http://www.oasis-open.org/

ments in online identity that provide the ability to globally identify entities have been discussed. With the unique identification of entities, sharing reputation information is possible. Existing work regarding the exchange of reputation has been given. This chapter has presented a review of relevant related work, providing a basis to the work discussed in this thesis.

Chapter 4

Design Strategy

Standardising the representation of reputation information is a crucial step towards increasing the interoperability of diverse reputation systems. Open reputation standards reduce the cost and complexity of sharing reputation information, enabling new reputation based opportunities to be explored.

This chapter describes the requirements and design goals of an architecture that supports the exchange of reputation information in the OpenRep architecture. The chapter first discusses the motivation behind constructing a description and interpretation system to enable the exchange of reputation information between OpenRep nodes. The requirements of the system are then presented and supported with three case studies, used to drive the research and demonstrate the capabilities of the system once complete. Finally, a set of overall design goals are discussed with reasoning as to their significance.

4.1 Motivation

Reputation systems are becoming increasingly popular and more prominent in a wide range of domains. Proprietary reputation systems have been designed specifically to cater for the requirements of the domain in which they are used. This is due, in part, to the fact that almost every aspect of a reputation system can be customised to better accommodate their particular environment. However, the diversity of reputation systems limits the ability to share and utilise information collected in disparate domains.

Reputation information can be generated with drastically different intentions and significance, for example from reporting theft through to superficial popularity measures. Generally, reputation systems cater for a single type of entity, for example humans, machines, products or groups, which therefore further complicates the process of sharing information across communities. Because the information can differ semantically, contextually and with regard to information type, it is difficult to exchange information in a meaningful way. With the proper description of information, existing boundaries can be removed, allowing the integration of information sources that would otherwise be divided. Establishing a standard method to describe reputation information will allow sources to effectively communicate and collaborate.

Integrating a description and interpretation system into the OpenRep architecture establishes a common language amongst participating sources. Through the typed sharing of information, individuals can be exposed to diverse forms of information, providing greater insight into others and increasing their ability to make informed decisions.

4.2 Case Studies

To effectively gather the requirements for the description and interpretation system, a thorough survey of reputation systems is necessary. The description and interpretation system is responsible for exchanging reputation information between reputation sources. A fundamental requirement of the system is that it is capable of facilitating the exchange of information from a wide range of reputation systems.

Domain analysis is one technique that can be used to emphasise features of reputation systems and focus the development of the description and interpretation system. To accurately model each domain three distinct case studies have been defined to assist in the process of establishing the requirements and design goals of the system. The case studies also provide a practical application scenario that can be implemented to demonstrate the capabilities of the system. The case studies have been specifically chosen to best represent reputation systems in real world situations, demonstrating how information from different types and contexts can be integrated. The three case studies described in this section cover an electronic auction, a virtual community and a social network application.

4.2.1 Auction

The first case study is focused on the electronic auctioning industry. This area has been selected as it represents a common example of reputation systems in existing research and is exemplified by many real world applications. Through the examination of various electronic auction places in active use, such as Amazon[1], eBay[2] and TradeMe[3], requirements can be generated for a well understood and practical use of reputation systems. The auction case study is focused on users generating and reviewing information regarding other traders in the environment. There is a strong requirement to aggregate reputation information over time to so that users can analyse how other traders have performed historically during transactions.

4.2.2 Virtual Community

A virtual community based case study represents a source of popular and widely implemented reputation systems. The virtual community study

[1]http://www.amazon.com/
[2]http://www.ebay.com/
[3]http://www.trademe.co.nz/

characterises commonly used online environments, such as Reddit[4], Digg[5], Epinions [6], IMDb[7] and various other domain specific forums. This case study provides a platform to demonstrate how OpenRep can be beneficial to a form of reputation system that is primarily used as a standalone service without consideration for exchanging information. The prevalence of both human and automated machine users in these communities presents the question of should these categories be treated differently, and if so, how can they be differentiated within the system.

4.2.3 Social Network

A social networking case study provides an alternative perspective from the other case studies and allows the system to be tested in a highly individualised socially-oriented reputation setting. This case study demonstrates the potential of the OpenRep system to support an extensive range of information types and reputation systems. The case study focuses on a human relationship management service, in which the interactions of participants in a social network are monitored and evaluated as a measure of trust. The service ranks acquaintances based on the level and type of interactions between them and the user. This information can be used or distributed as an indication of the strength of the relationship and therefore trust between the users. The case study also presents the requirement of customisable policies to enable others to interpret such information, allowing the allocation of specific weights to different domains and types.

[4]http://www.reddit.com/
[5]http://digg.com/
[6]http://www.epinions.com/
[7]http://www.imdb.com/

4.3 Requirements

The main goal of the description and interpretation system is to allow different reputation systems to share information meaningfully. To achieve the level of integration desired between reputation systems, a method of communicating effectively between systems is required. The system must provide a suitable medium in which information can be expressed, exchanged and interpreted. The requirements of the system are:

Requirement 1 *Description and Interpretation System*

1.1 *Standardise the expression of reputation information.*

1.2 *Enforce the correct use of the description mechanism.*

1.3 *Facilitate the exchange of reputation information between heterogonous systems.*

1.4 *Provide a mechanism to interpret the information.*

4.3.1 Reputation Description

There are two mechanisms that are most appropriate to standardise the description of reputation information, these are the creation of a Domain Specific Language (DSL) or the definition of a typed schema such as XML [8]. The development of a DSL has been selected over the definition of a typed schema for multiple reasons.

A DSL sacrifices generality for expressiveness in a particular domain [45]. The key requirement of the reputation medium is to allow the expression of information without any loss of detail. A DSL provides the ability to define a vast range of types, collections and constraints, allowing information to be expressed in fine grained detail while maintaining the relationships between data types. Functions in particular are a valuable form of communication between reputation systems. A DSL enables direct description of functions and parameters. DSLs also offer a human readable platform of communication, increasing maintainability and allowing users to customise information. The primary disadvantages, in this context, of using DSLs are the cost of learning a new language and the transformation from a data structure, to a linear expression, then back to a data structure for processing. Due to the diversity of reputation systems and the information generated, the transformation of information is necessary whether a language or schema is used. The cost of learning a new language can be minimised by leveraging existing languages to increase the intuitiveness of the reputation language.

The requirements of the reputation language are that it be powerful enough to enable the description of any reputation information, including collections of types and the relationships held between values. Common operations performed on reputation information should also be supported, demonstrating the ability to express requirements and other operations. The core requirements are therefore:

Requirement 2 *Reputation Language*

2.1 *Include sufficient types in the language to express reputation information.*

2.2 *Support collections and data structures to represent relationships between information types.*

2.3 *Provide an adequate set of operations and functions.*

2.4 *Cater for both human and machine based information.*

4.3.2 Verification System

In order for the reputation language to function properly as a standardised communication method, confidence in the use of the languages is necessary. In particular, a mechanism to ensure syntactic and semantic constraints are enforced is required. The correct expression of reputation information, with regard to syntactical description and sound type use, is a nontrivial property [10]. Before processing an instance of described information it must be shown to be valid, for example ensuring functions are specified with compatible parameters and that types are used appropriately. Policies are necessary to describe type hierarchy and function schematics. The requirements for the verification system are:

Requirement 3 *Verification System*

3.1 *The use of the reputation language must be validated.*

3.2 *The definition of what constitutes a type error must be established.*

3.3 *Customisable policies are required.*

4.3.3 Exchange Protocol

A standardised exchange protocol is required to share reputation information between heterogonous reputation systems. This protocol must ensure reputation information is packaged and exchanged in a generic structured format such that consistent interpretation of the information can be performed. The exchange protocol defines a means of encoding reputation information when communicating between sources. The encoding must allow the information to be packaged without loss of meaning, integrity or contextual information. To accomplish this goal, a method of expressing contextual information must first be defined. The requirements for the exchange protocol are:

Requirement 4 *Exchange Protocol*

4.1 *Encode information without the loss of meaning.*

4.2 *Provide the ability to encode contextual information.*

4.3 *Ensure information integrity.*

4.4 Design Goals

The previous section outlined the core requirements that must be supported by the description and interpretation system. These requirements each influence the design of the description and interpretation system, incorporating the language, verification system and encoding protocol together as a single useable service. The system is designed to fit within the OpenRep framework, providing a standard method of communication throughout the reputation environment.

The verification system must be abstracted from the end user to simplify use when interpreting input. To accomplish the desired encapsulation, an interface to the verification system must be defined. The interface is responsible for marshalling information from the validator to the semantic checking system and returning outcomes to the caller. The validation process is an asynchronous process and therefore can be handled using an event based architecture, allowing callers to subscribe to different notifications and asynchronously invoke functionality as needed. Finally the encoding of information into the exchange protocol must also be addressed. With these considerations in mind, the core design goals for the description and interpretation system are:

Design Goal 1 *Description and Interpretation System*

1.1 *Create a portable system that can be incorporated into the OpenRep stack.*

1.2 *Provide a simple interface to the system, reducing the complexity to process reputation information.*

1.3 *Build an event based verification system, allowing users to asynchronously invoke particular functionality depending on the result of the execution.*

1.4 *Provide a secure method to exchange reputation information.*

The design of the system can be seen in Figure 4.1. The figure depicts the components of the system. The verification system and validation unit are abstracted away from the user, behind an interface. The parser and lexer rules are utilised by the validation unit to parse information. The exchange protocol is situated between the description system and other OpenRep nodes, enabling information to be exchanged.

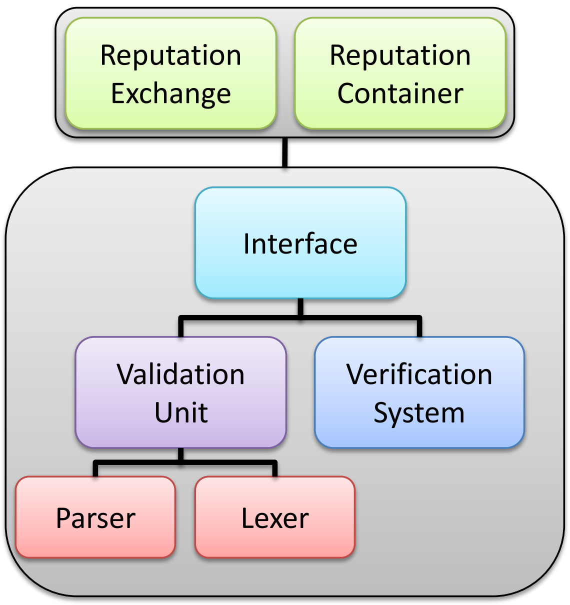


Figure 4.1: An overview of the description and interpretation system.

4.4.1 Reputation Language

A language can be defined as a grammar containing a set of lexer and parser rules. The decision of which parser generator tool best meets the stated requirements must be addressed before the expression of the language can take place. A lexer is responsible for describing the fundamental elements of the language, for example key words, operators and what constitutes an integer. Once the lexer rules are constructed, the parser rules can be defined as groupings of the lexer terminals. The parser is responsible for a higher level role of the language, determining the combinations of lexer tokens that represents a valid description of reputation information. Creating an intuitive language that leverages standards used in prominent programming languages will reduce the overhead required to learn the reputation language.

The reputation language requires a sufficient set of data types to describe reputation information. To ensure this requirement is met, many reputation systems have been surveyed to compile a expansive set of types to support. Through the examination of the identified set of types, the language can be defined to accommodate the elemental features of those identified, allowing the innovation of customised reputation types to continue being supported. The design goals for the reputation language are:

Design Goal 2 *Reputation Language*

2.1 *Define an appropriate set of terminals and lexer rules.*

2.2 *Define a non-ambiguous set of parser rules that fulfil the requirements of the language.*

2.3 *Have the language be syntactically intuitive and easy to use.*

4.4.2 Verification System

For reputation systems participating in the OpenRep environment to have confidence in the correct use of the reputation language, a system to verify language instances prior to interpretation must be included in the description and interpretation system. Essentially, the verification system must comprise of two distinct elements: a language validation mechanism and a semantic constraint enforcement system.

The purpose of a type system is to prevent execution errors when executing the input [10]. For the verification system to work as intended, potential type errors must be identified. The verification system is designed to work with the defined reputation language to provide a typed method of describing reputation information. Before a semantic enforcement system can be designed to identify and alleviate type errors, the definition of a type error must be defined. The design goals for the verification system are:

Design Goal 3 *Verification System*

3.1 *Ensure the syntactical correctness of instances of the reputation language.*

3.2 *Establish a set of type errors and construct a static type enforcement system to ensure potential type errors are identified.*

3.3 *Define and implement the functionality to validate type compatibility with regard to type hierarchy.*

4.4.3 Reputation Exchange Protocol

When designing a comprehensive solution to meet the requirements for the exchange of reputation information between OpenRep nodes, prominent encoding standards and languages must be investigated. Definition languages, such as Web Service Definition Language [12], Interface description languages and exchange protocols such as SOAP [51] demonstrate a successful method of encoding information. From these XML based languages and protocols, attributes that could potentially be useful for the reputation exchange protocol can be found.

The design of the reputation exchange protocol has been broken into two distinct XML Schema Definition (XSD) [23] schemas, the Reputation Exchange (RX) and Reputation Container (RC), to facilitate the transfer of multiple pieces of reputation information at once. The goal of the RC schema is to encompass reputation information elements and include the security elements associated with data integrity. The RC schema is also designed to ease the burden of processing the reputation information, by providing additional information identifying the number of items being transferred and the role of the exchange, be it a request or function invocation. The RC schema can be seen in Appendix B.

The RX schema is designed to fulfil the requirement of encoding instances of reputation information without losing semantic meaning. The RX must provide adequate encoding capabilities to allow reputation information to be explicitly described, along with any associated contextual information and details of the source and target. The RX schema must also facilitate an optional *transaction* element, used to exchange entire histories of a users interactions, potentially providing reputation provenance. A mandatory summary element is to be included for each piece set of information sent. The RX schema can be seen in Appendix C. Encoding instances of reputation information as RX elements and then wrapping RX elements within an RC schema provides a basis for information to be exchanged through a standardised format.

The method of interaction between sources must also be established. For reputation systems to work in a coordinated manner, key aspects regarding interoperability must be defined. Common interfaces and a specification regarding the requirements of interactions are concerns that must be addressed before the exchange protocol can be instituted.

Design Goal 4 *Exchange Protocol*

4.1 *Create a schema to encode reputation information without loss of semantic meaning.*

4.2 *Create a schema to encase the reputation information, providing security features and parsing advantages.*

4.5 Conclusion

The proposed description and interpretation system is designed to enable the OpenRep architecture to exchange reputation information between reputation systems. This integration of reputation systems will enable access to a greater amount of reputation information, from a variety of perspectives, which in turn will allow more informed decisions to be made.

Three case studies have been defined to generate requirements and focus the implementation. The case studies have been selected for their unique features and each provides an insight into a unique reputation system that represents a real world scenario. The case studies have demonstrated their value by identifying areas that may otherwise have been overlooked, such as the complexity involved with integrating human and machine sources as well as the requirement to encode contextual information.

The requirements of the description and interpretation system have been identified and discussed. From the requirements, the design goals for each component of the system have been defined. The design goals are utilised during the creation and implementation of the system as a whole. This chapter has presented numerous concerns that must first be addressed in the implementation of the description and interpretation system.

Chapter 5

Techniques and Tools

The questions raised in Chapter 4 are fundamental to the functionality of the description and interpretation system. Concerns relating to OpenRep's interoperability standards, the integration of human and machine entities, contextual representation and the selection of an appropriate parser generator are essential to the operation of the system. A discussion of the assumptions and decisions made for each of these topics is given below.

5.1 OpenRep Interoperability Standards

For the reputation systems to interact, a common form of communication is required. It has been decided that standardising the methods of communication will be enforced through three mechanisms. The communication standards focus on the exposed interface, a set of standards and a protocol.

The interfaces exposed by reputation systems participating within OpenRep are required to conform to the Representational State Transfer (REST) [20] style. REST utilises the HTTP protocol for communication, meaning a RESTful interface supports HTTP requests, such as GET and POST. The HTTP and HTTPS protocols also allow caching and proxies to be used to assist in high load situations, increasing both the scalability and usability of the service. Finally, REST encourages complex tasks to be organised into resources that operate through a standard interface.

The OpenRep specification provides a set of common parameters to generalise the interactions between reputation systems. We have designed the specification to establish a standard method of communicating between reputation systems within the OpenRep environment. The specification defines the structure and function of requests and responses.

In addition to the OpenRep specification, the protocol to exchange reputation information must be defined. Through the use of a standard interface, an open specification determining the requirements of interactions and the development of an exchange protocol, interoperability between reputation systems is possible.

5.2 Integrating Humans and Machines

Before OpenRep can operate as a federated reputation system, the plausibility of integrating human and machine based sources must be addressed. Until machines are able to pass a Turing test [73], there is an obvious discrepancy between an automated machine entity and a human user interacting. Reputation systems are designed to function for a specific set of users. In most well known reputation systems, such as eBay, the user base is primarily human. Through the investigation of various human and machine based reputation systems, a set of parallels and differences can be established. From this set, the complexities and potential solutions associated with integrating human and machine based reputation systems should be identifiable.

To accomplish the goal of identifying differences and challenges when merging inconsistent user bases, a number of steps need to be taken. Firstly, the key phases in a reputation system need to be recognised and categorised. By categorising primary functions and points of action in a reputation system, behaviour can be collected and analysed. Example case studies can then be investigated with relation to the reputation system phases, allowing behavioural characteristics to be recognised. From the characteristics, differences and similarities can be extracted and reviewed to base the opinion of whether they should be integrated.

After investigating multiple reputation systems, seven reputation phases have been identified as definitive. The following have been determined as distinctive points in which users take action:

- **Resource Detection:** How an entity discovers resources to fulfil their requirements. Human partners are typically found through searches. Slashdot provides a mechanism in which other entities vote on content to establish a value, then if the value passes a threshold, the information is displayed to users.

- **Reputation Presentation:** How reputation information is discovered and viewed by entities. eBay and Epinions display a summary of a users reputation along with their identity. CARE [78] utilises a reputation flooding mechanism, where local repositories are frequently updated and searched.

- **Reputation Collection:** The process in which information regarding others is collected by an entity. While RATEWeb [40] primarily supports the direct collection of information, when a service is identified, a list of prior partners is also provided. The historic partners can be queried for their evaluation of interacting with an entity.

- **Calculation:** The steps taken with reputation to establish an indication of trust. Machine based entities are often extremely methodical and precise when calculating whether an entities reputation meets a threshold. Human users, for example in eBay, are often catered for by being provided precalculated values to simplify the interpretation.

- **Interpretation:** The process of understanding the calculated information. Machine based entities aim to employ precise threshold tests to determine trust. The weight given to various reviews changes considerably between human users, resulting in inconsistent evaluations. Where a machine may be programmed to calculate a negative performance as being equivalent as a positive performance, humans often assign larger weights to negative reviews.

- **Resource Consumption:** How an entity utilises a remote resource. Typically this involves an entity interacting with the target or information being displayed. CARE uses the reputation information to determine which email is spam.

- **Feedback Generation:** The process in which reputation information is generated. Generally, human users are reluctant to provide feedback and require motivating. Emotional responses can also prove unreliable. Similarly, machine agents can have difficulty determining an accurate evaluation of an interaction if unforeseen circumstances arise.

Slashdot, eBay and Epinions have been selected as case studies to represent human based reputation systems. CARE and RATEWeb were chosen as the most relevant machine based reputation systems that provided sufficient detail to analyse with regard to the reputation phases.

Once the case studies were examined and compared, the similarities and differences were apparent. With the assumption of honest entities, the goal of each entity type in the reputation system is identical. Both aim to be considered reputable and employ the system in a manner to incentivise others to trade with them and alleviate the probability of interacting with dishonest users.

Humans were found to not always perform the optimal action when attempting to accomplish an exchange. Although human users are often provided information from clinical reputation calculations; mistakes, biases and unfounded logic can heavily affect a decision being made. Machines are assumed to be programmed to act as effectively as possible.

It is also assumed decisions are based entirely on information collected within the reputation environment. Both of these assumptions are inherently false. Programmers are prone to mistakes and humans often have external relationships that can be given precedence over reputation systems.

Human users motives are not strictly pure or specifically defined, meaning they are subject to unforeseeable change. This makes interacting with them unpredictable and potentially inefficient, which may cause further trouble as machines struggle to adapt to unanticipated circumstances.

Other differences, such as fault tolerance, downtime or accounting for new sources or contexts of information were also identified as potential problems. Without an adequate resolution to these, machine entities would gain a reputation for being inadequate and unreliable, potentially causing them to be avoided by the human populace. Machines need to be able to handle rare and possibly unforeseen circumstances, while maintaining the ability to provide meaningful feedback to human users and fulfil transactions without being interrupted by faults.

Although other differences have been identified as troublesome, the key concern when integrating humans and machines is the ability to learn and adapt. Although a reputation system itself is a tool to provide adaption in partner selection, humans are able to probe other entities behaviour to develop an understanding of their motives and heuristics. This leads to the fact that a machines programming can be learnt by an observant human user. Once learnt the user can exploit the machine to their own advantage.

From the above work, the identification of a critically limiting factor, demonstrating that human and machine reputation systems cannot be integrated was not found. Though ability to provide assurance of their capabilities to achieve tasks is key to their incorporation. Provided the machines are not able to be gamed to other users advantage, their integration has been determined as possible.

5.3 Reputation Contextual Expression

Reputation information is context dependent. Without an associated context, reputation has very little value and is difficult to extract meaning from. Contextual information is often taken for granted in reputation systems due to architectures typically being single domain oriented. Schlosser et al. [65] demonstrate different forms of contextual information that is often overlooked through an example of goods being traded. They explain that the price and quality of an item are important characteristics when trading, but other aspects, such as delivery time and after sales services are also significant to decision making.

In order to express contextual information, a layered approach has been decided as the most appropriate mechanism. Throughout this research, contextual information regarding reputation values is described in layered context levels. Context levels allow reputation information to be interpreted to different extents, dependent on such factors as the similarity between the described context and the interpreters, or to the level at which it loses significance.

For example, reputation information generated within an auction is fundamentally associated with layers of contextual information describing the environment. Online auctions branch from an online domain and are refined to an electronic market based context. Further contextual layers are then applicable, for example the role of the trader (buyer or seller), the type of item being traded or the price of the item. The description of context when trading reputation information is essential. Through the association of appropriate contextual information with reputation values, meaning can be conveyed.

5.4 Parser Generator Tools

When designing a language, a parser and lexer generator need to be selected to establish the parser and lexer rules of the language. Before the expression of the language can begin, the selection of an appropriate tool to generate the parser and lexer is necessary.

Many parser generators and the generally associated lexers have been investigated and compared. Lex and YACC [37], Flex and Bison [36], Javacc [34] and ANTLR [52] were each reviewed as potential candidates to generate the reputation language. When deciding on the appropriate parser generator for the language, the main criteria was that it simplify the definition of the language while being suitable to express it fully. After comparing and evaluating each of the candidates, it was decided ANTLR would be most suitable. ANTLR is both a parser and lexer generator used to construct languages and automates the time consuming components associated with creation. ANTLR takes as input a single context-free Extended Backus-Naur Form (EBNF) [66] grammar, capable of describing both the parser and lexer rules.

ANTLR has a number of advantages over the other potential parser generators that were considered. One of the most noticeable advantages is that it could create a parser and lexer in Java, allowing fluent integration into the description and interpretation system project. ANTLR is generally considered easy to use, providing a high degree of debugging capabilities. The ability to include the abstract syntax tree structure into the grammar was also an advantage and simplifies the parsing of the output. ANTLR also provides an integrated development environment called ANTLRWorks. ANTLRWorks facilitates a graphical representation of the grammar rules and errors. ANTLRWorks also contains a debugging environment, showing the generation of parse and abstract syntax trees when processing input. ANTLR supplies less cryptic and easier to interpret error messages than those often generated by YACC and Bison.

The main disadvantages of using ANTLR are related to its complexity. ANTLR has a steep learning curve as it is able to accomplish many things. ANTLR is also considered to run slower than the YACC and Bison alternatives, it also requires a dependency to be added to the class path of a project.

ANTLR uses LL(*) parsing, meaning it parses from left to right with arbitrary lookahead with backtracking, rather than YACC and Bison's LALR parsing. The Java Virtual Machine [38] is used to generate the parser and lexer.

A lexer defines of a set of rules applying to character sequences, made to represent tokens. The lexer rules defined for the reputation language consist of the definition of literals, operators, keywords and identifiers. ANTLR also provides a mechanism for hiding input in a special channel, meaning is not processed or included as a token passed to the parser. White space and comments have been directed to the hidden channel, providing a java like commenting mechanism.

The parser takes the sequence of tokens identified by the lexer and attempts to map the tokens to rules defined in the parser grammar. The rules defined for the parser describe the language. Rules are defined in a tree structure, where input can be classified as a set of rules depending on the branches the tokens match.

5.5 Conclusion

The chapter has examined areas of work that required consideration before proceeding with the work. The concerns identified in Chapter 4, relating to fundamental features of the description and interpretation system have been accommodated, allowing the system to function as required.

The OpenRep interoperability standards have been presented and discussed. Enforcing a REST interface, standardising communication techniques and implementing an open exchange protocol enables the case studies being developed to communicate effectively.

The integration of human and machine based reputation systems is an essential element of a federated reputation system. Through the study of existing reputation systems it has been shown that humans and machines can effectively operate together, expanding the potential reach of the OpenRep system.

To meaningfully exchange reputation information, the ability to express context is essential. The technique developed to encode contextual information has been presented and reasoned, allowing it to be incorporated into the work.

Finally, the selection process of a suitable tool to generate a parser and lexer for the reputation language has been discussed. The advantages of ANTLR and the features it employs have been given to support the decision.

Chapter 6

Reputation Language

To facilitate the sharing of reputation information the description and interpretation system requires a common description mechanism. In this architecture a reputation language has been selected to enhance the ability to express information with vastly different types and structures. A discussion regarding the use of an XSD schema or reputation language is presented in Chapter 4. The reputation language standardises the method of expressing information throughout the entire OpenRep environment. The language is a fundamental element of the description and interpretation system and is necessary to achieve the meaningful exchange and consistent interpretation of information between OpenRep nodes.

The primary goal of the language is to ensure reputation information can accurately be expressed. Reputation information is often represented using different types and typically contains relationships between data elements when capturing and describing an interaction. For example, a collection of enumerated ratings are associated with a textual comment when providing feedback in eBay.

The requirements defined in Chapter 4 identified that the language must be able to describe a range of information, from individual values through to complex associations of types, without losing meaning. The expression of contextual information and whether human or machine entities generated the information must also be addressed when federating reputation systems. The language must not be ambiguous in order for others to understand it, meaning a single input cannot be interpreted to have more than one meaning. This chapter presents the design and implementation of the reputation language used in the reputation description and interpretation system.

6.1 Design

The fundamental requirements of the reputation language are discussed in Chapter 4, emphasising the need for sufficient type support. Having defined the required functionality, this section describes the method used to enforce valid use of the language, the syntax used to provide intuitive use of the language, types of reputation information, the relationships between information and operations supported by the language.

6.1.1 Language Enforcement

To ensure compatibility and provide a sound medium to express and interpret reputation information, misuse of the language is not acceptable. In order to provide assurances that the language is used correctly, two potentially dangerous conditions must be addressed, these are syntactic and semantic errors. Syntactical errors refer to the incorrect expression of information, forming an invalid description that cannot be parsed. Semantic errors imply incompatible type use, for example attempting to multiply two strings.

The practice of enforcing semantic constraints within the language has been extensively investigated. The reputation language is designed to be statically typed, where instances of the language are checked for semantic errors at compile time, before the program is interpreted [5]. Static typing provides earlier type checking and better enforcement of styles than dynamic checking at run time [2]. Through the sound definition of a static type system, a language is said to be strongly typed [76]. Because the reputation language is a method of expressing reputation values and relationships, a dynamically typed language is not necessary.

6.1.2 Language Review

An investigation of domain specific languages and prominent programming languages, such as Pascal [30], Java [38], C++ [70], C [26], Lisp [43] and Haskell [1] to name a few, highlights many common traits. The reputation language supports features from each of these prevalent languages to form an intuitive and easy to use syntax. The language is created bottom up, where primitive types are defined first and collections are built over these primitive types. From the examination of existing languages, a complete set of operators has been collected and included in the reputation language. Key words regarding types, for instance 'char', 'int' and 'array', are leveraged from established languages to increase the intuitiveness of the language, meeting the design goal specified in Chapter 4.

6.1.3 Reputation Types

The process of creating the reputation language requires the examination of many reputation systems to identify common types and features. The survey of reputation systems during the creation of the Taxonomy of Reputation Systems [28] provided a basis of reputation types and methods for expressing reputation information. As discussed in Chapter 3, it was found that most reputation information is represented as binary, discrete (bounded or unbounded), continuous, string or vector values.

In order to gain more perspective on the information types being used, the case studies have been developed. One of the most crucial steps during the reputation type discovery process has been the construction of domain ontologies for each of the case studies. Through the evaluation of the ontologies, the diversity of reputation information is apparent. The reputation language supports an extensive set of primitive types, providing a suitable foundation to express all forms of identifiable reputation information.

Domain Ontologies

Domain ontologies have been defined for each of the case studies to assist in the investigation of reputation types. An ontology is a group of concepts and the relationships defined between them [68]. Domain ontologies represent a particular part of an environment, allowing the detailed examination of specific elements. The domain ontologies for each case study describe the potential sources of reputation information, and the relationships between them, available within their respective environment. The relationships between reputation elements are vitally important as they drastically influence the meaning of the information itself. The development of ontologies enables the definition of information types and presents situational functions for their respective domains. The ontologies demonstrate that data type alone will not suffice to represent reputation information. Instead, structures that represent and express the information and relationships between types are able to describe the reputation's meaning in its entirety, providing greater detail to the consumer.

Figure 6.1 shows the domain ontology created for the auction case study. The figure depicts the elements available within the auction domain. The key sources include Price, Reliability, Shipping, Communication, Feedback and details about Users. Relationships, for instance those highlighted between Reliability, After Sales Services and Payment Method, have been identified as holding between elements that influence one another.

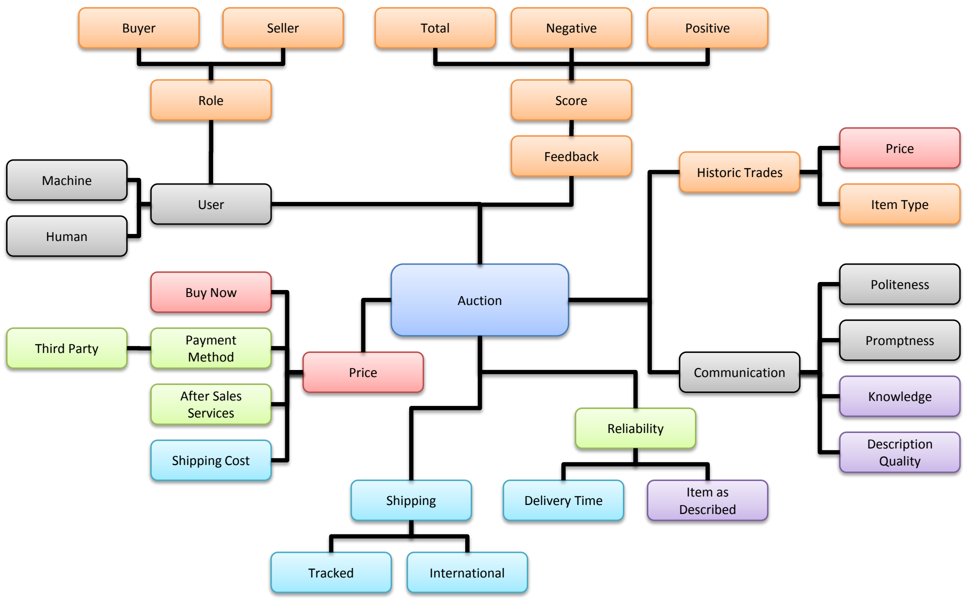


Figure 6.1: The domain ontology for the auction case study.

6.1.4 Complex Types

The examination of many reputation systems currently in use and the investigation of the case studies with domain ontologies, demonstrate that simple values are not sufficient to express the complexity associated with the majority of reputation information. Context and relationships between data types play a significant role in giving reputation information meaning. After investigating the methods users employ to evaluate reputation values, it is evident that various collections are necessary. Structures, such as arrays and customisable data sets are used to group information together, signifying a relationship between the values.

Complex data types allow the expression of information that contains more implications than primitive types can provide. Reputation information is often collected as multiple values that refer to particular aspects of an interaction, rather than a single, all encompassing, value. With the support of collections, reputation information can be expressed as a whole, encoding relationships between information and their associated meaning. Similarly, star ratings are often used to facilitate scaled reviews and cannot be described with a single value. Instead, a data type that encodes the value range as well as the value selected is necessary.

6.1.5 Functions

Users are often capable of performing operations on reputation information in both human society and computer based reputation systems, for instance filtering information for relevance or particular levels. To demonstrate the capabilities of the reputation description and interpretation system, a set of functions that have been identified as beneficial to users within reputation systems are incorporated into the language. This also meets the requirement specified in Chapter 4, accommodating an adequate set of operations and functions.

Within human society it is often practical to leverage existing relationships to bootstrap others. A vouching mechanism has been included within the language to perform this task between reputation systems. When one reputation system does not have a priori knowledge of an entity, they are able to send a vouch request to other reputation systems that have previously interacted with the target entity. A vouch request invites another, informed, source to risk their own reputation by providing confidence in the entity in question in return for further trust. Computer based reputation operations, such as filters and assertions, have been included to provide the functionality to manage reputation information and pass requests between sources. With the inclusion of functions, the reputation language supplies a rich medium in which reputation sources can communicate.

6.2 Implementation

The reputation language is implemented as an ANTLR grammar. ANTLR, as discussed in Chapter 5, is a powerful parser and lexer generator. The language is expressed as a single, context-free, EBNF grammar, as is required by ANTLR. The syntax of the language is designed to be as intuitive as possible, leveraging prominent languages in use today. Operators, such as *not equal* and *and* follow the generally accepted syntax provided by languages such as C and Java, being defined as *!=* and *&&* respectively.

Parentheses are used to indicate various groups of definition. Round brackets denote the beginning of a set of arguments for a function. Curly brackets signify variables and collections, for instance when declaring a list, this can be seen in Figure 6.2. Square brackets are reserved for value descriptions, typically symbolizing the definition of a social identifier, for example a users name. The following contains a brief schematic example for each of these.

- **Primitive Declarations:** {*type*} {*name*} = {*value*}
- **Star Declarations:** *star* {*name*} = { {*value*} .. {*value*} *:* {*value*} }
- **Collection Declarations:** {*type*} {*name*} = { {*value*} {, {*value*} } }
- **Social Value:** {*type*} { *[* {*Social Identifier*} *]* *,* *[* {*Social Identifier*} *]* {, {*value*} } }
- **Functions:** {*type*} *(* {*value*} {*operator*} {*value*} *)*

ANTLR provides the ability to form Abstract Syntax Trees (AST) when parsing values. The tree is customisable, allowing the definition of when nodes are created and increases the ability to interpret content. The difference between an AST and Parse Trees can be seen in Figures 6.2 and 6.3, where both the parse tree and AST are given for a List declaration.

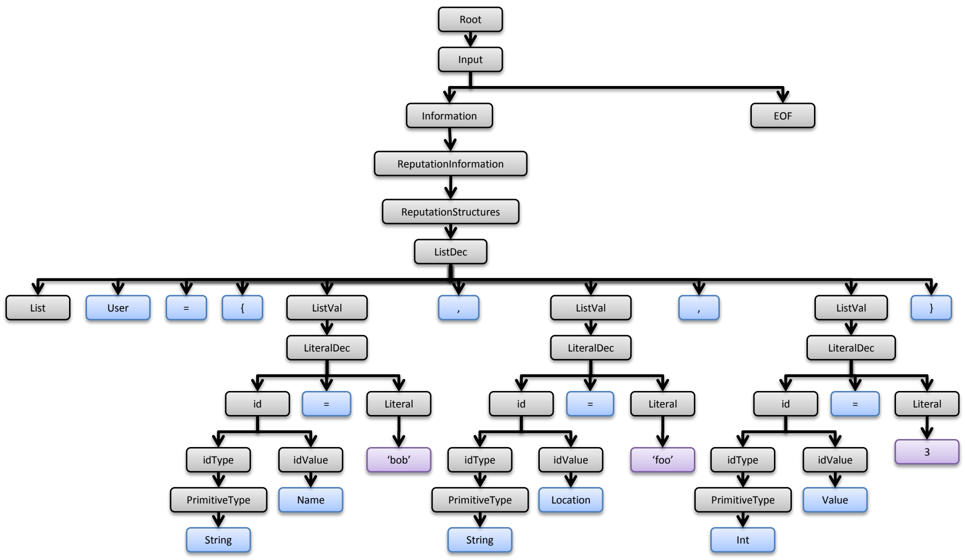


Figure 6.2: The parse tree generated when parsing a list.

Due to the differences between human and machine entities, identified in Chapter 5, the encoding of an entities type is necessary. The reputation language supports this feature at the highest level. *Human* or *Machine* can prefix any information described within the language and can then be interpreted during processing. This fulfils the requirement specified in Chapter 4.

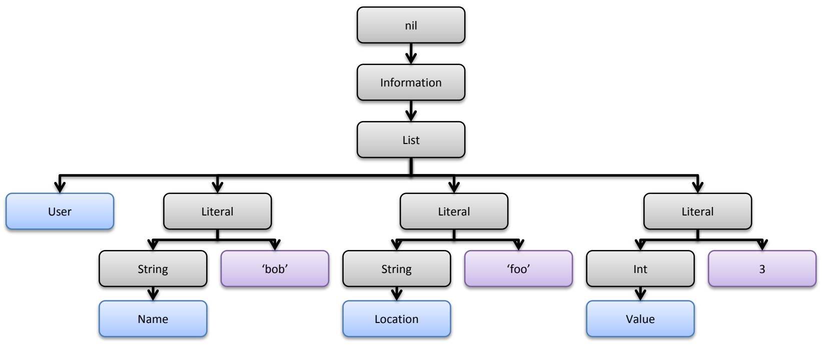


Figure 6.3: The abstract syntax tree generated when parsing a list.

6.2.1 Primitive Types

A complete set of primitive types are provided in the reputation langauge. *Characters*, *strings*, *booleans*, *integers*, *floats*, *doubles*, *identifiers* and *null* enable the language to support all forms of reputation information that have been identified. *Ranged* values, such as star ratings, are also categorised as primitive types and allow users to signify values within a range. The primitive types can be assigned to a variable or built on to provide more complex types.

Identifiers

Three forms of identifier are also included, an *ID*, *SocialID* and *SocialName*. The basic *ID* is used to name variables and structures. The requirement of this identifier is that it be prefixed with a character, then followed by any number of letters, underscore characters or numbers. *SocialID* is included to support the identifiers provided by Facebook. Facebook stores user identities as nine digit numbers which do not conform to the requirements of an *ID*. Social identifiers are encapsulated within square brackets. The *SocialName* identifier is used to identify individuals within the social environment. Names are considered valid if they begin with a letter and are followed by any number of letters, spaces, underscores, hyphens or single quotes. Each identifier can optionally be prefixed with an *idType*, further describing the identity with titles, for example *Group* or *Role*.

6.2.2 Complex Types

The complex types in the language focus on representing relationships between reputation values. The complex types build upon the primitive types, organising values to express reputation information in its entirety. Each complex type is included for a specific purpose, allowing distinct types of information to be expressed as fine grained values while maintaining the affiliations between individual elements. The inclusion of data structures and collections meets Requirement 2.2, allowing groups of information to be specified as a singularity. The following provides details of the complex types available within the reputation language with example descriptions.

- **Array:** An array is a data structure that maintains information in an organised manner. When describing an array of values, the type must be specified. During the interpretation phase of the language, the type is checked against the content to ensure valid use.
  - *array string users = {'bob', 'alice', 'sam'}*
- **List:** Lists store a collection of primitive values in an organised structure. Unlike an array, lists do not require a collection type to be specified, instead, each individual information value stored within the list is required to identify its own type. Lists allow information of different types to be combined within a single structure, as can be seen in Figure 6.3. Due to the nature of reputation information, feedback is often provided and consumed as a combination of different types.
  - *list user = {string name = 'bob', string location = 'foo', int value = 3}*
- **Enumerated Type:** An enumerated type provides a basis to create and describe bounded discrete values. Reputation systems often enable reputation to be collected and consumed through a set of distinct state ranges, for example, Slashdots' *Terrible* to *Excellent* range. The enumerated values are specified as a set of *Identifiers* separated by commas. A colon is used to denote the end of the enumerated values and is followed by an individual *Identifier*, signifying the selected value. The verification system actively ensures the correct use of the enumerated type, validating the selected value is included within the set of enumerated values.
  - *enum rating = {a, b, c : a}*

- **Star Rating:** Many reputation systems use star ratings as a prominent form of feedback. Rating mechanisms provide the capability to evaluate an entities performance within a visual scale. Percentage based techniques simplify both the generation and consumption of reputation information for users. Due to the widespread popularity of rating methods, a specific type is included in the reputation language to represent them. Star ratings are represented in a similar way to enumerated types, where a range of values are defined, followed by a colon to signify the selected value.
  - *star rating = {1 .. 5 : 3}*
- **Structure:** The most extensive method of describing reputation information is to use structures. Structures can contain any well formed type of reputation value within them. Due to the customisable nature of structures, they are ideal to express reputation information in its entirety. Structures can potentially be recursive, as *information* types can be resolved to further structures.
  - *struct bob = { string name = 'bob', int count = 3, array string content = { 'one', 'two', 'three'}}*

6.2.3 Social Types

Due to the highly individualised nature of the social networking scenario, social specific types are required to express the information regarding the interactions between users. In order to achieve the goal of creating a social network based friendship management service, the interactions and associations between individuals must be typed.

Investigating the social networking domain and constructing a domain ontology regarding potential interactions, identifies a subset of types to demonstrate the potential for exchanging information collected within a social context. Although the case study focuses on Facebook[1], similar concepts from Google+[2], LinkedIn[3] and Twitter[4] have also been considered.

The subset of Facebook interactions prove the concept of typing exchanges between users is viable. Types, such as those representing users included in the same photo, enable this unique form of reputation information to be expressed, exchanged and interpreted with meaning. The four types of information selected are as follows.

- **Like:** A *like* symbolises the action of actively approving of another party. In Facebook *likes* can be used to acknowledge other's assertions on message boards or within comments. For the purpose of the social networking case study, *likes* are the action of endorsing a third party entity, such as a sports club or musician. The type *like* describes the level of similarity between two users by comparing the sets of supported pages each user maintains.

- **Tag:** Most social networks support the concept of *tagging* users in a photo. This social type records the identities of friends *tagged* in photos with a user, representing a real world engagement of the users. Being included in a photograph with one another demonstrates a tangible relationship between two users. Policies describing the differences and weights associated with physical interactions compared to either electronic communications or similarities must be addressed.

- **Comment:** A *comment* provides an instance of electronic interaction between individuals, although it is often indirect and asynchronous. Through the collection of distinct interactions between users, a degree of association can be derived.

[1]https://www.facebook.com/
[2]http://plus.google.com/
[3]http://linkedin.com/
[4]http://twitter.com/

- **Post:** a *post* acts in a similar fashion to *comments*, though *posts* can be used to express more direct communication. *Posts* are actively created statements from one user to another.

6.2.4 Functions

In addition to the static types presented earlier in this section, the reputation language also includes a number of utility functions and operations that support active behaviour. Functions support a higher degree of flexibility and communication between sources than is otherwise possible with only static constructs. The functions included in the language each contribute to a system that exceeds the general purpose of exchanging information. Each function is capable of acting on statically defined values and variables to increase usability. The following describes the goals and functionality associated with each operation.

- **Filter:** Filter functions allow the selection of only desired information while ignoring information that is not of interest. The first parameter of a filter function has the potential to be another filter statement, providing a recursive nature to the expression, enabling complex queries to be described.

  - *filter (bob > 10)*

- **Vouch:** A vouch request enables reputation sources to leverage other's knowledge of an entity. Requests are made when one source does not have experience with an entity and wishes to gain information from another source. To enable the complete interaction required for this filter, a potential response can be incorporated into the function as a boolean value.

  - *vouch (bob for alice {: {response} })*

- **Assert:** An assert statement acts to ensure a requirement is true. The goal of an assertion is to ensure a condition holds between reputation sources.
  - *assert (bob in group trusted)*
- **Event:** An event statement allows reputation sources to place a condition on some information. With the incorporation of rules and policies, events provide the functionality to specify when a rule should be executed. In this example, the consumer is expected to execute the rule *markUntrusted* if the condition is met.
  - *event bobEvent = (bob < 10 : markUntrusted)*
- **Social:** Social functions support the aggregation and filtering of social values. The majority of the functions accept two *socialValues* as parameters. The social functions provide the necessary functionality to implement the social network case study, allowing information to be manipulated and queried within the service.
  - *add (tag {[bob], [alice]}, tag {[bob], [alice], 5]})*

6.3 Conclusion

This chapter has presented the domain specific reputation language designed to describe a diverse range of reputation information. A domain specific language is used in the description and interpretation system rather than a static schema to increase the usability, maintainability and descriptiveness required when dealing with reputation information. After defining the fundamental aspects of the language, such as the syntax and type enforcement methods, the specific types associated with reputation were derived.

Reputation types have been identified through an extensive review of existing reputation systems and the case studies. During the investigation of the case studies, domain ontologies were created to identify common reputation types and potential relationships. The construction of the language uses ANTLR, a powerful tool used to generate parsers and lexers. A discussion of necessary reputation types, including primitive, complex and domain specific types, was presented as well as reasoning as to their inclusion. A set of utility functions, designed to increase the usability of the language and overcome the limitations of a strictly descriptive language, are described.

The reputation language is a suitable mechanism to express reputation information that exceeds the requirements specified in Chapter 4. Through the use of the described reputation language, reputation systems will have a suitable medium to communicate with one another in a meaningful way.

Chapter 7

Verification System

The verification system is designed to interpret and ensure the language, presented in Chapter 6, is used correctly across nodes in the OpenRep architecture, in particular it enforces syntactic and semantic constraints. The verification system is a component of the description and interpretation system, designed to be included within the *composition* layer of the OpenRep stack. The verification system is composed of three key elements; the interface, language parsing unit and type enforcement system. Combined, these three modules create a functional solution to verify the language is used correctly.

The language parsing aspect of the verification process relies on the parser and lexer rules that have been generated from the development of the language. The parser and lexer implementation allows arbitrary input to be parsed into tree structures and processed for errors. Invalid input that cannot be interpreted will result in program level exceptions and will not be executed.

Type enforcement is responsible for ensuring semantic errors are identified. Invalid type use, such as performing an arithmetic operation on two string types, is considered unsafe and must notify the verification system of invalid input. The language is characterised as statically checked and strongly typed, meaning type checking is performed during the compilation of the input. Static checking also provides some advantages over dynamic checking, for example, it is possible to identify certain class errors and improves the readability of instances of the language [39].

The design of the verification system, as stated in Chapter 4 focuses on the internal requirements and utilisation of these two validation elements. The major goals to meet these requirements are the suitable compatibility checks, establishing what constitutes a type error and the definition of a type hierarchy.

7.1 Design

The three design requirements for the verification system are that it 1) provides a functional mechanism to validate proper syntactical use of the language, 2) enforces static type checks and 3) is capable of returning feedback to the caller through an event based interface. To meet these goals, the verification system is designed as three distinct modules. These three modules are packaged as a single usable verification service to simplify distribution.

7.1.1 Interface

The interface is responsible for abstracting the complexity of the verification system from the user. The interface encapsulates the invocation of the verification mechanisms and handles the marshalling of information between the underlying modules. The interface is also designed to provide the functionality for users to subscribe to notifications through listening lists, for example to retrieve output asynchronously. Due to the event based nature of the language, the compilation and execution of the input raises events based on the outcome. The outcome of the verification is designed to be represented by a Java object to more easily integrate with OpenRep, however, the representation could be adapted to any high level programming language. The event carries the outcome of the verification systems invocation as a payload, detailing either the errors that were found, or describing the information that was encoded in the language.

7.1.2 Language Parsing

The validation module of the system establishes whether or not the input is well formed and is a valid instance of the language. To accomplish this task, the lexer is executed with the input text to generate a string of tokens. If phrases are found that cannot be tokenised, an exception is raised to warn of potentially invalid input. The token stream can then be passed to the parser to match tokens to rules. An AST is created to statically check the types of input and their use. The language validation feature is also designed to provide the utility functions on the AST. Functions regarding the traversal of the tree, retrieving specific elements, function parameters or establishing the types of input and their associated values are the responsibility of this module.

7.1.3 Type Enforcement

The static type enforcement module is designed to ensure type errors are identified and the system is notified before execution. Many lessons can be learned from other type systems in the design of the type enforcement component. For example, The Xtext Type System [18] provides the functionality to test a range of instances, such as the proper use of functions and the compatibility of provided parameters. The evaluation of the Xtext Type System and the type systems from languages such as Java and Pascal, has identified the necessary capabilities of the verification system:

- Evaluating type equality.
- Determining type compatibility through hierarchical rules.

- Ensuring type consistency in structures.
- Enforce correct function parameters.
- Manage correct operator use.

7.1.4 Type Errors

For the verification system to successfully identify type errors, the definition of a type error must first be established. In this context, a type error is the invalid use of data types. With regard to the reputation language, two forms of type error have been addressed. Both of these errors must be identified by the verification system for it to validate input.

- **Incompatible Values:** The declaration of information must be checked for compatible types. When describing primitive types, the keyword prefix must be checked for compatibility with the following value. Similarly, complex structures require the validation of declared types with contained information. For example, an *array* with a specified *int* type cannot contain information that is not compatible with an integer.
- **Incompatible Arguments:** Invalid operator and function type compatibility, such as using a multiplication operator with *string* values are required to be identified. Function signature compliance, where the correct number and type of arguments are ensured, is also important when verifying the correct semantic use of the language.

7.1.5 Type Hierarchy

The reputation language supports a type hierarchy, in which information types can be substituted with one another if they are determined to be compatible. Hierarchical rules provide a mechanism to incorporate inheritance into the language without supporting class structures and customisable polymorphism. An example of the type hierarchy can be demonstrated with the use of identification types within the language. Because the language supports reputation information that not only represents individuals, but also *groups* and *roles*, an *identifier* type has been defined as being compatible with each of these subtypes. When establishing whether an input is valid, the verification system consults policy files to determine the required signature of a function. Once the required types of a function are identified, the types provided in the input are evaluated. If a match is not found, a policy file of type hierarchy rules is recursively searched to establish compatibility. The following gives a brief example of the hierarchy rules present in the reputation language.

- socialType: like, comment, post, tag, literal
- socialId: socialName
- id: user, group, role, socialId
- value: id, literal, null
- literal: char, int, float, double, string, bool
- string: char
- float: int
- double: int

7.2 Implementation

The verification system is implemented as a standalone Java service composed of three distinct verification modules. The interface allows users to invoke the functionality of the verification system and receive feedback by subscribing and listening for notifications and events. The language validation unit incorporates the lexer and parser, defined in Chapter 6, to process input instances of the language into ASTs. The static type enforcement system checks the AST to ensure the correct use of the language. Output is provided through the construction of a response object and is returned through the interface. Figure 7.1 describes the process of parsing an instance of the reputation language through the verification system.

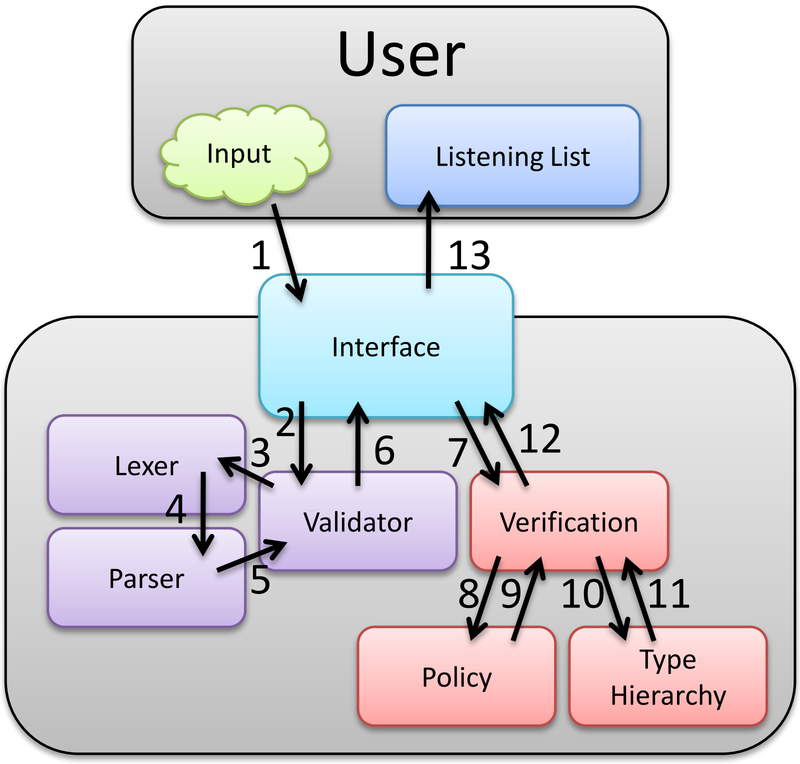


Figure 7.1: The flow of information when parsing input. Input originates from a user and is passed to the interface. The interface uses the validation unit to process the lexer and parser rules. The resulting AST is then verified to identify semantic errors.

7.2.1 Interface

The interface allows users to subscribe to notifications and request verification of input. The language provides an entry point to the verification system and is responsible for marshalling information between the modules to perform the appropriate checks on the input. The input is first provided to the validation unit, constructing an AST if the language can be properly tokenized and processed by the parser. The AST is then passed to the validation system, where the static type checking rules are executed. Depending on the output of the checks, an appropriate response object is created. The response object can contain information representing different error states, or, in the case of successful verification it represents the parsed information. The response object is then packaged as a payload with identification information and stored into an event object. The event is then fired on the respective notification list.

7.2.2 Language Parsing

The language parsing unit employs the parser and lexer rules that have been created with the definition of the reputation language. The validator is responsible for syntactically checking the instance of the reputation language. If the lexer or parser fail to evaluate the input correctly, an error message is returned to the user. The validator also provides support functionality for processing the AST that is created. Because the AST structure can be specified within the language grammar, the process of traversing the tree to extract information must be customised. Functions supplying the ability to identify types of information, retrieve parameters and evaluate the operators being invoked, are defined within the validation unit.

7.2.3 Type Enforcement

The core of the type enforcement system contains the functionality to examine ASTs and compare input against policies. Policy files specify the capabilities of functions and determine what constitutes proper use. The policy files also contain the type hierarchy, describing the order of types, allowing the system to examine whether the target types are compatible with the given input.

The fundamental task of the type enforcement system is the comparison of types. In order to identify the type errors presented in Section 7.1.4, the ability to derive the compatibility of two types is essential. The verification system includes functions to evaluate equality, compatibility and the subtype structure of types. This has been implemented in Java as string based equality checks on type names and catching execution errors when parsing values to the described type.

Verification of value assignment typically relies on analysis of the AST. When the AST contains information about a type and value, the type enforcement system ensures the type and value are compatible. The verification system includes assignment verification mechanisms to ensure each of the complex information types defined in Chapter 6 are used correctly. Enumerated values and Star ratings include additional policies to ensure the selected value is contained within the set or range. Collections, such as the Array, have their respective types stored and evaluated against the data they contain. Operators and functions are included in a policy file stating the highest level parameters they are applicable for. Through the inclusion of the type hierarchy, input can be recursively checked for type compatibility to verify the input is appropriate. Domain specific information, regarding the social network based case study is also included in the policy files.

7.3 Summary

The verification system is designed to enforce the characteristics of a strongly typed, statically checked language. It is designed as three individual modules and has been packaged into a portable system, capable of being integrated within the OpenRep stack to meet the requirements.

The interface establishes the event based nature of processing the reputation language, raising events dependent on the evaluation of the input. The validation unit has the capability to ensure instances of the language are syntactically correct. Using the parser and lexer, created during the expression of the reputation language, input is processed into abstract syntax trees. The type enforcement system is capable of evaluating ASTs to identify type errors. Through the implementation of a variety of type compatibility checks and policy files that determine the usability of structures and functions, the type enforcement system is capable of recognising semantic errors and enforcing constraints.

The implementation of the verification system represents a usable system, capable of processing the reputation language. Type errors are identified, providing the ability to reliably exchange reputation information. The verification system is an effective mechanism to control the language and ensure it is used correctly. The verification system is implemented as a standalone service with an intuitive interface, capable of functioning within the OpenRep composition layer.

Chapter 8

Case Study Implementation

The three case studies, presented in Chapter 4, have been implemented to represent a diverse range of potential scenarios and to also demonstrate the capabilities of the description and interpretation system. The case studies, electronic auction site, social network and virtual community, each represent a functional reputation system within their respective domain with the ability to distribute information through OpenRep's standardised service-based interface. The case studies each provide a unique perspective of reputation information, serving as an ideal test bed to demonstrate the capabilities of the description and interpretation system to federate diverse reputation systems.

Each implementation is implemented as an OpenRep node and together form a federated reputation environment. Both the social network and virtual community case studies represent source nodes within the network, where the *collection* and *composition* layers of the OpenRep stack are implemented. Source nodes provide the ability to collect information from entities within their domain and distribute it to others in the environment. The auction based implementation performs as a both a source and agent node, requiring all three levels of the OpenRep stack. Because the auction case study includes the *interpretation* layer, it is capable of high level decision making.

Each case study exposes a standardised REST interface that is both interoperable and modular to enable the distribution of the implementations through the network. The implementation also contains a local set of data regarding users and their associated reputation scores. The description and interpretation system has been integrated in each implementation, utilising the reputation language, verification system and exchange protocols.

To further demonstrate the federation of reputation information, the auction study has been implemented as a manager to coordinate reputation exchange between all case study nodes operating within the environment. The auction implementation is capable of actively invoking all forms of request upon other services within the environment as well as responding to incoming requests.

Due to the fact that reputation information is highly context dependent, policies regarding the interpretation of information and specifying the relative weights associated with various types during aggregation are required. In order for the auction case study to perform as an agent node, capable of aggregating and making decisions on the information received, an extensible policy architecture has been used to further test the framework.

Figure 8.1 depicts a potential topology of a reputation environment. As shown in the figure, multiple reputation sources can exist simultaneously and interact with one another. Requests are made to different services in the environment and are processed through the reputation description and interpretation system. The reputation protocol is used to exchange information between the services and policies are used to act upon this information to base decisions.

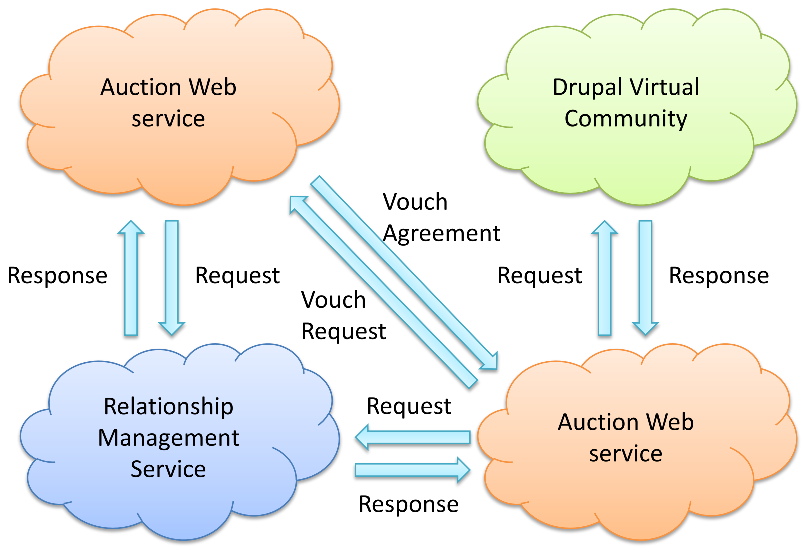


Figure 8.1: An example reputation environment with two auction services deployed and interacting with the social and virtual community services.

8.1 Auction

The electronic auction case study is the primary resource in the reputation environment. Rather than only servicing requests for information, and acting as a standalone service, the auction service is capable of requesting information from other services in the environment to form an aggregated view. Multiple instances of the scenario can be deployed simultaneously, creating a rich reputation environment with multiple requests being made concurrently. Rather than using a complex universal identity model, users are identified by their user name, this is sufficient to provide consistency between reputation sources. In a production setting, a more complex universal identity model would be used, however the interactions between entities, from the perspective of the reputation system, would be identical. The auction service is unique in that it describes and supports the requirement regarding customisable policies for reputation consumers, described in Chapter 4. These policies provide the auction services with the ability to process reputation information from external sources and appropriately aggregate it with its own.

The auction case study also demonstrates the vouching scenario, that is, reputation sources are able to vouch for users within the environment. Figure 8.2 shows the simplified process involved in accomplishing a successful vouch exchange, real world scenarios require a search for potential vouchers and conditions to be checked before agreeing to vouch.

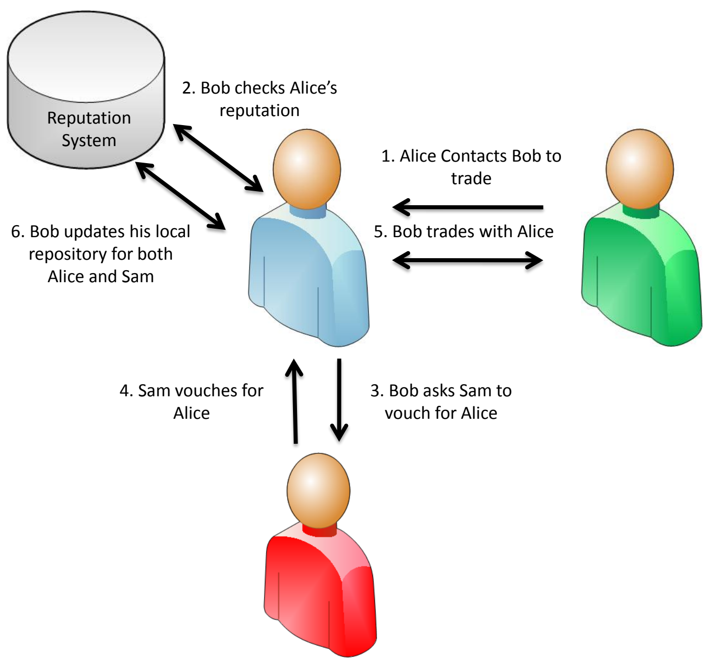

Reputation System
2. Bob checks Alice's reputation
1. Alice Contacts Bob to trade
5. Bob trades with Alice
6. Bob updates his local repository for both Alice and Sam
4. Sam vouches for Alice
3. Bob asks Sam to vouch for Alice

Figure 8.2: A vouch scenario. The users Bob and Alice have no prior knowledge of one another. Bob asks Sam to vouch for Alice, then proceeds to trade with Alice, leveraging Sam's trust.

8.1.1 Implementation

The auction scenario is designed to run a simulation of the different case study implementations interacting.

The auction service is designed to operate as an electronic auction, similar to eBay or TradeMe. The auction consists of a set of users with associated trades between one another and feedback regarding transactions. User and transaction information is stored in a service specific database. In order to provide operations that filter potential traders based on previous feedback, information associated with item type and price is also included.

The simulation runs by randomly generating a potential trade and sending a request to one of the other services deployed in the environment. The primary task of each service is to respond to requests using the exchange protocol defined in Chapter 5. With the interface and exchange method predetermined, information associated with identity, mode, role and transactions are specified through an HTTP GET request to a service. Responses and more detailed requests can be sent through the HTTP POST and PUT request structure.

The case study is implemented as a RESTful Jersey Web service. Jersey [55] is an open source API for developing REST web services. The service framework was created and managed using the java project management tool Maven [14]. Maven allows for the efficient creation, management and deployment of the service to a specified container. Through the inclusion of the *maven-jetty-jersey plug-in* in the project object model, Maven can be used to rapidly deploy the service to a local Jetty[1] container.

Before determining the most suitable applications and frameworks for this service several alternatives were considered. Restlet[2], Axis2 [53] and WSRF [16] are all potential frameworks in which to implement the services. Jersey has been chosen as it provides a clean set of APIs and anno-

[1]http://jetty.codehaus.org/jetty/
[2]http://www.restlet.org/

tations used with Java to create services. Jetty was selected over Apache Tomcat [48] and Grizzly[3] as it is a light weight servlet container that can easily be integrated with Jersey through Maven. The combination of these tools enables quick and easy creation of services.

To function as a reputation source, the reputation information regarding users must be stored and persisted throughout the experiment. An embedded Apache Derby[4] database is used to store the necessary information for the service. An embedded database was chosen over MySQL[5] or PostgreSQL[6] to ease the build up and tear down cost of deploying multiple services. Derby was selected over SQLite[7] and Hibernate[8] as it is a lightweight database and has a small memory footprint. Derby proved to be more than adequate for what was being created. It supports efficient creation and population of a new database every time the service is deployed and has been shown to perform well under all the scenarios considered. In order to populate the database, the service is started with a random seed value that allows the simulation to be repeatable for testing purposes and yet enable distinct instances of the service to generate unique data.

To generically exchange information, the reputation exchange protocol, presented in Chapter 5 is used. The exchange protocol encapsulates each interaction between the services in the environment. In order to simplify the creation and parsing of exchange protocol transactions, the Java Architecture for XML Binding (JAXB) [19] is used. JAXB simplifies the process of interacting with XML, or JSON [15], within Java. Rather than requiring manual parsing through XML with DOM [29] or SAX [44], JAXB automates the creation of Java objects to represent XML schema defini-

[3]http://grizzly.dev.java.net/
[4]http://db.apache.org/derby/
[5]http://www.mysql.com/
[6]http://www.postgresql.org/
[7]http://www.sqlite.org/
[8]http://www.hibernate.org/

tions, such as those created for the exchange protocol. Marshalling XML to and from the Java classes is therefore automated and provides a fast and effective mechanism to interpret and create XML representations of the data.

Customisable policies are necessary to enable the interpretation of information both internally and also from external sources. After much investigation into different policy and rule languages, Drools [7] was identified as the best solution to provide a dynamic and extensible policy language. Drools provides an integrated platform for rules, workflow and event processing in Java. Drools rules are highly customisable while being easy to understand and create. The rules defined for the auction scenario allow the description of fine grained policy, enabling such functionality as the aggregation of reputation information and vouching requests to be processed. The rules currently determine the weights assigned to different types of information for aggregation calculations. They also specify the conditions under which it is acceptable for the system to vouch for another user.

8.2 Social Network

The social network case study is designed to show the potential diversity of the type system and reputation language. The prototype application is a relationship management service, which categorises social network friends into groups based on the level of interaction between them and the user in question. This application is intended to collect different forms of information from the environment, express the information and then process it. The application also exposes a REST interface in line with the OpenRep standard to facilitate interactions with other reputation systems in the environment.

The relationship management application is implemented as a Facebook application as Facebook is the predominant social network in use worldwide, with over 800 million active users[9], it also provides a thorough platform to collect information on social interactions between users.

The relationship management application is able to extract different forms of information about users and their relationships, from the Facebook Graph API[10], interpret it against a set of policies and store it with relation to a particular user. Currently, the information collected is based on the direct interactions between the logged in user and the users contained in their friends list, such as both being tagged in a photograph and posting on each other's walls. The application could be extended to monitor the interactions between friends of friends as well, which would provide more complex reputation information from which decisions could be made.

8.2.1 Implementation

The core of the relationship management application is responsible for authenticating the Facebook application's information, extracting user and relationship information from the Graph API and processing the JSON response. The inclusion of the description and interpretation system into the application allows the information to be expressed and processed. Once the information can be described in the reputation language, it can be verified and used to make decisions. Different reputation types can be assigned to the information, allowing the information to be processed, stored and shared.

In order for the application to be accessible through Facebook it must be hosted externally. While this can be done using any third party hosting provider, we chose Google's App Engine [13] as it provides a scalable and highly available platform. Finally to provide transparent integration with Facebook the case study application also includes a Facebook interface which can be incorporated as a canvas for a Facebook application page.

[9]http://www.facebook.com/press/info.php?statistics

[10]https://developers.facebook.com/docs/reference/api/

This canvas interface provides a simple user abstraction of the application functionality including buttons to load the information about a user from the Graph API, sort it and facilitate sandbox tests. The canvas interface is built upon the underlying REST interface that supports the OpenRep standard and enables other services to request information for specific users.

The friendship management social application is hosted on Google's App Engine. The Google development tools support local development and testing of the service on a standalone Jetty server and simplify deployment to the cloud platform. The service is implemented as a RemoteServiceServlet, essentially an RPC servlet, that provides a full HTML user interface to the application. The Servlet also allows HTTP request handlers to be overwritten, such that it can support the standard REST interface that has required for reputation exchange in our model.

The service was constructed with the use of the Google Widget Tools (GWT)[11] library. The GWT include a plug-in for Eclipse[12], meaning the service can be created, tested locally and deployed to the Google App Engine, through the GUI interface of Eclipse. GSON[13] is used to efficiently convert the JSON response from Facebook into a set of Java classes that can be used to interpret information.

For the service to collect information from Facebook's Graph API, the application and user must first authenticate themselves. Facebook uses the OAuth 2.0 protocol [25] for authentication and authorisation. The authentication process requires a three step interaction between the application service and Facebook. Before retrieving a valid access token, the user is redirected to Facebook to enter their credentials and grant access to the application.

The Facebook Graph API provides access to a large amount of information regarding the authenticated user and their relationships to other

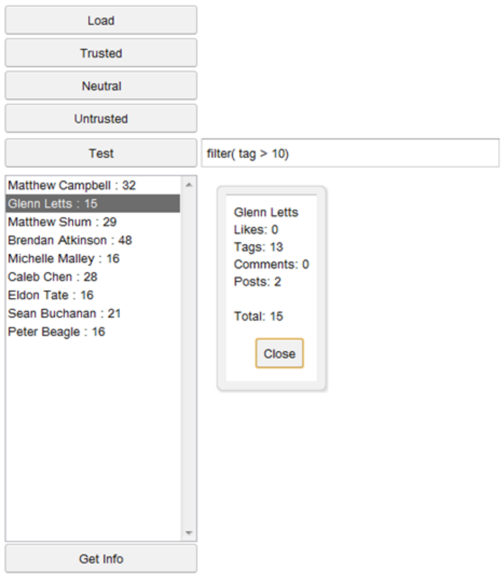
[11]http://code.google.com/webtoolkit/

[12]http://www.eclipse.org/

[13]http://code.google.com/p/google-gson/

objects, for instance friends, photos, feeds, and posts. In order to perform the request, the access token received from the OAuth authentication process is attached to a GET request specifying the information required, such as the feed and fields to return, to the Facebook Graph API. Once a JSON object is returned to the service, GSON is used to create Java classes from input.

The resulting GSON classes can then be expressed as reputation information in the reputation language. The verification system is then used to evaluate the input and determine how to process it. The response from the verification system is extracted from the resulting event and processed.



filter(tag > 10)

Matthew Campbell : 32
Glenn Letts : 15
Matthew Shum : 29
Brendan Atkinson : 48
Michelle Malley : 16
Caleb Chen : 28
Eldon Tate : 16
Sean Buchanan : 21
Peter Beagle : 16

Glenn Letts
Likes: 0
Tags: 13
Comments: 0
Posts: 2

Total: 15

Figure 8.3: The relationship management service user interface.

As shown in Figure 8.3, the user interface consists of five main buttons and a text box. The first button, *Load*, performs the requests to the Graph API. Once load has run, the other buttons become available, allowing the user to access a list of users in each category. The trusted, neutral and untrusted buttons each send a request to the service to perform a *filter* function, encoded in the reputation language, over the information. The test button has been included to allow input to be sent directly to the service's verification system. Once the list box has been populated with information from one of the above buttons, a friend can be selected to inspect. The inspected users information is then shown in a dialog popup box. Any error messages are prompted to the user in a window alert screen.

The design and implementation of this case study led to the identification and definition of the set of social types presented in Chapter 6. For example, information relating to users being tagged in photos together, commenting on each other's walls, the amount of times they have 'liked' a post the user has made and the number of times posted on the users wall are each collected and used to make reputation based decisions.

The service is publically available for installation and use within Facebook to categorise their relationships. The service is currently hosted on Google App Engine.

8.3 Virtual Community

The virtual community scenario is based on a common reputation context domain, and represents a further analysis of the type of reputation information currently available. The scenario represents a diverse range of reputation systems from online communities, for example forums, chat rooms, review based websites and news aggregators, such as Reddit, Digg, Slashdot and Epinion. Virtual communities exist for a vast range of topics and often support a mechanism to "rate" comments and users. Although there are wide variety of online communities available, they essentially all provide similar functionalities, that is, a mechanism for users to share information between one another. For this reason, this scenario is implemented as the most generic case of virtual community: an online forum. Online forums are a standard means of community interaction on the internet and as such there are a large number of freely available forum implementations that can be used to bootstrap the implementation of the scenario and test the portability of the system.

8.3.1 Implementation

The existing service used in the virtual community implementation is a Drupal[14] content management system with an embedded forum module. The forum module is capable of generating and storing reputation information about users. The service also provides the capability to receive and respond to requests through the OpenRep REST interface. However, the initial Drupal module did not support the use of the exchange protocol or reputation language developed in this thesis. For the existing service to be integrated into the reputation environment, the description and interpretation system must be present in conjunction with the service using the exchange protocol.

The design of the service requires the packaging of the description system and exchange protocol as a tool. To integrate the description system and exchange protocol into the Drupal client, the client must invoke the packaged tool before responding to a request. Once a request is made, the client retrieves the user information, expresses it within the reputation language and provides it to the packaged system. The packaged system responds with an encoded XML file containing the verified expression of the reputation information. Reponses from the packaged system should be adequately tested for errors raised by the client.

To achieve this integration the description and interpretation java application is invoked directly from the PHP [6] forum service before sending the response to a request. The implementation processes requests made to the service for reputation information regarding users.

For this scenario the description and interpretation system is packaged

[14]http://drupal.org/

with the exchange protocol as a standalone Java application. A handler interface automates the process of marshalling information from the description system to the exchange protocol wrappers. To use the packaged system, the Java program must be called with a complete set of parameters describing each of the description systems' and exchange protocols' required information. The package includes and utilises all of the description system and exchange protocol, employing JAXB to encode the response. The packaged application returns an XML response in which the reputation information is encoded, provided it was verified correctly.

The service is capable of responding to requests for information only about users. Through the invocation of the Java program, a full set of parameters enable the expression, verification and encoding of the reputation information. The result is supplied as a valid instance of the reputation exchange protocol, enabling others to request information and interpret it. The information reflects the interactions between users within the Drupal forum client.

This virtual community case study has motivated the examination of many reputation systems currently in use throughout the Internet. The inclusion of this case study shows the potential reach of a federated reputation system.

8.4 Summary

The three case studies have each been implemented as a service, representing an OpenRep node, and deployed in each domain in order to demonstrate the capabilities of the reputation architecture. The case studies have each provided invaluable insight and direction throughout the investigation, design and implementation of the reputation language, verification system, and exchange protocol. Each of the case studies is uniquely implemented to test the potential of such a description and interpretation system to be used across various domains.

In order to evaluate the exchange architecture a reputation simulation environment has been created that is able to communicate with implementations of each of the scenarios and perform complex actions, such as vouching for users that were previously unknown to a particular reputation system. Reputation information can be generated by, and exchanged between, individual services in the environment. Through the use of these diverse scenarios, the viability of the reputation description and interpretation system can be demonstrated. The language is able to express reputation information from a variety of sources and contexts, and the verification system enforces static type checks over the expressed information, raising errors when necessary. The case studies have also proven the effectiveness of the reputation exchange protocol and the standardised OpenRep interface, allowing a new level of interoperability between diverse reputation sources.

The implementation of the case studies illustrates the potential of a federated reputation system. The diversity of the case studies considered exemplifies the possibility of sharing reputation information from many different domains. Through the combination of a global identity management system and the participation of potential reputation systems, a sea of reputation information could be made available to users and reputation systems alike.

Chapter 9

Evaluation

The research presented in this thesis covers a broad range of areas which each require individual evaluation. This chapter presents analysis of each section of the work individually before reviewing the language, verification system and exchange protocol as a whole. Section 9.1 reviews the criteria and reasoning behind each of the core case studies with regard to the benefit they have provided and their capability to perform as OpenRep nodes. The language is then examined in Section 9.2, in particular this section evaluates how well the system meets the stated requirements and design goals presented in Chapter 4. Section 9.3 presents a evaluation of the verification system with respect to its ability to enforce the syntactic and semantic constraints of the language. The exchange protocol is empirically evaluated in Section 9.4, as part of this evaluation the exchange protocol is compared with existing protocols. The testing procedure is discussed in Section 9.5, explaining the testing strategy and results for each of the components of the system. Finally, Section 9.6 reviews the entire project to determine how well it enables the expression and interpretation of reputation information in the OpenRep architecture. Throughout this chapter components are analyzed with respect to the requirements presented in Chapter 4 and existing solutions for similar problems.

9.1 Case Studies

The three case studies described in Chapter 4 represent a diverse set of reputation environments derived from extensive analysis of both commercial and academic reputation systems. The case studies were designed to best complement each other by presenting three unique views of reputation information based on reputation systems in use today. The goal of defining and implementing these case studies was threefold: derive requirements for each component, direct the design and implementation of the reputation services with real world use cases, and to act as a tool against which the different components can be evaluated. The three case studies have each been implemented as services that are independently accessible as well as interoperable within the wider reputation environment.

9.1.1 Auction

The auction case study was selected for its ability to relate to leading reputation systems in real world applications, such as eBay and TradeMe. The online auction domain is arguably the most common use of reputation in real world activities, and therefore it is easy to relate to and provides a degree of familiarity for readers interpreting this research. Through an investigation of electronic auction sites, the concept of collections of reputation types was evident. Primarily, electronic markets keep relatively similar forms of reputation information regarding traders and their interactions. Typically, the information is stored in numerical form, for example counting the number of trades that were successful and unsuccessful. The development of ontologies have proven to be a necessity when establishing the requirements of the terms that must be expressed. The action ontology identified the clear relationships between information, for example boolean or enumerated star values that are often associated with textual information in the form of comments.

9.1.2 Social Network

The goal of the social network based case study was to create a functional social service and extract reputation information from a social context. This case study represents a novel use of reputation in the ever expanding world of social networking. The implementation of the case study focuses on the design and creation of a friendship management service in which interactions between users in a social network and categorises the set of participants into groups based on their derived trustworthiness.

The social network case study gives a unique perspective on what constitutes reputation information and how it can be expressed. From more common reputation sources, such as auctions and news aggregation web sites, reputation types are often expressed in generic forms, such as numerical or textual information. The inclusion and typing of various interactions between users, such as being included in the same photograph, increases the robustness of the system and provides an insight into future possibilities for reputation collection while also demonstrating the extensibility of the reputation description system and its potential to be applied to sources that may otherwise have been excluded. The case study highlights the wide variety of information that can be used to derive reputation information and ensures a level of future proofing for this work

9.1.3 Virtual Community

The virtual community case study was developed based on investigation of many main stream reputation sources, such as discussion boards and social news websites. The virtual community example represents a significant portion of reputation systems currently in use. Many web sites utilise reputation systems in order to prioritise information for their users. Through the inclusion of the virtual community study, a wide range of reputation systems can be examined in detail. The concept of including reputation information from sources that often feature reputation systems as an unessential measure of popularity, contrasts the general perception of reputation. Creating a federated reputation system, such as OpenRep, would therefore enable a vast number of such sources to contribute information that is otherwise redundant.

9.1.4 Comparison

Each of the case studies provides a unique view of potential reputation systems and generates requirements for the development of the description and interpretation system. They have also facilitated the development of the first three distinct services to act as OpenRep nodes, allowing evaluation of the performance of the system under different circumstances. The vastly different methods of implementation, using a Jetty WebApp, Cloud based Facebook application and CMS plugin, demonstrate the ability of the description system to be implemented over a range of potential sources.

Overhead

Each of the implementations demonstrate a real world environment in which a reputation system operates. Figure 9.1 shows the overhead of using the verification system within each case study when various forms of input are processed. The measurements are taken before invoking the verification system and after the notification is returned to the caller. Each of the systems is invoked with the same set of language instances and the processing time is measured. The figure shows the Google app engine performs well until it is required to load and process policy files, this is due to the bandwidth and latency limitations. The difference between the remote service based auction implementation and the locally packaged community system highlights the additional overhead associated with service execution. The figure demonstrates that the overhead of invoking the verification system to process and interpret instances of the reputation language is minimal compared to that of parsing a file. From this observation, the overhead of verification system can be given context and considered to be light weight and having little effect on an OpenRep node's performance.

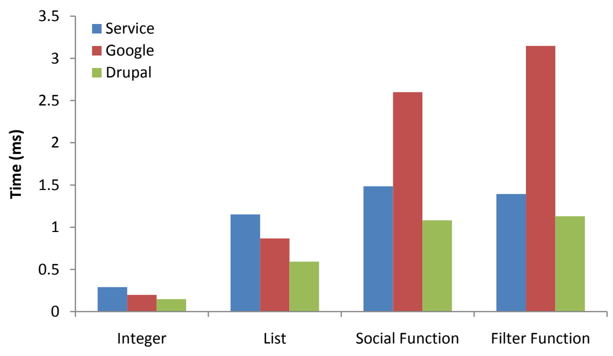


Figure 9.1: The overhead of the verification system processing an integer declaration, list structure, social addition function and a filter function under the three case study implementations.

Each of the services functions with data specific to their environment. In order to demonstrate the capability of the OpenRep system to scale, filter functions have been tested with datasets relating to each case study domain. Figure 9.2 shows the time taken to filter a particular number of reputation entries from a set of 5000. The test uses integer values to represent auction information, textual information corresponding to the data ranges utilised by Slashdot and social structures with associated integer values. Tests have been run locally to remove the effect of latency. As shown in the figure, the filtering functions scale linearly, with little difference between the form of filtering. This demonstrates the capability of the system to perform reliably as a datasets size increases.

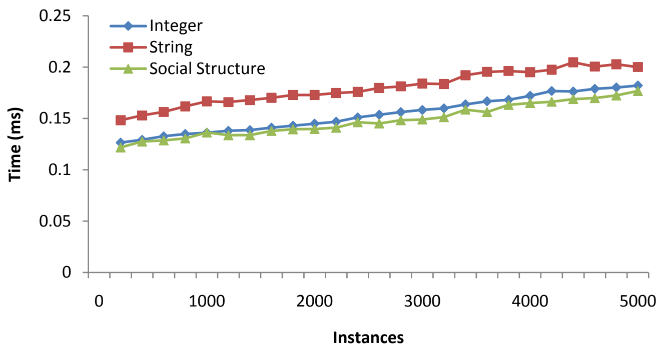


Figure 9.2: The time taken to filter reputation information. Types of information relating to each case study are filtered for particular ranges.

9.2 Reputation Language

The core requirements for the reputation language are to be able to accurately express a wide range of reputation information and to also provide a standard method of communication. The process of defining a domain specific language for reputation was selected over the definition of reputation specific XML structures to enhance the maintainability and portability of information. Due to the expressive requirements of describing and enforcing the typed communication of reputation information and actions, a language was the best suited solution.

Evaluation of a domain specific language is a difficult task as it can be conducted in many dimensions. In order to evaluate the effectiveness of the proposed reputation language a range of metrics such as expressiveness, readability, portability and consistency have been derived from the design principles described in Chapter 4. The following section explores these metrics and their relation to the construction of the language.

- **Expressiveness:** The language supports a standard set of primitive types, such as integers, booleans, characters, strings and floating point numbers. Reputation information is rarely displayed and consumed in its primitive state, rather collections of information are provided to users. Collections, such as lists or enumerated types, are used to represent a set of discrete values, such as trust levels ranging from terrible to excellent. Through the use of primitive types and collections, all forms of reputation information can be described, meaning that the language is able to express reputation information in the domains considered in this thesis.

- **Readability:** The syntax of the language is based on a combination of existing programming languages. The primary goal regarding the syntax was to allow the language to be intuitive to use and interpret. Through the use of common syntactic features and key words from many prominent languages and a well defined set of operators, the reputation language is neither verbose nor is it complicated, for these reasons the language is considered easy to read and understand.

- **Writability:** The cost associated with learning a language is often considered the biggest limitations of a domain specific language. However, leveraging common syntax and structure from other languages reduces the burden to learn how to use the reputation language. In addition the simplistic nature of the language allows even non-technical users the ability to quickly express reputation information.

- **Consistency:** The language has been revised multiple times throughout development to ensure consistency between expressing reputation information of different types. Reviewing the use of different parentheses and key words has established a distinctive syntax for the language, while being consistent with commonly used notation. The syntax is regular throughout the language, allowing function and type definition to be intuitive.

- **Operation Support:** The reputation language includes a full set of functions that support typical reputation operations performed by users, such as filtering. The functions provide the ability to invoke operations on information contained by other reputation sources and request specific content regarding users. Interactions, such as vouching, have not previously been supported in reputation architectures and as such the development of this language represents a novel contribution. The ability to perform these actions has extended the usability of the language and demonstrates a new area of potential for reputation system cooperation.

- **Cost:** The cost associated with expressing, maintaining and executing the language is closely related to that of the verification system. Once the cost of learning the language has been achieved, the time required to maintain information described with the language is minimal. The overhead of using the language is discussed further in Section 9.3.

- **Portability:** The portability of the language was paramount during its creation. Due to the nature of the description and interpretation system, the language must be consistently used and interpreted in a variety of domains. A large element of the portability of the language is reliant on a standard interpretation unit. This has been achieved through the definition of a verification system, capable of consistent interpretation across reputation systems.

9.3 Verification System

The fundamental requirements of the verification system are to impose semantic and syntactic validation checks over the language to identify errors and invalid instances. Customisable policies regarding the execution errors are also necessary.

The design goals of the verification system relate to the functionality required to enforce static type checks. The verification system has been implemented as a Java program, responsible for enforcing the correct use of the reputation language. A type hierarchy has been integrated within the language to enable compatible types to operate transparently. The static type checking process incorporates policy files to determine the hierarchy of types when verifying the language. The language parsing feature manages to properly identify and deal with invalid instances of input. The type errors identified in Chapter 8 are correctly recognized as being invalid and appropriate exceptions are raised.

During the implementation of the component, the requirement to provide asynchronous notifications through an event based feedback mechanism was explored. To create an event based system, both an interface that could facilitate the interactions and the events themselves was designed and implemented. The interface provides the ability to register for a variety of notifications and define listeners that will be alerted. The feedback itself is contained as the payload of the event.

9.3.1 Usability

The usability of the verification system has been considered extensively during the development process. The system can be invoked in two ways, either through individualised calls to the separate functions, openly provided through each of the elements, or through the interpreters ReST API. The interface simplifies the procedure of utilising the verification system by automating the marshalling of information between verification steps.

Error messages regarding unsafe instances are returned to the user with a status code and message. Errors range in severity from warning level messages to critical problems. Warnings typically notify the user of type mismatches, for example if two different types of numerical value are being used together, though the type checker established they were compatible. Critical errors result from invalid types of information being used together when there is no supporting policy stating their compatibility.

9.4 Reputation Exchange Protocol

The reputation exchange protocol is designed to facilitate the transfer of reputation information between reputation sources. The protocol is composed of two separate parts expressed as XML schema definitions. The Reputation Exchange (RX) schema encodes reputation elements specific to a single identity. The Reputation Container (RC) schema maintains data integrity, encapsulates RX information and serves as an envelope describing the payload.

The protocol is designed to meet the requirements presented in Chapter 4. The ability to encode information without loss, data integrity and being able to describe associated information for the source and target are necessary. The following section investigates the creation of the protocols and evaluates their ability to meet these requirements. Metrics associated with these conditions, as well as parsability and cost are considered. The protocol has also been reviewed with regard to existing protocols and definition languages.

To include the XML schemas into the scenarios and test them practically, the process associated with creating and parsing XML elements with DOM or SAX would be tedious. Instead, JAXB has been used to provide a streamlined mechanism for use. JAXB generates Java class files from an XML schema that can be created and used within Java as common objects. Rather than creating a DOM element and adding a child element to it, using the JAXB generated classes, an instance of a container can be created. An instance of a security class can also be created and have the nonce set with an automatically generated Java function, the container can then incorperate the security element by invoking another generated function. JAXB greatly increased the usability of the schemas and allows users to interact and parse the XML packages more efficiently.

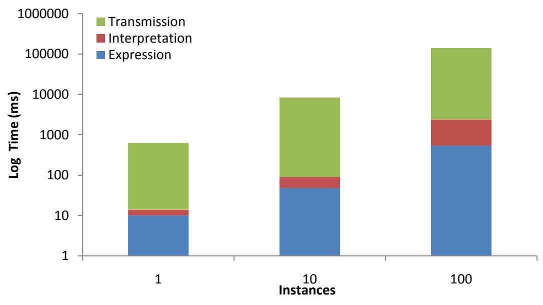


Figure 9.3: Individual reputation response. The time taken to express, interpret and transmit a single exchange protocol for each individual instance of integer based reputation values.

The exchange protocol enables multiple instances of reputation information to be packaged as a single batch response. As the overhead of interpreting different forms of information has already been addressed, instances of integer values have been used during these tests to highlight the overhead of expression. Figure 9.3 shows the time taken to construct a reputation exchange object for each individual instance of the reputation language. The service then transmits the information to the relationship management service, executing on the Google App Engine, for interpretation. The scale is logarithmic as the transmission time is relatively consistent and grows linearly with the number of instances. The figure depicts the rapid increase in time to express and interpret information as the number of instances grows. Figure 9.4 demonstrates the advantage of utilising the batch response functionality. This figure shows the time to express and interpret information is comparatively, far less expensive once the number of instances increases, due to a single exchange object being used to encode the information.

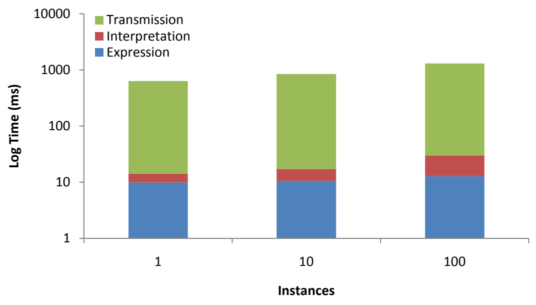


Figure 9.4: Batch reputation response. The time taken to express, interpret and transmit a single exchange protocol containing multiple instances of integer based reputation values.

9.4.1 Protocol Metrics

The following section analyzes the exchange protocol with regard to a set of metrics and evaluates the methods of addressing the requirements. Prominent definition languages and protocols have also been considered to ensure commercial development standards are met.

- **Encoding:** The RX schema is designed to encode reputation information and enable it's transportation. The RX schema includes security, target, provider, information and statement elements. Reputation information is expressible in detail with associated contextual layers under the information element. To give the information meaning, elements regarding the entities involved are encoded as target and provider. The RC schema also supports the transfer of statements and functions. The reputation standards that have been established, specify that the exchange protocol is able to fully describe the action that is being performed. The RC operation element contains elements to specify the mode, be it a response to a request or an invocation of an action, and statues, describing the condition of the response or action. Combining these schemas together provides a protocol capable of fully expressing the reputation information without loss of semantics.

- **Entity Information:** The RX schema contains target and provider elements, both derived from a common entity type. The entity type includes identification of the user, group or potentially organisation. The provider element includes an optional attribute for specifying whether the information is sourced from a human or machine environment, as entailed in the design goals.

- **Integrity:** The security associated with the exchange protocol is provided at two levels, first at the RX level, then more thoroughly at the RC level. The RX security element is a simplified version of the RC security element, which does not include a checksum to reduce overhead. Rather, the security element contains a timestamp and nonce, to provide a level of security for the individual reputation information package. The RC security element encases a timestamp, nonce and checksum. The proper use of the security element can ensure the integrity of the information with a high probability [33]. When compared with other transport protocols and encoding languages, such as the Web Service Definition Language (WSDL) and the SOAP protocol, the RC security element provides a similar standard of data safety. SSL encryption can also easily be enabled between OpenRep nodes, further increasing the reliability of the reputation information.

- **Contextual Expression:** The concept of layering reputation contextual information, as discussed in Chapter 5, has been implemented via the inclusion of a recursive information element in the RX schema. Each level of context includes a summary of the level, allowing users to parse to a particular level of confidence that is appropriate to them and then act on the information without descending further. The potential to provide a users reputation provenance as a list containing each transaction is also incorporated into the schema. This supports the functionality to encode a users entire history, comprised of individual interactions, rather than transmit a single, summarised, element of information. Consumers are then capable of performing their own filters and operations over the information.

- **Parsability:** The concept of reducing the cost to parse the reputation information encoded in these schemas is clearly important. The design goals described in Chapter 4 established the necessity of parsability prior to development. To increase the intuitive nature of the protocol, consistent element structure and a standard naming system has been employed across both schemas. Elements expressing the number of items being transferred have been included wherever practical. Summary fields have also been included within the information type to ease the burden of processing. The summary fields allow information to be interpreted as a single element, rather than having to calculate a result from a set of information. The schemas also include the option of batching a list of transactions in a single message, thus removing the overhead of requesting, receiving, and parsing multiple messages.

9.5 Test Cases

To evaluate the capability of the language, verification system and exchange protocol, a collection of test cases have been defined. The tests have been designed specifically to assess whether each of the components effectively perform their designated task, evaluating whether they can reliably be incorporated into the OpenRep system. The following section discusses the strategy employed and the results collected from the testing procedure.

9.5.1 Test Strategy

The testing process is two-fold. The first set of tests are manually constructed with the specific purpose of testing the functionality of the system and ensuring it behaves as intended under particular circumstances. The second set of tests utilise anonymised, real world, information collected from prominent web sites that are relevant to each of the case studies.

The first set of tests establish whether each of the components performs as expected with various input ranges. The test cases have been defined to represent a variety of information that could be generated and processed, this information range is based on analysis of the case studies. Due to the control over these tests, edge cases and invalid input can be constructed purposefully to examine the performance of each module to enforce constraints.

The testing process is designed to first determine whether the reputation language is capable of expressing reputation information adequately. To investigate this aspect, a range of textual descriptions of reputation information are expressed within the language. Once the capability of the language is established, instances of the language are generated to test the other components. Many test cases representing an extensive range of potential input, including edge conditions and invalid input, have been defined. The tests range in complexity from individual values to complex user defining structures, containing numerous embedded values and collections. Tests specifically defined to utilise the type hierarchy, described in Chapter 7, ensure policy files are consulted during interpretation. Similarly, a full set of function descriptions, such as vouching scenarios, are included to appropriately test the functionality of the Drools rules and policies.

The second set of tests are employed to demonstrate the capability of the components to process real world information. Publically available data has been collected and anonymised from eBay, TradeMe, Slashdot, Digg, Reddit and Facebook for testing purposes. The real world information best simulates the input the components are designed to process and provides a real world basis for this evaluation.

9.5.2 Test Results

The initial test cases focus on the capability of the reputation language to describe information. To do this descriptions of typical information generated within the case study domains are expressed in the reputation language. The descriptions differ substantially in type, complexity and structure. The tests show that the language is capable of expressing various forms of reputation information, ranging from primitive type declarations to complex collections.

Having established that the language is capable of expressing the required information, the exchange protocol can also be tested. The exchange protocol tests examine three specific cases; encoding information and context, encoding sets of information and describing functions. To test the ability of the protocol to encode information, instances of the language must be described. The contextual information associated with the input provides a basis to test the capacity to describe context. Various degrees of context, representing the domains of the case studies, are describable, as is discussed in Section 9.4.1. To investigate the ability to exchange multiple instances of information, a set of numerous information inputs has been encoded. The tests demonstrate the ability to describe reputation provenance within a single transmission. Functions and their responses are also tested to ensure they are expressed in the language and encoded in the protocol, utilising the features of the RC schema to specify the type of input.

Extensive testing is required to demonstrate the verification system performs as intended. The interpretation of information must be consistent and conform to syntactic and semantic constraints. The testing of the verification system involves processing numerous instances of simple and complex forms of information with an emphasis on edge cases and invalid input. The tests incorporate instances requiring hierarchical evaluation, in which policies must be consulted to determine compatibility, and ensuring that warning level error messages are successfully raised. Errors regarding syntactic misuse of the language are identified as the lexer or parser rules fail to build valid parse trees, raising exceptions to be returned to the caller as critical errors. Semantic constraints are enforced through typing, multiple test cases over various different types and structures demonstrate that valid instances of the language are successfully parsed and that invalid instances raise appropriate error messages. Table 9.1 shows a brief example of the tests that have been executed. The different levels of result are evident. Input is deemed to either pass, raise a warning or cause an error. Warnings are raised when type compatibility is not immediately obvious and the type hierarchy must be consulted. Critical errors imply either syntactic or semantic constraints have been broken. Functions are also thoroughly tested through a test suite containing a combination of input data and language functions, these tests include the invocation of Drools rules when necessary.

Instances of social information have been generated to test the performance of the components with this unique form of information. The test cases show invalid social specific types are identified when type errors are present. Social functions behave as intended, providing the ability to perform operations on social types, such as adding and filtering information. As these capabilities are currently deployed they can be performed over live information provided by Facebook.

To demonstrate the real world application of the description and interpretation system, information has been collected from a number of prominent online environments. Various instances of information that are typically consumed by users in eBay and TradeMe are used in the tests to ensure the information is accurately expressed within the reputation language as structures with embedded values. The information is also en-

Case	Syntactic	Semantic	Result
int i = 1	pass	pass	pass
int i = 1.1	pass	warning	warning
int i = ’x’	pass	fail	critical error
int i = abc	fail	-	critical error
machine int i = 1	pass	pass	pass
star rating = {1 .. 5 : 2 }	pass	pass	pass
star rating = {1 .. 5 : ’a’}	fail	-	critical error
star rating = {1 .. 5 : 7}	pass	fail	critical error
array string arr {’one’, ’two’, ’three’}	pass	pass	pass
array string arr {1, ’two’, ’three’}	pass	fail	critical error
tag { [bob] , [alice], 2 }	pass	pass	pass
tag { ’bob’ , ’alice’ }	fail	-	critical error

Table 9.1: A subset of the test cases and their results.

coded in the exchange protocol with contextual information similar to that used in the auction case study. The information can successfully be interpreted by the verification system and processed by the recipient service. To test the virtual community case study, information from Reddit, Digg and Slashdot has also been described in the language. Finally, to evaluate the social case study against real world information, the service has been deployed and run against the Facebook Graph API with valid user credentials. Friends of the logged in user are processed and segregated into trust groups following the defined policies.

The testing proves empirically that each of the components of the description and interpretation system perform as required. Numerous instances of reputation information have been expressed within the language, encoded in the exchange protocol and interpreted by the verification system. The verification system has been tested extensively to ensure the syntactic and semantic constraints of the language are enforced. Finally, real world information has also been tested within the system to enable the evaluation of the system.

9.6 Description and Interpretation System

From the evaluation of the language, verification system and exchange protocol, the entire description and interpretation system can be assessed. The key requirements of the system are that it can adequately describe reputation information and allow standard interpretation when exchanged between OpenRep nodes. The design goals derived portability, such that the module can be included or excluded from an existing reputation system with minimal effort.

The design of the reputation description and interpretation system focused on providing a system that while meeting the overarching requirements, is capable of being integrated within the OpenRep architecture. The system is implemented with a single point of entry to utilise the entire reputation verification system, with tedious tasks being automated. The system relies on an asynchronous event based system, allowing the user's program functionality to be associated with specific events and perform tasks only when appropriate input is given.

The implementation of the system has achieved all that was specified in the design and requirements. The system provides an Interpreter class which users can initiate and assign listeners for various events. The interpreter also provides a function called process, that accepts a string containing an instance of the language. The process function verifies the input through the verification system and the result is returned to the user through a customized event.

The verification successfully employs the developed lexer and parser, allowing the language to be checked for badly formed input and type checked to identify errors before execution.

The case studies have demonstrated the portability and capabilities of the system. The system has been shown to work under different circumstances and within different environments, with overhead being shown to be of little significance. The exchange protocol enables reputation information to be exchanged safely without the loss of meaning.

The simulation has enabled the description and interpretation system to be tested extensively in different situations. Information regarding users reputation within one domain has successfully been transported to another with the invocation of user defined policies determining the aggregation process. Complex operations, such as filters and vouching have been shown to be effective uses of the reputation information and illustrate the potential of such a system to be employed in OpenRep.

9.7 Summary

This chapter has examined many aspects of the development and testing of the reputation description and interpretation system. The case studies have been shown to be an invaluable resource for the design and development process while creating the system. These case studies have given an insight into the use of reputation information in a number of domains and provided a real-world perspective to this work. The reputation language in particular has evolved to meet the diverse requirements of these case studies. The language therefore meets the stated requirements of the description system and provides a common language to enable communication between OpenRep nodes. The type system has been shown to be an effective way of detecting error states and therefore allows users and providers to have confidence that the language is used correctly by others. The exchange protocols have been discussed and shown to fulfil the needs of the overall system. The protocol allows information to be exchanged safely and in a standard way between domains without losing meaning. The combination of these components realises the vision of the interpretation system required for the collection and composition layers of the OpenRep architecture and enables the exchange of reputation information across a federated heterogeneous reputation environment. This has been effectively demonstrated through the implementation of the case studies as scenarios which show the ability to use these resources to accomplish the goals initially set out.

Chapter 10

Conclusion

This thesis presented the design, implementation and evaluation of a reputation description and interpretation system, capable of exchanging information between heterogeneous reputation systems. Reputation can be a powerful tool used to provide an indication of trustworthiness in others. For example, an honest reputation allows others to have confidence that one will perform honourably during an interaction.

Reputation information is increasingly common in many environments, albeit with differing intentions. The ability to exchange reputation information across domains increases the potential pool from which to gather reputation information which in turn could lead to more accurate perspectives of entities' behaviour. Because reputation is generally collected within proprietary systems, the reputation information is generally heterogeneous and incomprehensible between domains. The OpenRep federated reputation system aims to enable collaboration between individual reputation systems by providing mechanisms for standardised reputation exchange. This thesis presents a core underlying component of the OpenRep system, describing a method to express, exchange and interpret reputation information across domains. The major requirements of this system, as presented in Chapter 4 are:

- The definition of a language to express reputation information.

- A mechanism to ensure the syntactic and semantic correctness of language instances.

- The design and implementation of an exchange protocol to transfer reputation information between individual reputation systems.

This chapter reviews the process of meeting these goals, discusses the contributions made in this thesis and presents future research possibilities.

10.1 Review

The description and interpretation system is designed to enable the exchange of reputation information between OpenRep nodes. The system involves a number of different modules relating to the expression, exchange and interpretation of information. Each of the modules are designed to be integrated into the OpenRep stack, as shown in Chapter 2. The following sections discuss the core modules and concepts presented in the thesis, summarising their contribution to the exchange of reputation information.

10.1.1 Humans and Machine Reputation

The integration of human and machine entities in the context of OpenRep was identified as a potential issue in Chapter 4. To establish whether humans and machines should be integrated, Chapter 5 examined the differences between human and machine users by studying actions taken during the definitive phases of a reputation system. Through the investigation of multiple reputation systems, seven phases of induced reputation action were established: resource detection, search, collection, calculation, interpretation, consumption, and the generation process. Having studied a number of reputation systems, specifically designed for either human or machine users, the primary difficulty limiting the integration of humans and machines was found to be the ability to learn and adapt.

Although a task of a reputation system is to provide insight into entity selection, human entities are capable of probing others to develop an understanding of their motives. In the case of a machine based entity there is potential for its programmed behaviour to be learnt and exploited by observant human users. Due to the realisation that there is a difference between human and machine based entities, the process of describing the entities generating information was identified as a necessity for the reputation language.

10.1.2 Exchange Protocol

The exchange protocol enables reputation information to be encoded in a standard way without loss of semantic meaning. Integrating the contextual expression features, discussed in Chapter 5, provided a foundation to express reputation information in full, permitting standardised transportation. The exchange protocol consists of two distinct XML schemas, the Reputation Exchange (RX) and Reputation Container (RC). The RX schema allows reputation information regarding an individual entity to be encoded securely. The RC is capable of encompassing multiple RX elements in order to encode detailed reputation messages. Together, the RX and RC modules form the exchange protocol and enable OpenRep nodes to communicate with one another.

10.1.3 Reputation Language

The requirement to express reputation is fundamental to OpenRep's ability to exchange reputation information. The main goal of the reputation language, described in Chapter 6, is to accurately describe reputation information without losing semantic meaning. Various design goals for the language were identified in Chapter 4, including providing an appropriate set of terminal types, complex structures and creating an intuitive syntax. The language incorporates a complete set of primitive and complex types. Reputation information can be described in full, without the loss of relationships between data types. The syntax of the language leverages existing languages, such as Java and Pascal, which enables intuitive use. The language also supports the expression of reputation based functions. Filtering information, asserting statements, applying events and performing vouch exchanges are each expressible in the language. The language has been evaluated against metrics regarding expressiveness, readability, writability, consistency, operation support, cost and portability. The language provides a comprehensive platform to demonstrate the potential for OpenRep nodes to communicate, exchange reputation information and invoke operations between one another.

10.1.4 Verification System

The verification system, presented in Chapter 7, is designed to interpret and verify use of the reputation language. The system performs checks, enforcing syntactical and semantic constraints over the language. The inclusion of a verification system standardises the method by which OpenRep nodes interpret reputation information. When information is exchanged between either the *collection* and *composition* layers, or between OpenRep nodes, the information is processed by the verification system. The verification system is comprised of three elements; the interface, language parser unit and type enforcement system. The interface provides an asynchronous method of processing information, where users are capable of listening for various notifications depending on the output of the interpreter. The language parser processes the language instance against the parser and lexer rules generated during the construction of the reputation language. The type enforcement system is responsible for ensuring semantic errors are identified. The language is designed to be statically checked and strongly typed, meaning checks regarding type use are performed during the compilation of the language.

A type hierarchy, described in Chapter 7, enables the compatibility of types to be taken into account during interpretation. The verification system is capable of identifying type errors regarding incompatible values and incompatible arguments. Combining each of the verification system modules creates a usable service to verify instances of the reputation language. The verification system can be integrated into the *composition* layer of the OpenRep stack to facilitate the interpretation of information.

10.1.5 Case Studies

The three case studies, discussed in Chapter 8, represent a diverse range of reputation systems. The case studies were selected to best demonstrate the capabilities of the OpenRep system to be incorporated into existing reputation systems. An electronic auction site, social network and virtual community have been used to focus the development and assess requirements of the system throughout this thesis. The three case studies are each implemented as OpenRep node services, exposing a REST interface in line with the OpenRep standards discussed in Chapter 5. The auction service provides a higher degree of functionality than the other case study implementations, capable of requesting information from the other implemented OpenRep services. The social friendship management service and virtual community forum provide the functionality to collect information from a Facebook application and a Drupal forum module respectively. This reputation information is then distributed throughout the network.

10.2 Contributions

This thesis makes multiple contributions to the development of a federated reputation system. The expression and interpretation of reputation information forms the basis of a framework to exchange information between reputation systems. The description of contextual information, differences between human and machine entities within reputation environments and standards regarding interoperability between reputation systems have also been presented. Specifically the main contributions of this thesis are:

- The design, prototype implementation and evaluation of a reputation description and interpretation system. The description and interpretation system provides many features to facilitate the exchange of reputation information. The key elements of the system are:
    - The development of a reputation language, used to express reputation information and provide a common form of communication between reputation sources. The language supports the description of reputation values and operations.
    - The definition of reputation types, capable of describing all identified forms of reputation information. The typing system includes the definition of domain specific types, such as the classification of interactions between users of a social network.
    - The design and implementation of a verification system for the reputation language. The verification system includes a static type enforcement system to ensure correct use of the language.
    - An exchange protocol, defined in XML schemas, standardises the encoding and transfer of reputation information. The protocol regulates communication between reputation sources and embodies multiple features to reduce the overhead of communication.
    - The consistent expression of contextual reputation information allows reputation sources to encode information without loss of detail. Through the formalisation of the environment with regard to context, information can be interpreted with meaning.

- An investigation regarding the plausibility of integrating human and machine based reputation systems. Seven phases of a reputation system have been identified from the examination of a diverse set of human and machine based reputation systems. In addition, investigation of reputation systems with respect to the actions available to users under each of the identified phases provides a unique insight into the complexities associated with the integration of human and machine generated reputation information.

- The design and implementation of a reputation environment, containing three uniquely constructed case studies. Each of the case studies represents a potential reputation source within an open environment. The case studies provide a practical basis to drive the development and evaluation of the description and interpretation system. The case studies include an electronic auction, virtual community and social network based relationship management service. The services are each independently implemented as a Jersey Web service utilising a Derby database, a Drupal client with a built in forum module and a Facebook application hosted in the Google app engine.

10.3 Future Work

This thesis has identified a number of areas of future work, possible extensions, and potential research areas.

10.3.1 OpenRep Integration

The primary direction of future work is to fully integrate the description and interpretation system into the OpenRep architecture. When complete, OpenRep will establish a federated reputation system in an open environment, using the modules presented in this thesis as core components.

The exchange protocol will be supported by the composition layer of the OpenRep stack in order to enable reputation systems to communicate in a standardised way. The verification system will also be integrated into the composition layer to interpret and validate information from the collection layer and communications from the other OpenRep nodes. The reputation language will be used in OpenRep to provide a standard medium in which to express reputation information, allowing it to be exchanged across disparate domains and retain meaning.

10.3.2 Social Relationship Management Service

The implementation of the social network based case study could be extended in multiple ways. For example, utilising Google's available cloud based data management for user information would increase the applications usability by other resources, improve availability, and increase scalability. The implementation also highlighted the use of novel social information as a means of forming reputation, a similar architecture could be applied in many other social applications, for example, the social relationship management service could be applied within the Social Cloud [11] architecture to provide an explicit trust measurement.

10.3.3 Reputation Sources

The ability to derive reputation information from sources that have not been explicitly designed to generate reputation information should be investigated. In conjunction with global identification, the inclusion of citation indexing websites, for example CiteSeerX[1], and search engines would allow a vast amount of reputation information to be collected. An investigation to identify other potential reputation sources, such as these, and an evaluation of the value of their information with regard to the level of trust that can be extracted, is an exciting area of future work.

[1]http://citeseer.ist.psu.edu

Appendix A

Reputation Language Grammar

```
grammar ReputationLanguage;

//-----------------------------------------------------------
// Parser
//-----------------------------------------------------------

options { output=AST;}

@header { package nz.ac.vuw.ecs.chard.ryan.reputation; }

input  : userType? information EOF -> ^(INFORMATION (userType)?
 information)
;

userType: HUMAN -> ^(HUMAN)
| MACHINE -> ^(MACHINE)
;

information
```

```
: socialInformation -> ^(SOCIAL socialInformation)
| reputationInformation
| statement
;

//Social information
socialInformation
: socialValue -> socialValue
| socialFunction -> ^(FUNCTION socialFunction)
;



//Social values: tag { [one] , [two] , 3 }
socialValue
: socialTypeIdentifier LCURL idValue COMMA idValue (COMMA
INT_LITERAL)? RCURL -> ^(socialTypeIdentifier idValue idValue
 (INT_LITERAL)?)
;

//Social functions: add ( [one] , [two] ) | filter ( tag > 10 )
socialFunction
:  SOCIALADD LPAREN socialValue COMMA socialValue RPAREN
 -> ^(SOCIALADD socialValue socialValue)
| SOCIALSUB LPAREN socialValue COMMA socialValue RPAREN
-> ^(SOCIALSUB socialValue socialValue)
| FILTER LPAREN socialTypeIdentifier operator socialValue RPAREN
 -> ^(FILTER socialTypeIdentifier operator socialValue)
;

//Reputation statements
statement
```

```
: filterStatement
| ASSERT LPAREN ID operator statementParam RPAREN
 -> ^(ASSERT ID operator statementParam)
| VOUCH LPAREN id FOR id (COLON bool)? RPAREN
 -> ^(VOUCH id id bool?)
| EVENT ID EQUALS LPAREN ID operator statementParam
 COLON rule RPAREN -> ^(EVENT ID operator statementParam rule)
;

//Rules
rule : (RULE ID | ID operator statementParam) -> ^(RULE ID
operator statementParam)
;

//Parameters for the statements
statementParam
: idValue -> ^(ID idValue)
|  literal -> ^(LITERAL literal)
;
//Filter type for recursive feature
filterStatement
: FILTER LPAREN filterValue operator statementParam RPAREN
 -> ^(FILTER filterValue operator statementParam)
;
filterValue
: id -> id
| filterStatement -> filterStatement
;

//Reputation based information
reputationInformation
```

```
: literalDec -> literalDec
| reputationStructures
;

//Literals
literalDec
: id EQUALS literal -> ^(LITERAL id literal)
;

//Structures
reputationStructures
: enumDec
| listDec
| arrayDec
| structDec
| starDec
;

//Enumeration type: enum nums = {a1 , a2, a3 : a3}
enumDec : ENUM ID EQUALS LCURL enumVals (COMMA enumVals)*
 COLON enumVals RCURL -> ^(ENUM ID enumVals (enumVals)*)
;
enumVals: ID -> ^(ID)
;

//List type: list user = {string name = 'one', string location
 = 'two, int val = 3}
listDec : LIST ID EQUALS LCURL listVal (COMMA listVal)* RCURL
-> ^(LIST ID listVal (listVal)*)
;
listVal: literalDec -> literalDec
```

```
;

//Array type: array string arr {'one', 'two', 'three'}
arrayDec: ARRAY idType ID EQUALS LCURL arrayVals RCURL ->
^(ARRAY idType ID arrayVals)
;
arrayVals
: literal (COMMA literal)* -> ^(VALUES literal (literal)*)
;

//Struct type: struct strct = { string name = 'bob', int count = 3,
array string content = { 'one', 'two', 'three' } }
structDec
: STRUCT ID EQUALS LCURL information
(COMMA information)* RCURL -> ^(STRUCT ID
 information (information)*)
;

//Star type: star rating = {1 .. 5 : 3}
starDec : STAR ID EQUALS LCURL subrangeType
COLON INT_LITERAL RCURL -> ^(STAR ID subrangeType INT_LITERAL)
;

//ID types
id : idValue -> ^(ID idValue)
| idType idValue -> ^(idType idValue)
;

idType : USER
| GROUP
| ROLE
| primitiveType -> primitiveType
| ID
;

idValue : ID
| SOCIALID
| SOCIALNAME
;

//Social type identifiers
socialTypeIdentifier
: SOCIALLIKE
| SOCIALPOST
| SOCIALCOMMENT
| SOCIALTAG
;


//Value types
subrangeType
: INT_LITERAL DOTDOT INT_LITERAL ->
^(RANGE INT_LITERAL DOTDOT INT_LITERAL)
;

primitiveType
: BOOLEAN
| CHAR
| STRING
| INT
| FLOAT
;
```

```
literal
: bool
| CHAR_LITERAL
| INT_LITERAL
| FLOAT_LITERAL
| STRING_LITERAL
| NULL
;

bool : TRUE
| FALSE
;

//Operators
operator: NOT_EQUALS
| LT
| LTE
| GT
| GTE
| PLUS
| MINUS
| TIMES
| DIV
| IN
;




//---------------------------------------------------------
```

```
// Lexer
//---------------------------------------------------------


//Number literals
INT_LITERAL
: DIGITS+
;

FLOAT_LITERAL
: DIGITS '.' (DIGITS)?
| '.' DIGITS
;

fragment
DIGITS
: ('0'..'9')+
;

//Character and String Literal
CHAR_LITERAL
: '\'' LITERAL_CHAR '\''
;

STRING_LITERAL
: '\'' LITERAL_CHAR LITERAL_CHAR* '\''
;

fragment
LITERAL_CHAR
: ESC
```

```
| ~('\''|'\\')
;

fragment
ESC : '\\'
( 'n'
| 'r'
| 't'
| 'b'
| 'f'
| '"'
| '\''
| '\\'
| '>'
| 'u' XDIGIT XDIGIT XDIGIT XDIGIT
)
;

fragment
XDIGIT : '0' .. '9'
| 'a' .. 'f'
| 'A' .. 'F'
;

//Comments
COMMENT : '//'(~('\n'|'\r'))* {$channel=HIDDEN;}
;


//White space
WS : ( ' '
| '\t'
| '\f'
| ( '\r\n'
| '\r'
| '\n'
)
)
{$channel=HIDDEN;}
;



//Key words
CHAR : 'char';
STRING : 'string';
BOOLEAN : 'boolean';
INT : 'int';
FLOAT : 'float';
DOUBLE : 'double';
TRUE : 'true';
FALSE : 'false';
NULL : 'null';
STRUCT : 'struct';
FILTER : 'filter';
ASSERT : 'assert';
VOUCH : 'vouch';
ENUM : 'enum';
RANGE : 'range';
LIST : 'list';
ARRAY : 'array';
STAR : 'star';
```

```
GROUP : 'group';
ROLE : 'role';
USER : 'user';
FOR : 'for';
RULE : 'rule';
EVENT : 'event';
ITEM : 'item';
VALUE : 'value';
VALUES : 'values';
FUNCTION: 'function';
HUMAN : 'human';
MACHINE : 'machine';

//Social key words
LITERAL : 'literal';
SOCIAL : 'social';
SOCIALPOST : 'post';
SOCIALTAG : 'tag';
SOCIALLIKE : 'like';
SOCIALCOMMENT
: 'comment';
SOCIALADD
: 'add';
SOCIALSUB
: 'sub';
INFORMATION
: 'information';

//Operators
LCURL : '{';
RCURL : '}';
```

```
LPAREN : '(';
RPAREN : ')';
LSQR : '[';
RSQR : ']';
COMMA : ',';
DOT : '.';
DOTDOT : '..';
COLON : ':';
NOT : '!';
NOT_EQUALS
: '!=';
EQUALS : '=';
LT : '<';
LTE : '<=';
GT : '>';
GTE : '>=';
PLUS : '+';
MINUS : '-';
TIMES : '*';
DIV : '/';
AND : '&&';
OR : '||';
IN : 'in';



//An identifier
ID
//options {testLiterals=true;}
: ('a'..'z')('a'..'z'|'0'..'9'|'_')*
;
```

```
//social identifiers
SOCIALNAME
: LSQR ('a'..'z')('a'..'z' | ' ' | '_' | '-' | '\'')* RSQR
;

SOCIALID
: LSQR INT_LITERAL RSQR
;
```

Appendix B

Reputation Container Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:rc="http://specs.open-rep.net/rc/1.0"
xmlns:rx="http://specs.open-rep.net/rx/1.0"
elementFormDefault="qualified">
 <!-- the container object -->
  <xs:element name="container">
   <xs:complexType>
    <xs:sequence>
     <!-- security -->
     <xs:element name="security" type="secureType"/>
     <!-- operation -->
     <xs:element name="operation" type="opType"/>
     <!-- statesments -->
     <xs:element name="statements" type="statementsType"
 minOccurs="0"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <!-- security type -->
```

```
<xs:complexType name="secureType">
 <xs:sequence>
  <xs:element name="timestamp" type="xs:date"/>
   <xs:element name="nonce" minOccurs="0">
    <xs:simpleType>
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
   </xs:element>
   <xs:element name="checksum" minOccurs="0">
    <xs:simpleType>
    <xs:restriction base="xs:string"/>
    </xs:simpleType>
   </xs:element>
 </xs:sequence>
</xs:complexType>
<!-- opertion type -->
<xs:complexType name="opType">
 <xs:sequence>
  <xs:element name="mode">
   <xs:simpleType>
    <xs:restriction base="xs:string"/>
     </xs:simpleType>
    </xs:element>
    <xs:element name="status" minOccurs="0">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="code" type="xs:integer" use="required"/>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- statement type -->
  <xs:complexType name="statementsType">
   <xs:sequence>
    <xs:any/>
     </xs:sequence>
    <xs:attribute name="count" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:schema>
```

Appendix C

Reputation Exchange Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:rc="http://specs.open-rep.net/rc/1.0"
 xmlns:rx="http://specs.open-rep.net/rx/1.0"
 elementFormDefault="qualified">
<!-- the reputation object -->
 <xs:element name="reputation">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="security" type="secureType"/>
     <xs:element name="target" type="targetType"/>
     <xs:element name="provider" type="providerType"/>
     <xs:element name="information" type="informationType"/>
     <xs:element name="statement" type="statementType"
minOccurs="0"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <!-- security type -->
  <xs:complexType name="secureType">
```

```
   <xs:sequence>
    <xs:element name="timestamp" type="xs:date"/>
    <xs:element name="nonce" id="token">
     <xs:simpleType>
      <xs:restriction base="xs:normalizedString">
       <xs:whiteSpace value="collapse"/>
      </xs:restriction>
     </xs:simpleType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <!-- entity type -->
 <xs:complexType name="entityType">
  <xs:sequence>
   <xs:element name="identity" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="type" type="xs:string"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="group" minOccurs="0">
    <xs:complexType>
    <xs:attribute name="type" type="xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <!-- target type -->
 <xs:complexType name="targetType">
  <xs:complexContent>
   <xs:extension base="entityType">
    <xs:sequence>
```

```
      <xs:element name="value" type="xs:string" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="type" type="xs:string"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
  <!-- provider type -->
  <xs:complexType name="providerType">
   <xs:complexContent>
    <xs:extension base="entityType">
     <xs:attribute name="type" type="xs:string"/>
     <xs:attribute name="source" type="sourceType"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
  <!-- source type -->
  <xs:simpleType name="sourceType">
   <xs:restriction base="xs:string">
    <xs:enumeration value="human"/>
    <xs:enumeration value="machine"/>
   </xs:restriction>
  </xs:simpleType>
  <!-- informaiton type -->
  <xs:complexType name="informationType">
   <xs:sequence>
    <xs:element name="context" type="informationType"
minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="transactions" type="transactionType"
minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="summary" type="summaryType"/>
   </xs:sequence>
```

```
 <xs:attribute name="type" type="xs:string" use="required"/>
 <xs:attribute name="role" type="xs:string"/>
</xs:complexType>
<!-- summary type -->
<xs:complexType name="summaryType">
 <xs:sequence>
  <xs:element name="value" type="reputationType"/>
  <xs:element name="count" type="xs:integer" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<!-- transaction type -->
<xs:complexType name="transactionType">
 <xs:sequence>
  <xs:element name="date" type="xs:date"/>
  <xs:element name="role" type="xs:string"/>
  <xs:element name="value" type="reputationType"/>
  <xs:any minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<!-- reputation information type -->
<xs:complexType name="reputationType">
 <xs:sequence>
 <xs:element name="value" type="xs:string"/>
  <xs:any minOccurs="0"/>
  </xs:sequence>
 <xs:attribute name="type" type="xs:string"/>
</xs:complexType>
<!-- statement type -->
<xs:complexType name="statementType">
 <xs:sequence>
  <xs:element name="value" type="xs:string"/>
```

```
    </xs:sequence>
    <xs:attribute name="type">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="filter"/>
       <xs:enumeration value="assert"/>
       <xs:enumeration value="vouch"/>
       <xs:enumeration value="event"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:schema>
```

Bibliography

[1] *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, May 2003.

[2] ABADI, M., CARDELLI, L., PIERCE, B., AND PLOTKIN, G. Dynamic typing in a statically-typed language. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 1989), POPL '89, ACM, pp. 213–227.

[3] ABERER, K., DATTA, A., DESPOTOVIC, Z., HAUSWIRTH, M., PUNCEVA, M., AND SCHMIDT, R. P-grid: A self-organizing structured p2p system, July 28 2003.

[4] ABERER, K., AND DESPOTOVIC, Z. Managing trust in a peer-2-peer information system. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management* (New York, NY, USA, 2001), ACM, pp. 310–317.

[5] BAARS, A. I., AND SWIERSTRA, S. D. Typing dynamic typing. *SIGPLAN Not. 37* (September 2002), 157–166.

[6] BAKKEN, S. S., AULBACH, A., SCHMID, E., WINSTEAD, J., WILSON, L. T., LERDORF, R., ZMIEVSKI, A., AND AHTO, J. PHP manual.

[7] BALI, M. *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009.

[8] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., EVE, AND YERGEAU, F., Eds. *Extensible Markup Language (XML) 1.0*, fourth ed. W3C Recommendation. W3C, Aug. 2003.

[9] CANTOR, S. Shibboleth architecture protocols and profiles. http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-arch-protocols-latest.pdf, September 2005.

[10] CARDELLI, L. Type systems. In *The Computer Science and Engineering Handbook*, A. B. Tucker, Ed. CRC Press, 1997, ch. 103, pp. 2208–2236.

[11] CHARD, K., BUBENDORFER, K., CATON, S., AND RANA, O. Social Cloud Computing: A Vision for Socially Motivated Resource Sharing. *IEEE Transactions on Services Computing 99*, PrePrints (2011).

[12] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. Web service definition language (wsdl). Tech. rep., Mar. 2001.

[13] CIURANA, E. *Developing with Google App Engine*. Springer, 2009.

[14] COMPANY, S. *Maven: The Definitive Guide*, 1 ed. O'Reilly Media, Inc., Oct. 2008.

[15] CROCKFORD, D. The application/json media type for javascript object notation (json). RFC 4627, July 2006.

[16] CZAJKOWSKI, K., FERGUSON, D. F., FOSTER, I., FREY, J., GRAHAM, S., SEDUKHIN, I., SNELLING, D., TUECKE, S., AND VAMBENEPE, W. The ws-resource framework. Tech. rep., Globus, 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf [Accessed Nov 2011].

[17] DELLAROCAS, C. The digitization of word of mouth: Promise and challenges of online feedback mechanisms. *Management Science 49* (2003), 1407–1424.

[18] EYSHOLDT, M., AND BEHRENS, H. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (New York, NY, USA, 2010), SPLASH '10, ACM, pp. 307–309.

[19] FIALLI, J., AND VAJJHALA, S. Java architecture for xml binding (jaxb) 2.0. Java Specification Request (JSR) 222, October 2005.

[20] FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[21] GAL-OZ, N., GRINSHPOUN, T., AND GUDES, E. Privacy issues with sharing reputation across virtual communities. In *Proceedings of the 4th International Workshop on Privacy and Anonymity in the Information Society* (New York, NY, USA, 2011), PAIS '11, ACM, pp. 3:1–3:5.

[22] GAMBETTA, D. *Trust: Making and Breaking Cooperative Relations*. Blackwell Publishers, February 1990.

[23] GAO, S., SPERBERG-MCQUEEN, C. M., THOMPSON, H. S., MENDELSOHN, N., BEECH, D., MALONEY, M., PETERSON, D., MALHOTRA, A., AND BIRON, P. V. *XML Schema Definition Language (XSD) 1.1*, 2008.

[24] GRINSHPOUN, T., GAL-OZ, N., MEISELS, A., AND GUDES, E. Ccr: A model for sharing reputation knowledge across virtual communities. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on* (sept. 2009), vol. 1, pp. 34 –41.

[25] HAMMER-LAHAV, E. The oauth 2.0 authorization protocol. Tech. rep., Sept. 2011.

[26] HARBISON, S. P., AND STEELE, G. L. *C, a Reference Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[27] HENDRIX, F., AND BUBENDORFER, K. Openrep: A distributed reputation system architecture. 2011.

[28] HENDRIX, F., CHARD, R., AND BUBENDORFER, K. A taxonomy of reputation systems. 2011.

[29] HORS, A. L., HGARET, P. L., WOOD, L., NICOL, G., ROBIE, J., CHAMPION, M., AND BYRVE, S. Document object model (dom) level 3 core specification. W3C Recommendation, April 2004.

[30] JENSEN, K., WIRTH, N., MICKEL, A. B., AND MINER, J. F. *Pascal User Manual and Report: ISO Pascal Standard*. Springer, Sept. 1991.

[31] JØSANG, A., ISMAIL, R., AND BOYD, C. A survey of trust and reputation systems for online service provision. *Decision Support Systems 43*, 2 (2007), 618–644.

[32] KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International World Wide Web Conference* (New York, NY, USA, 2003), ACM, pp. 640–651.

[33] KEHNE, A., SCHÖNWÄLDER, J., AND LANGENDÖRFER, H. A nonce-based protocol for multiple authentications. *SIGOPS Oper. Syst. Rev. 26* (October 1992), 84–89.

[34] KODAGANALLUR, V. Incorporating language processing into java applications: A javacc tutorial. *IEEE Softw. 21* (July 2004), 70–77.

[35] KREPS, D. M., AND WILSON, R. Reputation and imperfect information. *Journal of Economic Theory 27* (1982), 253–279.

[36] LEVINE, J. *Flex & Bison: Text Processing Tools*. O'Reilly, Aug. 2009.

[37] LEVINE, J., MASON, T., AND BROWN, D. *lex & yacc, 2nd Edition (A Nutshell Handbook)*. O'Reilly, Oct. 1992.

[38] LINDHOLM, T., AND YELLIN, F. *Java Virtual Machine Specification*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[39] MADSEN, O. L., MAGNUSSON, B., AND MØLIER-PEDERSEN, B. Strong typing of object-oriented languages revisited. In *Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications* (New York, NY, USA, 1990), OOPSLA/ECOOP '90, ACM, pp. 140–150.

[40] MALIK, Z., AND BOUGUETTAYA, A. Rateweb: Reputation assessment for trust establishment among web services. *VLDB J. 18*, 4 (2009), 885–911.

[41] MARTI, S., GANESAN, P., AND GARCIA-MOLINA, H. Sprout: P2p routing with social networks. In *Volume 3268 of Lecture Notes in Computer Science* (2004).

[42] MASSA, P., AND BHATTACHARJEE, B. Using trust in recommender systems: An experimental analysis. In *In Proceedings of iTrust2004 International Conference* (2004), pp. 221–235.

[43] MCCARTHY, J. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962.

[44] MEGGINSON, D. Simple API for XML (SAX 2.0). http://sax.sourceforge.net/, 2004.

[45] MERNIK, M., HEERING, J., AND SLOANE, A. M. When and how to develop domain-specific languages. *ACM Comput. Surv. 37* (December 2005), 316–344.

[46] MICHIARDI, P., AND MOLVA, R. Core: A collaborative reputation mechanism to enforce node cooperation. In *in Mobile Ad Hoc Networks. Communication and Multimedia Security* (2002).

[47] MILGROM, P., AND ROBERTS, J. Predation, reputation, and entry deterrence. *Journal of Economic Theory 27*, 2 (Aug. 1982), 280–312.

[48] MOODIE, M. *Pro Apache Tomcat 6*, 1 ed. Apress, 2007.

[49] NASH, J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America 36*, 1 (January 1950), 48–49.

[50] NEUMAN, C., KOHL, J., YU, T., HARTMAN, S., AND RAEBURN, K. The kerberos network authentication service (v5). Tech. rep., 1993.

[51] NIELSEN, H. F., MENDELSOHN, N., MOREAU, J. J., GUDGIN, M., AND HADLEY, M. Soap version 1.2 part 1: Messaging framework. W3c recommendation, W3C, June 2003.

[52] PARR, T., AND FISHER, K. Ll(*): the foundation of the antlr parser generator. *SIGPLAN Not. 46* (June 2011), 425–436.

[53] PERERA, S., HERATH, C., EKANAYAKE, J., CHINTHAKA, E., RANABAHU, A., JAYASINGHE, D., WEERAWARANA, S., AND DANIELS, G. Axis2, middleware for next generation web services. In *Proceedings of the IEEE International Conference on Web Services* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 833–840.

[54] PINGEL, F., AND STEINBRECHER, S. Multilateral secure cross-community reputation systems for internet communities. In *Trust, Privacy and Security in Digital Business*, S. Furnell, S. Katsikas, and A. Lioy, Eds., vol. 5185 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, pp. 69–78.

[55] POTOCIAR, M. Jsr 311: Jax-rs: The java api for restful web services. Tech. rep., Oracle, 2009.

[56] PRAKASH, V. V., AND O'DONNELL, A. Fighting spam with reputation systems. *Queue 3* (November 2005), 36–41.

[57] QUILL, A. The role of reputation in online markets for small-ticket goods, 2007. MEA Conferent - Energy and Environmetnal Economics 2007.

[58] RAUB, W., AND WEESIE, J. Reputation and efficiency in social interactions: An example of network effects. *The American Journal of Sociology 96*, 3 (1990), 626–654.

[59] REBAHI, Y., MUJICA, V., AND SISALEM, D. A reputation-based trust mechanism for ad hoc networks. In *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 37–42.

[60] RECORDON, D., AND REED, D. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management* (New York, NY, USA, 2006), DIM '06, ACM, pp. 11–16.

[61] RESNICK, P., ZECKHAUSER, R., SWANSON, J., AND LOCKWOOD, K. The value of reputation on ebay: A controlled experiment. *Experimental Economics 9* (2003), 79–101.

[62] SABATER, J., AND SIERRA, C. Regret: reputation in gregarious societies. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents* (New York, NY, USA, 2001), ACM, pp. 194–195.

[63] SABATER, J., AND SIERRA, C. Social regret, a reputation model based on social relations. *SIGecom Exch. 3* (December 2001), 44–56.

[64] SABATER, J., AND SIERRA, C. Social regret, a reputation model based on social relations. *SIGecom Exch. 3*, 1 (2002), 44–56.

[65] SCHLOSSER, A., VOSS, M., AND BRÜCKNER, L. On the simulation of global reputation systems. *Journal of Artificial Societies and Social Simulation 9* (2005), 1.

[66] SCOWEN, R. S. Extended bnf - a generic base standard. In *Proceedings of the 1993 Software Engineering Standards Symposium (SESS'93)* (Aug. 1993).

[67] SELCUK, A., UZUN, E., AND PARIENTE, M. A reputation-based trust management system for p2p networks. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on* (april 2004), pp. 251 – 258.

[68] SQUICCIARINI, A. C., BERTINO, E., FERRARI, E., AND RAY, I. Achieving privacy in trust negotiations with an ontology-based approach. *IEEE Trans. Dependable Secur. Comput. 3* (January 2006), 13–.

[69] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review 31*, 4 (2001), 149–160.

[70] STROUSTRUP, B. C++. In *Encyclopedia of Computer Science*. John Wiley and Sons Ltd., Chichester, UK, pp. 174–176.

[71] TONG, X., AND ZHANG, W. Group trust and group reputation. In *ICNC '09: Proceedings of the 2009 Fifth International Conference on Natural Computation* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 561–565.

[72] TOURZAN, J., AND KOGA, Y. Liberty id-wsff architecture overview, version 2.0.

http://projectliberty.org/liberty/content/download/2207/14681/file/liberty-id-wsf-2.0-mrd-v1.0.pdf, Accessed 2009., 2006.

[73] TURING, A. M. Computing machinery and intelligence. *Mind LIX* (1950), 433–460.

[74] WANG, Y., AND VASSILEVA, J. Toward trust and reputation based web service selection: A survey. *International Transactions on Systems Science and Applications 3* (2007), 118–132.

[75] WILSON, R. Reputations in games and markets. In *Game-theoretic models of bargaining*, A. E. Roth, Ed. Cambridge University Press, Cambridge, 1985, pp. 27–62.

[76] WRIGHT, A. K., AND FELLEISEN, M. A syntactic approach to type soundness. *Information and Computation 115* (1992), 38–94.

[77] WU, X., HE, J., AND XU, F. A group-based reputation mechanism for mobile p2p networks. In *GPC '09: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 410–421.

[78] XIE, M., AND WANG, H. A collaboration-based autonomous reputation system for email services. In *Proceedings of the 29th conference on Information communications* (Piscataway, NJ, USA, 2010), INFOCOM'10, IEEE Press, pp. 992–1000.

[79] XIONG, L., AND LIU, L. Peertrust: supporting reputation-based trust for peer-to-peer electronic communities. *Knowledge and Data Engineering, IEEE Transactions on 16*, 7 (2004), 843–857.

[80] YANG, B., AND GARCIA-MOLINA, H. Designing a super-peer network.

[81] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A resilient global-scale overlay

for service deployment. *IEEE Journal on Selected Areas in Communications 22* (2004), 41–53.