Network-Aware Evaluation Environment for Reputation Systems

Alessandro Celestini, Rocco Nicola, Francesco Tiezzi

Network-aware Evaluation Environment for Reputation Systems*

Alessandro Celestini[1], Rocco De Nicola[1], and Francesco Tiezzi[1]

IMT Institute for Advanced Studies Lucca, Italy
{alessandro.celestini,rocco.denicola,francesco.tiezzi}@imtlucca.it

Abstract. Parties of reputation systems rate each other and use ratings to compute reputation scores that drive their interactions. When deciding which reputation model to deploy in a network environment, it is important to find the most suitable model and to determine its right initial configuration. This calls for an engineering approach for describing, implementing and evaluating reputation systems while taking into account specific aspects of both the reputation systems and the networked environment where they will run. We present a software tool (NEVER) for network-aware evaluation of reputation systems and their rapid prototyping through experiments performed according to user-specified parameters.

Keywords: Reputation systems, Network-awareness, Evaluation tool

1 Introduction

In recent years, we have seen an increasing use of reputation systems in different areas of ICT, from e-commerce to different forms of open computer networking, such as P2P, ad-hoc, or sensor networks. This phenomenon is likely to continue, due to the success of networked applications (like social networks or other Web 2.0 technologies) and to the need, in such environments, of instruments to build up relationships of trust among the interacting parties. In order to establish such trust relationships, parties in a reputation system are free to interact and rate each other after any interaction, such ratings are then used to derive parties' reputation scores. The computed *reputation* score is a collective measure of parties' trustworthiness and is used when selecting the party to interact with.

Parties in a reputation system can exchange ratings and interact by relying on a network infrastructure. If we take as a starting point a centralised architecture that is widely used for networked trust infrastructures reported in Figure 1, we have a *rating server* collects ratings from system's parties and makes them publicly available, while a *search server* allows parties to find resource providers in the system. Every party can play the role of a client, of a provider, or both, and may offer different kinds of resources (services, computational and storage resources, etc.). Whenever a party needs a resource, first it queries the search server to get the list of parties providing it, and then retrieves from the rating server the ratings of each provider in the list. Thus, to choose a provider, it computes the reputation scores of each of them and selects the one with the highest reputation score. Finally, after the interaction, it rates the provider

* This work has been partially sponsored by the EU project ASCENS, 257414.

according to the quality of the provided resource. On top of the general infrastructure just described, different kinds of reputation system can be layered, which mainly differ for the model they use to aggregate ratings when computing reputation scores. Several models have been proposed and once a reputation system has to be deployed in a network environment, we might ask which reputation model is more suitable for the given environment and how the reputation system should be configured in order to meet the desired behaviour. This calls for an engineering approach for describing, implementing and evaluating reputation systems while taking into account real-world implementation details of such systems and of the network environment where they have to be deployed.

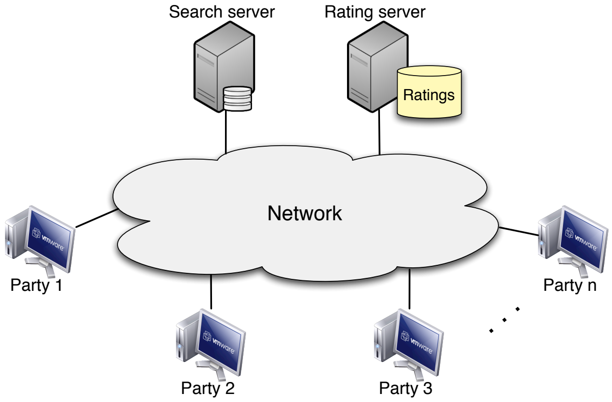

Fig. 1: General infrastructure of a reputation system

In this paper, we address this issue by introducing NEVER a software tool for network-aware evaluation of reputation systems. On the one hand, we provide a framework for rapidly developing Java-based implementations of reputation system models and for easily configuring different networked execution environments on top of which the systems will run. On the other hand, we offer a tool that automatically performs experiments on the reputation system implementations according to user-specified parameters; this enables the study of their behaviour while executing on given network infrastructures. The main novelty of our approach, with respect to other proposals in the literature with a similar aim, is that we allow the evaluation of implemented reputation systems through experiments on real networks, rather than performing simulation of models of reputation systems that abstract from many details. In this way, given a specific network environment, we can study the system behaviour to find the configuration that better meets the system requirements by tuning its parameters (reputation model, response timeouts, resource quality evaluation, ratings aging, etc.). Moreover, the analysed systems could be then directly used in the corresponding end-user applications (we will come back on this point in Section 4).

Summary of the rest of the paper. Section 2 describes the architecture and functional principles of our tool NEVER. Section 3 provides a brief overview of the tool component dealing with networking aspects. Finally, Section 4 concludes the paper by also reviewing some of the related work and suggesting directions for future work.

Further details on theoretical and implementation aspects of the reputation models currently implemented in the tool, as well as reports on performed analyses, appear in the companion technical report [5], which can be found on the NEVER web site [1].

2 The NEVER tool

In this section, we present the architecture and the workflow of NEVER (Network-aware EValuation Environment for Reputation systems), graphically depicted in Figure 2. The NEVER tool consists of three main components: (1) the experiment manager, (2) the network infrastructuring support, and (3) the reputation models library.

The *experiment manager* is the components playing the main role, because it is in charge of managing the execution of each experiment. An experiment consists of a user-specified number of runs, each run performed with the same configuration. The number of runs and their duration, together with other experiments characteristics, are defined by users through configuration parameters.

The *network infrastructuring support* provides the libraries (i.e., classes and interfaces) required to create and set up a Klava *net* (see Section 3) implementing the general infrastructure graphically depicted in Figure 1. Each element of the infrastructure is a node hosted by a (possibly remote and/or virtual) machine. The NEVER tool takes as input the addresses of the hosting machines and automatically activates the nodes forming the wanted network infrastructure. We refer to Section 3 for further details on the network infrastructure library supporting our experiments.

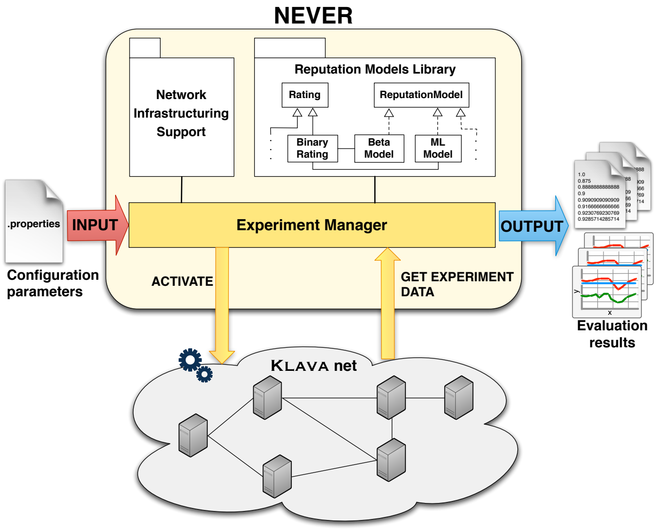

Fig. 2: NEVER architecture and workflow

The *reputation models library* acts as a framework allowing the user to define the trust and reputation models under evaluation. The library is a Java package containing a number of abstract classes and interfaces necessary to implement the models. In this way, the NEVER tool is customizable and extendible by the user. Specifically, a reputation model is defined by a class implementing the `ReputationModel` interface and, possibly, a class extending the abstract class `Rating`. The former class defines how reputation scores are computed, which rating values are used by the system and how parties in the system evaluate interactions. The latter class defines the kind of rating values and how to manage them. Thus, the addition of new reputation models to NEVER can be achieved by implementing `ReputationModel` and, if necessary, by extending `Rating`.

We describe now the NEVER workflow, by lingering on the main features of the experiment manager component. The tool takes as input a set of *configuration parameters*, written in a *.properties* file as pairs of the form *key* = *value*. Such parameters are used by the experiment manager to instantiate and carry out an experiment. First, the manager creates the network on top of which will be run the experiment. A node is created for each of the two servers and for each party in the system. Once the network is set up, the reputation system (configured according to user's parameters) is deployed on the network and the experiment starts, i.e. network components are enabled and system parties interact and rate each other. During the activity of the network, data about interactions are stored in appropriate files for a later analysis. Experiment runs are repeated in order to reach the desired precision; thus, the manager starts and stops runs till the last run is accomplished. Afterwards, data are analysed and provided as output, both in form of textual files and charts.

We conclude this section by commenting on the relevant configuration parameters. Through such parameters it is possible to specify the number of parties in the system and the addresses of the machines where parties have to run. For each party, a new Klava node is automatically created and deployed in the associated hosting machine. The tool also supports a 'local only' modality, where all Klava nodes are deployed in the same machine running the tool. Such modality can be useful to compare reputation systems in presence or absence of networking aspects affecting the evaluation.

A specific configuration parameter is used to set the main reputation model, which is used during the experiment to drive the interactions among parties. In fact, when a party is looking for a provider of a specific resource, it computes the providers' reputations and selects for the interaction the most trusted one, i.e. the party (or one of the parties) with the highest reputation value. Besides the main model, it is possible to give a list of trust and reputation models to be compared during the experiment: each party's reputation is computed according to all models specified in such list. Values of party's reputation are returned for each run and, at the end of the experiment, as a mean value over all runs. Moreover, the user can require to randomly select the providers, by thus ignoring the choice of the providers based on the main reputation model. Such modality is indeed often used in our experiments, because it gives the opportunity of evaluating models performances by comparing party reputations on the basis of approximately the same amount of ratings for each party.

A group of configuration parameters regulates parties' behaviour. The user specifies a set of possible party's behaviours and the percentage of parties with each given behaviour. Through such information, the experiment manager assigns a behaviour to each party. Moreover, it is possible to set parties' initial reputation by specifying the values and the number of their initial ratings. Such ratings determine the initial parties reputation computed by the system. In the default case, parties' behaviours are assumed to be fixed, but a changeable behaviour can be configured. In this case, the user sets when the variation has to happen and the magnitude of the variation. Currently, the variation implemented is negative, i.e. party's behaviour gets worse after variation. Several studies (see, e.g., [11, 16, 19]) use similar approaches for the evaluation of reputation models.

Finally, the configuration parameters allow the user to set two threshold values: the maximum delay and the maximum waiting time. The first parameter sets the maximum delay after which a resource is considered unsatisfactory, i.e. once the party receives the resource it checks if the arrival time exceeds the maximum delay and, in such a case, a negative rating is given to the provider no matter the quality of the resource. The second parameter sets the maximum time that a party will wait for a resource; expired this time a new provider is selected by the party and no rating value is given. In this way, a party will not wait indefinitely for a resource.

The NEVER tool is developed in Java, by exploiting freely available third-party libraries. Source and binary files of NEVER can be found at [1].

3 Network infrastructuring support

The network infrastructuring support of NEVER provides an API that allows the experiment manager to create different networks underlying the reputation systems to evaluate. To this aim, this tool component exploits the KLAVA library. In this section, we briefly introduce KLAVA and present the functionalities of each package component.

KLAVA. The Java library KLAVA [3] provides the run-time support for KLAIM actions within Java code. KLAIM [7] is a formal coordination language specifically designed for modelling mobile and distributed applications and their interactions, which run in a network environment. KLAIM provides communication primitives enabling tuple-based interaction à la Linda [9], which decouples the communicating processes both in space and time. Exchanged data are sequences of values, i.e. *tuples*. Communication is achieved via distributed multisets of tuples, called *tuple-spaces*, where processes insert, read and withdraw tuples. The data retrieving mechanism uses associative pattern-matching to find the required data in the tuple-space.

The network infrastructuring package. The network infrastructuring package specifies three different kinds of nodes that take part in the KLAVA net: a *rating server* node, a *search server* node and a *user* node. Each of these nodes implements a component of the infrastructure graphically depicted in Figure 1.

The rating server node serves as public database for collecting parties' ratings and executes the process `RatingServerProcess`. This process is in charge of collecting data produced by each experiment run.

The search server node assists parties while seeking a resource provider and executes the process `SearchServerProcess`. Such process waits for search requests sent by parties. Specifically, parties send requests to the server stating the type of the resource they want from the provider. Then, `SearchServerProcess` looks in the local tuple space for available providers offering such resource: for each provider matching the request, the process sends its address to the requesting party. The set of tuples sent to the party forms a list of provider addresses.

The user node implements a generic party; nodes of this kind interact to ask and provide resources and, after any interaction, rate each other. Two processes run on the user node[1]: the `ProviderProcess` and the `ClientProcess`. The former process implements the functionalities of a provider: when a new resource request coming from a client is received, the resource is selected and sent to the client. The resource selection consists of determining its quality according to the provider's behaviour; in fact, the actual provision of the resource is not relevant for our studies.

[1] Depending on the processes running in its node, a party can play the role of a client, of a provider, or both. We consider here the latter case, which is the most general.

The ClientProcess seeks providers for the resource it is looking for, and selects the most trusted one for the next interaction. Specifically, it determines the resource type it wants to request, asks the search server to find a provider for the given resource type and selects, among the providers returned by the search server, the most trusted one. Then, it checks if the reputation of such provider is higher than the minimum reputation value defined in the configuration file. If this check is positive, the process sends a request for the resource to the selected provider, otherwise it starts again the procedure from the beginning. Notably, the waiting time of a requested resource is bounded by a time-out specified in the configuration file. When the resource is received the process computes a rating value for the provider and sends it to the rating server.

4 Concluding remarks

In this paper we briefly presented NEVER, a network-aware tool for evaluating trust and reputation systems. The design of NEVER is based on the Klaim formal specification of trust and reputation system presented in [6]. We used the Java library Klava for implementing the models specified in Klaim. NEVER allows the rapid prototyping and testing of reputation system models in a real network environment, thus realizing a generic testbed for evaluating trust and reputation systems. We discussed the architecture of NEVER showing its logical structure.

In the companion technical report [5], we show how NEVER works by means of experimental data obtained through the evaluation of some of the implemented models. As an example, Figure 3 reports a graph produced as output by NEVER showing the reputation trends of four parties with respect to the number of available ratings for each of them.

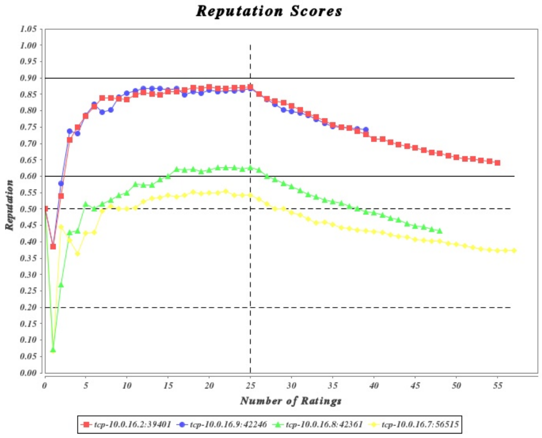

Fig. 3: Reputation trend of four parties

Related work. Among the many works in the literature whose goal is the evaluation and comparison of reputation systems, to the best of our knowledge, our contribution is the first effective tool allowing the evaluation in a real networked execution environment. Several works base their evaluation on a 'pen-and-paper' mathematical study of the models, without taking into account how they will be implemented and executed over distributed systems. For example, a formal framework for the comparison of probabilistic trust models, based on KL-divergence, is proposed in [18]. In this work KL-divergence is used as a measure of the quality of reputation functions. With the same purpose we exploit the notions of bayes and worst risk presented in [4]. NEVER computes empirical values of such risk functions for models set in the configuration file. Results of such computation are returned as output and are used for models evaluation.

Other works use simulation techniques for the evaluation of trust and reputation systems. For example in [14], a simulator implemented in Java is proposed as testbed (the ART testbed) enabling a competition forum for evaluating trust systems. In this case, no networking or other real world aspects are taken into account. Other examples of testbed are TREET [13] and the one proposed in [10]. The latter testbed is used for the evaluation of robustness of reputation systems. Specifically, this proposal focuses on robustness against unfair ratings, i.e. against parties that release scores that intentionally under-estimate interaction outcomes. The TREET testbed is proposed as an alternative to ART, which is considered not well-suited for general-purpose experimentation of reputation systems (it has, indeed, agents evaluation as its main purpose). Instead, TREET is specifically designed to support general-purpose experimentation and evaluation.

All these proposals are simulators or designs of testbeds that focus on marketplace applications. Our proposal, instead, does not fix a specific environment in which parties interact, but we use interactions as an abstraction of any parties relation. Moreover, we explicitly focus on probabilistic trust and reputation systems and on how they are evaluated. Our work aims at filling the gap between simulation and implementation of reputation systems, where networking aspects may play an important role when choosing and tuning trust and reputation systems. Indeed such aspects must be considered when implementing these systems. Specifically, problems such as how to rate parties when interactions are affected by network delays, or how to rate parties that are sporadically connected, have to be addressed. For this reason, reputation systems in NEVER are specified so that such problems can be taken into account by users when evaluating the systems. Indeed, they can be tuned on the basis of the features of the underlying network infrastructure exploited by NEVER for the execution.

Future work. We intend to continue our analysis programme by considering other reputation models proposed in the literature. Some of the models that we plan to consider in the near future are those surveyed in [17, 12].

Apart from considering richer reputation models, we intend to extend our investigation to reputation systems over network architectures that rely on distributed rating servers, rather than a single centralised one. Examples of such systems can be found in literature; where adaptations of trust models for decentralised architectures have been proposed. A reputation model adapted to ad-hoc networks for enhancing collaborations is proposed in [15]. For evaluating the relationships among devices in pervasive computing environments, a trust management scheme is introduced in [8], while [2] presents data structures and algorithms for assessing trust in a peer-to-peer environment. In particular, we intend to study how different underlying network architectures affects the performances of a given reputation system.

It is our intention to extend the tool to process real data from applications. The tool would be embedded in real applications and used to evaluate reputations systems in such environments. Applications could use reputation models in two different modalities: active or passive. In the active case, parties would compute reputation scores and use them to drive their interactions. In this modality the behaviour of an application would be modified by the deployed reputation system. In the passive case, the tool would collect rating values, compute reputation scores and just store them, without using such data to drive parties' interactions. The computed information would thus be used only for evaluating reputation systems. The passive modality would be useful in case of applications already deployed and in production. In this case it is important to understand how the application's behaviour would change before altering it. The passive modality could be also used for monitoring applications relying on existing reputation systems and contrast their reputation models with respect to the models implemented in our tool.

References

1. NEVER: Network-aware Evaluation Environment for Reputation Systems, 2013. Web site: `http://sysma.lab.imtlucca.it/tools/never/`.
2. K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *CIKM*, pages 310–317. ACM, 2001.
3. L. Bettini, R. De Nicola, and R. Pugliese. Klava: a Java Package for Distributed and Mobile Applications. *Software - Practice and Experience*, 32(14):1365–1394, 2002.
4. M. Boreale and A. Celestini. Asymptotic Risk Analysis for Trust and Reputation Systems. In *SOFSEM*, volume 7741 of *LNCS*, pages 169–181. Springer, 2013.
5. A. Celestini, R. De Nicola, and F. Tiezzi. Network-aware Evaluation Environment for Reputation Systems. CSA Technical Report #5/2013, IMT Institute for Advanced Studies Lucca, 2013. Available at `http://eprints.imtlucca.it/1537/`.
6. A. Celestini, R. De Nicola, and F. Tiezzi. Specifying and Analysing Reputation Systems with a Coordination Language. In *SAC*. ACM, 2013. To appear.
7. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *Transactions on Software Engineering*, 24(5):315–330, 1998.
8. M.K. Deno and T. Sun. Probabilistic trust management in pervasive computing. In *EUC*, volume 2, pages 610–615. IEEE Computer Society, 2008.
9. D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
10. A. A. Irissappane, S. Jiang, and Jie Zhang. Towards a comprehensive testbed to evaluate the robustness of reputation systems against unfair rating attack. In *UMAP Workshops'12*, 2012.
11. A. Jøsang and R. Ismail. The beta reputation system. In *Bled Conference on Electronic Commerce*, 2002.
12. A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.
13. R. Kerr and R. Cohen. TREET: the Trust and Reputation Experimentation and Evaluation Testbed. *Electronic Commerce Research*, 10:271–290, 2010.
14. K.K. Fullam et al. A specification of the Agent Reputation and Trust (ART) testbed: experimentation and competition for trust in agent societies. In *AAMAS*, pages 512–518. ACM, 2005.
15. C.T. Nguyen, O. Camp, and S. Loiseau. A bayesian network based trust model for improving collaboration in mobile ad hoc networks. In *RIVF*, pages 144 –151. IEEE, 2007.
16. J. Sabater and C. Sierra. Regret: reputation in gregarious societies. In *AGENTS*, pages 194–195. ACM, 2001.
17. J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24:3360, 2005.
18. V. Sassone, K. Krukow, and M. Nielsen. Towards a formal framework for computational trust. In *FMCO*, volume 4709 of *LNCS*, pages 175–184. Springer, 2006.
19. G. Zacharia and P. Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14(9):881–907, 2000.