Specifying and Analysing Reputation Systems with Coordination Languages

Alessandro Celestini, Rocco De Nicola, Francesco Tiezzi
IMT Advanced Studies Lucca
{alessandro.celestini,rocco.denicola,francesco.tiezzi}@imtlucca.it

ABSTRACT

Reputation systems are nowadays widely used to support decision making in networked systems. Parties in such systems rate each other and use shared ratings to compute reputation scores that drive their interactions. The existence of reputation systems with remarkable differences calls for formal approaches to their analysis. We present a verification methodology for reputation systems that is based on the use of the coordination language KLAIM and related analysis tools. First, we define a parametric KLAIM specification of a reputation system that can be instantiated with different reputation models. Then, we consider stochastic specification obtained by considering actions with random (exponentially distributed) duration. The resulting specification enables quantitative analysis of properties of the considered system. Feasibility and effectiveness of our proposal is demonstrated by reporting on the analysis of two reputation models.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Reputation systems; F.3 [**Theory of computation**]: Specifying and Verifying and Reasoning about Programs

Keywords

Reputation systems, Formal coordination languages, Stochastic analysis

1. INTRODUCTION

Trust and reputation systems are more and more used as tools to support decisions in several contexts, e.g. e-commerce applications, ad-hoc networks, sensor networks, P2P networks. Parties, which are willing to interact in these environments, are likely to be disconnected from their preferred security infrastructures and/or have no trusted information about their partners. Thus, they have to rely on other instruments to build up relationships of trust among each other. The best known applications where trust and reputation systems have been successfully used are related to e-commerce; examples in this context are auction sites, like eBay; online shops, like Amazon; and software application stores, like Google Play and Apple App Store.

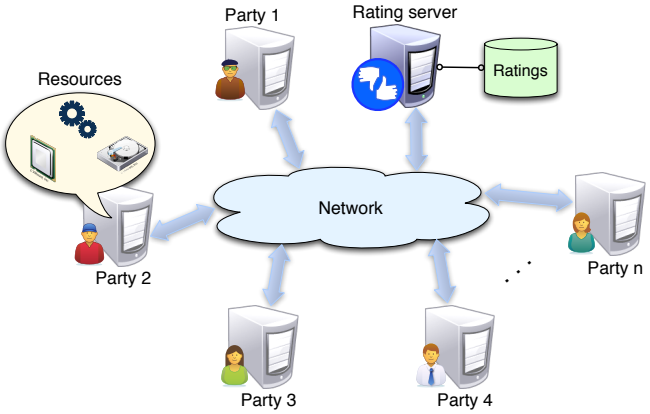

Figure 1: Infrastructure of a reputation system

In a reputation system, the involved parties rate each other, e.g. after completing an interaction, and use aggregated ratings about a given party to derive a *reputation score*, i.e. a collective measure of trustworthiness based on the ratings from the members of a community. Such value is used to assist other parties in deciding whether to interact with a specific partner.

A *networked trust infrastructure* allows parties of a reputation system to exchange ratings and interact. We consider a general infrastructure, graphically depicted in Figure 1, where a *rating server* collects all ratings from system's parties and makes them publicly available. Every *party* can play the role of a client, of a provider, or both, and may offer different kinds of resources (e.g., CPU time, disk space, files, services). Therefore, whenever a party needs a resource, it queries the rating server to determine the reputation of all parties providing the resource. Then, it selects one of the providers with the highest reputation score and, after the interaction, rates it according to the quality of the provided resource. Notably, we consider a centralised rating server, because this is the most widely used setting for networked trust infrastructures. On top of the general infrastructure described so far, different kinds of reputation system can be layered, which mainly differ for the model they use to aggregate ratings when computing reputation scores.

Due to the widespread use of reputation systems, research work on them is intensifying and several models have been

SAC'13 March 18-22, 2013, Coimbra, Portugal.

proposed. Thus, once a reputation system has to be deployed in a network environment, questions like the following may arise: Which reputation model is more suitable for the given environment? Does the model meet the expected behaviour? How does parties' behaviour affect their reputations? How do their initial reputation scores affect the model?

In this paper, we propose using a *coordination language* equipped with a formal semantics to deal with the above issues. On the one hand, in the last two decades coordination languages have been used for modelling and programming a variety of different systems that are operating in open and non-deterministic environments [14]. In particular, tuple-based languages (see, e.g., [15] for a survey) have been effectively used to implement coordination mechanisms in a distributed setting. On the other hand, formal methods are perfectly suitable means to precisely describe the relevant aspects of distributed systems, to state and prove their properties, and to direct attention towards issues that might otherwise be overlooked. Among the many coordination language proposals, here we focus on KLAIM [7], because it provides powerful coordination mechanisms based on the tuple-based communication model and, moreover, it is equipped with formal tools supporting verification and, most importantly, it has been specifically designed for network-aware programming of distributed applications. Thus, KLAIM appears to be well suited for specifying trust and reputation systems and for reasoning about them.

We proceed in three steps as follows:

1. We model the considered reputation system with KLAIM. Specifically, we provide a 'schema' specification that is parametric w.r.t. the reputation model used to determine the parties' reputation (and w.r.t. other parameters of the system). The specification of the given system is obtained by appropriately instantiating the parameters of the generic specification.

2. We enrich the specification with stochastic aspects, using the KLAIM's stochastic extension STOKLAIM [8], and formally express the desired properties using the stochastic logic MOSL [8].

3. We check the properties of interest against the STOKLAIM specification by means of the analysis tool SAM [8, 13].

We will focus on reputation models that use a probabilistic approach for computing reputation scores. This probabilistic trust approach is based on the definition of trust given by Gambetta [10]: parties' behaviour is modeled by a probability distribution and rating values are used for estimating distribution's parameters [11, 9]. To demonstrate the feasibility and effectiveness of our proposal, we analyse two models of reputation systems, namely the Beta model [11] and a model based on maximum likelihood estimation (which we call ML model) [9]. We show that, in the average, reputation scores in the latter model converge more rapidly to the right estimations of parties' behaviour than in the former one. Even though, the ML model is more unstable than the Beta model when few ratings are available.

2. BETA AND ML REPUTATION MODELS

In this section, we provide a few details about the two reputation system models taken into account in this paper: the first based on the Beta model [11], and the second based on maximum likelihood estimation (ML model) [9]. Before presenting them, we introduce their common background.

System parties rate each other in a binary way, i.e. an interaction can be either 'satisfactory' or 'unsatisfactory'. Let $\mathcal{P}$ be a set of party identities, the *behaviour* of each party $p \in \mathcal{P}$ is assumed to be probabilistic, in the sense that there is a fixed probability $\theta_p \in [0,1]$ that an interaction with the party p will be satisfactory. In a reputation system, the goal is to predict parties' behaviour in future interactions, given the rating values about past interactions, i.e. determining an estimation $\tilde{\theta}_p$ of the unknown parameters θ_p. The sequence of rating values $x_p^n = x_{1p}, \ldots, x_{np}$, with $x_{ip} \in \{0,1\}$, about past interactions with party p is considered as realization of a sequence of independent, identically distributed random variables $X_p^n = X_{1p}, \ldots, X_{np}$. The considered models assume that random variables X_{ip} are distributed according to a Bernoulli distribution with success probability θ_p. This means that, when interacting with a party p, the probability that the i-th interaction is satisfactory, given θ_p the behaviour of party p, is $Pr(\text{satisfactory} \mid \theta_p) = \theta_p$.

The **Beta model** seeks to estimate the a posterior distribution for the value θ_p, given the results of past interactions with party p. The model uses a conjugate prior distribution, specifically a *beta prior*. Hence, the a posterior distribution results in a beta distribution. Party's reputation score $\tilde{\theta}_p$ is given by the expected value of the beta distribution $Beta(\alpha+1, \beta+1)$ with $\alpha \geq 0$ and $\beta \geq 0$, where parameter α represents the number of satisfactory past interactions with party p and β represents the unsatisfactory interactions with p.

The **ML model** seeks for a value $\tilde{\theta}_p$ which maximises the likelihood expression $L(\theta)$:

$$L(\theta) = Pr(X_p^n \mid \theta) = \prod_{i=1}^{n} Pr(X_{ip} = x_{ip} \mid \theta)$$

The party's reputation score $\tilde{\theta}_p$ is the $\theta \in [0,1]$ that maximises the likelihood $L(\theta)$.

3. KLAIM AND RELATED STOCHASTIC VERIFICATION TOOLS

In this section, we informally present KLAIM, a tuple-space-based coordination language specifically designed for modelling mobile and distributed applications and their interactions, which run in a network environment. Then, we provide a brief overview of the KLAIM-based formal methods that can be exploited for specifying and verifying reputation systems. We refer the interested reader to [7] for a more complete account of KLAIM, and to [4] for a survey on research work based on KLAIM. Notably, we consider here a version of KLAIM enriched with standard control flow constructs (i.e., if-then-else, sequence, while, etc.), that are part of the input language of the analysis tool used in Section 5. These constructs simplify specifications and can be easily rendered in the language originally presented in [7]. When presenting the language, we omit describing the constructs that are not used in the specification introduced in the next section.

KLAIM specifications consist of *nets*, namely finite plain collections of nodes where *components*, i.e. processes P and data tuples $\langle t \rangle$, can be allocated. Nodes are composed by means of the parallel composition operator $\_ \| \_$.

Nodes have the form $s :: C$, where s is a unique *locality name* (i.e., a network address) and C is a set of hosted

components. During processes execution, *locality variables* l (i.e., aliases for addresses) are bound to localities s. The distinguished locality variable **self** is used by processes to refer to the address of their current hosting node. In this section, we will use ℓ to range over locality names and variables.

Processes are the KLAIM active computational units and may be executed concurrently either at the same locality or at different localities. They are built up from basic actions (see below) and process calls $A(p_1, \ldots, p_m)$ by means of sequential composition $\_;\_$, parallel composition $\_ \mid \_$, conditional choice **if** (e) **then** $\{\_\}$ **else** $\{\_\}$, iterative constructs **for** $i = n$ **to** m $\{\_\}$ and **while** (e) $\{\_\}$, and (possibly recursive) process definitions $A(f_1, \ldots, f_n) \triangleq \_$, where A is a *process identifier* and the formal parameters f_i are pairwise distinct. Notably, e ranges over *expressions*, which are formed from basic values (booleans, integers, strings, floats, etc.) and variables, by using standard operators on values.

During their execution, processes perform some basic *actions*. Actions $\mathbf{in}(T)@\ell$ and $\mathbf{read}(T)@\ell$ are retrieval actions and permit, respectively, to withdraw and read *data tuples* (i.e. sequences of values) from the tuple space hosted at the (possibly remote) locality ℓ: if some matching tuples are found, one is non-deterministically chosen, otherwise the process is blocked. These actions exploit templates as patterns to select tuples in shared tuple spaces. *Templates* T are sequences of actual and formal fields, where the latter are written $!\,x$ or $!\,l$ and are used to bind variables to values or locality names, respectively. Action $\mathbf{out}(t)@\ell$ adds the tuple resulting from the evaluation of tuple t (which may contain expressions) to the tuple space of the target node identified by ℓ, while action $\mathbf{eval}(P)@\ell$ sends the process P for execution to the (possibly remote) node identified by ℓ. Actions **out** and **eval** are both non-blocking. Finally, action $x := e$ assigns the value of e to x; this action, differently from all the others, is not indexed with an address because it always acts locally.

Quantitative analysis of a KLAIM specification can be enabled by associating a *rate* to each action, thus obtaining a STOKLAIM [8] specification. This rate is the parameter of an exponentially distributed random variable accounting for the action duration time. A real valued random variable X has a *negative exponential distribution* with rate $\lambda > 0$ if and only if the probability that $X \leq t$, with $t > 0$, is $1 - e^{-\lambda \cdot t}$. The expected value of X is λ^{-1}, while its variance is λ^{-2}. The operational semantics of STOKLAIM permits associating to each specification a Continuous Time Markov Chain that can be used to perform quantitative analyses of the considered system. The use of the exponential distribution is motivated by the fact that it enjoys convenient properties enabling automated analyses that are not always allowed by other distributions.

The desired properties of a system under verification are formalised using the stochastic logic MOSL [8]. MOSL formulae use predicates on the tuples located in the considered STOKLAIM net to express the reachability of a certain system state, while passing through, or avoiding, other specific intermediate states. Therefore, MOSL can be used to express quantitative properties of the overall system behaviour, to ask, e.g., whether the reputations of all parties converge (in a *steady state*) to the corresponding parties' behaviours. The results of the evaluation of such properties do not have a rigid meaning, like *true* or *false*, but have a less absolute nature, e.g. *in 97.5% of the cases, the reputations converge within t time units.*

Verification of MOSL formulae over STOKLAIM specifications is assisted by the analysis tool SAM [8, 13], which uses a statistical model checking algorithm [5] to estimate the probability of the property satisfaction. In particular, the probability associated to a path-formula is determined after a set of independent observations.

4. FORMAL SPECIFICATION OF REPUTATION SYSTEMS

In this section, we present the relevant aspects of the KLAIM specification of a reputation system deployed in the general infrastructure described in Section 1 and graphically depicted in Figure 1. To help readability, the specification code reported in this section is sometimes accompanied by comments (strings preceded by // indicates that the rest of the line is a comment).

The overall system can be rendered in KLAIM as follows:

$$
\begin{array}{l}
s_{rating} :: \langle \text{“}ratingList\text{”}, s_{party\_1}, m_1 \rangle \\
\qquad\quad \mid \langle \text{“}rating\text{”}, 1, s_{party\_1}, s_{rater}, v_{rating} \rangle \\
\qquad\quad \mid \ \ldots \ \mid \langle \text{“}rating\text{”}, m_1, s_{party\_1}, s'_{rater}, v'_{rating} \rangle \\
\qquad\quad \ldots \\
\qquad\quad \mid \langle \text{“}ratingList\text{”}, s_{party\_n}, m_n \rangle \\
\qquad\quad \mid \langle \text{“}rating\text{”}, 1, s_{party\_n}, s''_{rater}, v''_{rating} \rangle \\
\qquad\quad \mid \ \ldots \ \mid \langle \text{“}rating\text{”}, m_n, s_{party\_n}, s'''_{rater}, v'''_{rating} \rangle \\
\parallel \\
s_{party\_1} :: A_{party}(behaviour_1) \ \mid \ C_{providerList\_1} \\
\parallel \\
\ldots \\
\parallel \\
s_{party\_n} :: A_{party}(behaviour_n) \ \mid \ C_{providerList\_n}
\end{array}
$$

Each system party i is rendered as a KLAIM node whose locality name is s_{party_i} and, similarly, the rating server is rendered as a node, with locality name s_{rating}, as well.

The rating server provides the list of ratings concerning each party. Such list is rendered in KLAIM as a set of tuples of the form $\langle \text{“}rating\text{”}, i, s_{party\_j}, s_{rater}, v_{rating} \rangle$ indicating that s_{party_j} received the rating value v_{rating} from s_{rater}, such rating value is the i-th element of the list concerning s_{party_j}. A tuple of the form $\langle \text{“}ratingList\text{”}, s_{party\_j}, m_j \rangle$, denoting that the list for s_{party_j} has length m_j, is used to read the whole list. Such tuple is used also for coordinating read and write operations on the list: actions **in**, **out** and **read** act on this tuple by implementing a readers-writers lock mechanism. Finally, each party node s_{party_j} contains some tuples $C_{providerList\_j}$ representing the list of providers known by the party.

Depending on the processes running in its node, each party s_{party_j} can play different roles: it can provide resources, require resources, or both. We consider here the more complete case, where A_{party} is defined as

$$A_{party}(b) \ \triangleq \ A_{provider}(b) \ \mid \ A_{client}(b)$$

where the parameter b denotes the party's behaviour (for the analyses carried out in this paper b is the probability θ_p of the party defined in Section 2). The process for the provider role is defined as follows:

$$
\begin{array}{l}
A_{provider}(b) \ \triangleq \\
\quad \text{// wait for a new resource request} \\
\quad \mathbf{in}(\text{“}request\text{”}, !l_{applicant})@\mathbf{self}; \\
\quad \text{// provide the resource and restart} \\
\quad (\, A_{provideResource}(b, l_{applicant}) \ \mid \ A_{provider}(b) \,)
\end{array}
$$

The processing of a resource request is based on the definition of process $A_{provideResource}$, which provides to the applicant a resource whose quality depends on provider's behaviour. The definition of such process may vary from a reputation system to another. In fact, this is a parameter that must be specified when considering a specific reputation system.

The process for the client role, instead, is defined as follows:

$A_{client}(b) \triangleq$
  // initialise the tuple containing the locality of the
  // most trusted party and its reputation value
  out("*mostTrusted*", **self**, NO_ONE)@**self**;
  // read the list of known providers
  read("*providerList*", !m)@**self**;
  for $j = 1$ **to** m {
    // get an element of the list
    read("*provider*", j, !$l_{provider}$)@**self**;
    // compute the reputation of the j-th provider
    $A_{evaluateReputation}(l_{provider})$;
    // retrieve the computed reputation value
    in("*reputation*", !*rep*)@**self**;
    // update the most trusted party
    in("*mostTrusted*", !$l_{trusted}$, !*repMT*)@**self**;
    if ($repMT < rep$) **then**{
      out("*mostTrusted*", $l_{provider}, rep$)@**self**
    } **else** {
      out("*mostTrusted*", $l_{trusted}, repMT$)@**self**
    };
  };
  // get the most trusted provider
  in("*mostTrusted*", !$l_{trusted}$, !*repMT*)@**self**;
  // check the reputation of the most trusted provider
  if (MIN_REPUTATION $\leqslant repMT$) **then**{
    // send the resource request to the provider
    out("*request*", **self**)@$l_{trusted}$;
    // receive the resource
    in("*resource*", !*quality*)@**self**;
    // check resource's quality and rate the provider
    $A_{rate}(b, quality, l_{trusted})$
  };
  $A_{client}(b)$

A client cyclically determines the most trustworthy provider and requests a resource to it. To this aim, the client computes the reputation of each known provider (stored in a local list) through the process $A_{evaluateReputation}$, whose skeleton definition is as follows:

$A_{evaluateReputation}(l) \triangleq$
  ... possible variables initialisation ...
  // read the rating values of the provider l
  read("*ratingList*", l, !m)@s_{rating};
  for $j = 1$ **to** m {
    // get an element of the list
    read("*rating*", j, l, !l_{rater}, !*rating*)@s_{rating};
    ... use *rating* to compute the reputation of l ...
  };
  ... compute reputation ...
  out("*reputation*", *rep*)@**self**

The above process differs from a reputation system to another and is, indeed, a parameter that must be specified to consider a given reputation system. Similarly, the constant MIN_REPUTATION is a system's parameter and specifies the minimum reputation required by the client for an interaction with a provider. Instead, the constant NO_ONE simply denotes that there exists no provider for a requested resource. Finally, the process A_{rate} evaluates the quality of the obtained resource and rates the provider accordingly. As in the case of process $A_{evaluateReputation}$, its definition may differ from a reputation system to another.

The KLAIM processes defining the two reputation models considered in this paper are reported in [6].

5. STOCHASTIC SPECIFICATION AND ANALYSIS

In this section, we demonstrate how the KLAIM specification presented in the previous section can support the analysis of trust and reputation systems. Our approach relies on formal tools, such as stochastic simulation, modal logics and model checking, that permit expressing and evaluating performance measures in terms of logical formulae.

We enrich the KLAIM specification introduced in the previous section with stochastic aspects. As an excerpt of the STOKLAIM specification, we report below the stochastic definition of process $A_{evaluateReputation}$:

$A_{evaluateReputation}(l) \triangleq$
  ...
  read("*ratingList*", l, !m)@$s_{rating} : \lambda_1$;
  for $j = 1$ **to** m {
    read("*rating*", j, l, !l_{rater}, !*rating*)@$s_{rating} : \lambda_1$;
    ...
  };
  ...;
  out("*reputation*", *rep*)@**self** : λ_2

The actions highlighted by a gray background are those annotated with rates, where $\lambda_1 = 37.0$ and $\lambda_2 = 1400.0$. These rates assume that an ADSL connection (1.5 Mbit/s downstream and 0.5 Mbit/s upstream) is used and that the operation of reading a rating costs as the transfer of 5KB of data. For the local writing of a reputation value, we assume that the operation is executed on a local hard drive. We refer the interested reader to [2] for the rest of the stochastic specification.

The result of some simulation runs of the STOKLAIM specification, performed by using SAM, are reported in Figures 2 and 3. The charts present the trend of the reputation of a given party in the system; the results are averaged across 1000 simulation runs[1]. On the x-axis we find the numbers of rating values used to compute reputation scores, on the y-axis the reputation scores. The behaviour of the party is $\theta = 0.75$, represented in the charts as an horizontal line. Instead, the trends of his reputation scores calculated by using Beta and ML models are indicated by the polygonal lines. In Figure 2 we assume that each party starts with no ratings, instead in Figure 3 we fix an initial reputation score of 0.5 for each party. The reputation, in this case, is computed by means of 8 rating values (4 positive and 4 negative) already stored in the system for each party. The charts show that in the average the ML model converges more rapidly

[1]On an Apple MacBook Pro computer (2.4 GHz Intel Core 2 Duo and 4 GB of memory) simulation of a single run needs an average time of 0.04 seconds.

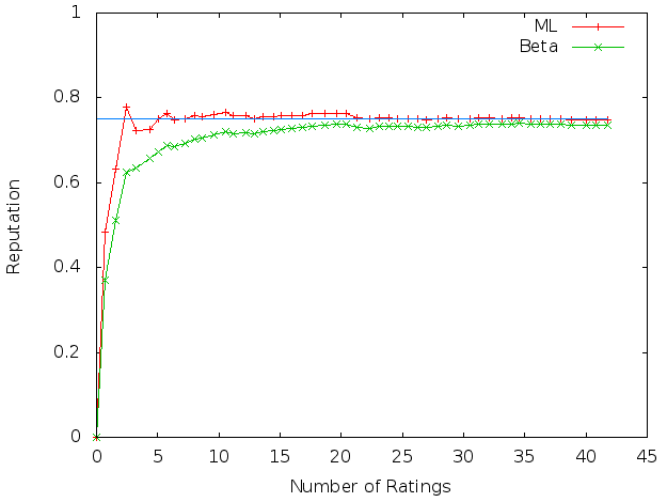

Figure 2: Reputation trends for party with behaviour $\theta = 0.75$ and no initial ratings assigned

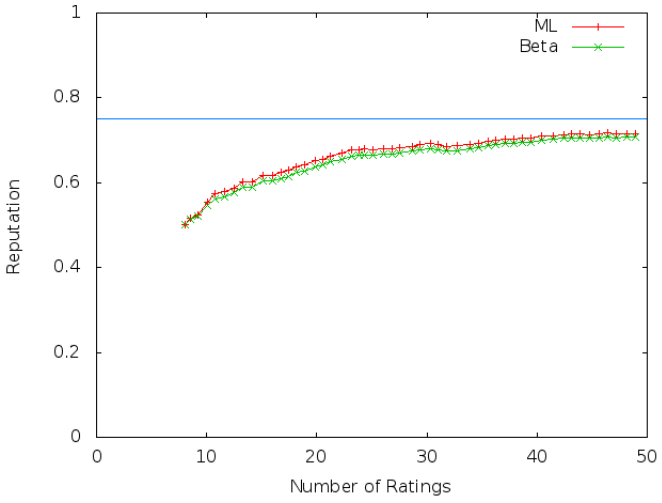

Figure 3: Reputation trends for party with behaviour $\theta = 0.75$ and 8 initial ratings assigned. The initial ratings fix party's reputation to 0.5

to the right estimation of party's behaviour than the Beta model. In the case of the Beta model, convergence is slower but smoother.

We performed similar experiments with different numbers of ratings. Such simulations show that, in presence of initial ratings, party's reputation score does not change brusquely when a new rating value is provided as in the absence of initial ratings. When the number of ratings increases, the reputation score takes more time to converge to party's behaviour, because the initial reputation is more solid. Other experiments show that, if party's behaviour is higher than 0.50, the Beta model underestimates it, i.e. party's reputation score is always lower than party's behaviour. Instead, when the behaviour is lower than 0.50, the Beta model overestimates it.

We have also analysed some properties of reputation systems by formalising them as MoSL formulae that we have verified over the StoKlaim specification by means of the SAM tool. The property "*the reputation of* s_{party_i} *is currently within* $[\theta_i + \delta, \theta_i - \delta]$" is expressed in MoSL by the following formula ϕ_{conv}:

$$\phi_{conv} = \langle \text{“}reputation\text{”}, s_{party\_i}, !rep \rangle @ s_{rating} \rightarrow (\theta_i + \delta \geq rep \quad \wedge \quad rep \geq \theta_i - \delta)$$

This formula relies on the *consumption* operator $\langle T \rangle @ s \rightarrow \phi$, which is satisfied whenever a tuple matching template T is located at s and the remaining part of the system satisfies ϕ. Hence, formula ϕ_{conv} is satisfied if and only if a tuple $\langle \text{“}reputation\text{”}, s_{party\_i}, v_{rep} \rangle$ is stored in the node s_{rating} and the reputation value v_{rep} is equal to the party's behaviour θ_i up to a given error δ. Notice that the Klaim model of the reputation system has been slightly modified to enable this analysis. In particular, the client process updates tuples of the form $\langle \text{“}reputation\text{”}, s_{party\_i}, v_{rep} \rangle$ in the rating server every time it computes a provider's reputation score. Thus, the property "*the reputation of* s_{party_i} *converges to his actual behaviour within time* t" is defined as *true* $U^{\leq t} \phi_{conv}$, where the *until* formula $\phi_1 U^{\leq t} \phi_2$ is satisfied by all the runs that reach within t time units a state satisfying ϕ_2 while only traversing states that satisfy ϕ_1. The model checking analysis has been then performed by estimating the total probability of the set of runs satisfying such formula (the maximal time t has been set to 50 seconds and δ to 0.1). Considering party's behaviour $\theta = 0.75$, we get that the formulae is satisfied, in the Beta model, with probability 0.8587 and with probability 0.7994 in the ML model. The average amount of ratings available for the computation after 50 seconds is 43. The same formula has been checked in case of a fixed initial reputation score (computed by means of 8 ratings). In this case we get probability 0.3972 for the Beta model and 0.4982 for the ML model. The results of the model checking confirm the simulation outcomes, i.e. in the average the ML model estimation is better than that of Beta model. In addition, such results point out that the ML model is less stable than the Beta model when few ratings are available. Notably, all these values have been estimated with both error probability and tolerance set to 0.1, which require 1198 simulation runs.

Similarly, the property "*the reputation of* s_{party_i} *goes below a given threshold within time* t" is formalised as

$$true\ U^{\leq t} \left(\begin{array}{l} \langle \text{“}reputation\text{”}, s_{party\_i}, !rep \rangle @ s_{rating} \\ \rightarrow threshold \geq rep \end{array} \right)$$

The threshold may represent, e.g., the minimum reputation that a provider must have in order to be considered trustworthy for an interaction. Considering party's behaviour $\theta = 0.25$ and $threshold = 0.35$, we get that the formulae is satisfied, in the Beta model, with probability 0.9914 and with probability 0.9947 in the ML model. The same formula has been checked in case of an initial reputation score fixed to 0.5 (using 8 ratings), by using the same parameters as before for the estimations; we get probability 0.9246 for the Beta model and 0.9446 for the ML model. Setting an initial reputation score slows down achieving the threshold.

6. CONCLUDING REMARKS

In the last two decades, coordination languages have been extensively exploited in different domains to support development and analysis of concurrent and distributed systems. In this paper, by relying on one of these languages, namely Klaim, we have shown how coordination languages and formal methods can be beneficial to the field of reputation systems. More specifically, we have illustrated how

such systems can be specified with KLAIM and analysed with KLAIM-based stochastic tools.

Related and future work. The terminology used in the literature to describe the systems considered in this paper is sometimes quite confusing, due to the usage of the term *trust* in different contexts with a variety of meanings. In fact, trust and reputation are often used as synonyms. The difference between the two concepts is clarified in [12]. According to this survey, trust is based on a subjective measure of reliability of a given party, derived from some private knowledge (e.g. past direct interactions). Reputation, instead, relies on an objective measure derived from referrals or ratings provided by other parties. Therefore, by adopting such distinction, our work mainly focus on reputation.

There are many works in the literature whose goal is the evaluation and comparison of reputation systems. However, to the best of our knowledge, our contribution is the first study based on the use of a formal coordination language.

Some related works make use of computational software programs [19] or software for multi-agent modelling [18], for simulating their reputation models. As an example, in [17] the RePast simulator [1] is used to conduct various studies on reputation systems in a multi-agent context. Such studies mainly differ from ours because we rely on formal methods' tools, such as coordination languages, stochastic modal logics, simulation and model checking, that permit expressing and evaluating performance measures in terms of logical formulae. This is a strong conceptual framework that provides abstraction primitives for conveniently specifying reputation systems and their properties. Stochastic modelling and verification tools permit considering the typical *uncertainty* of real systems. In this way, the analysis can cover more relevant situations of the considered system.

In this paper, for the sake of presentation, we have taken into account two reputation system models, i.e. the Beta model and the ML model. The proposed approach can be extended to other reputation models proposed in the literature (e.g., those surveyed in [16, 12]); we intend to do that as a future work. We want also to consider attacks to reputation systems, e.g. by cheating raters or through purchase of reviews. This, of course, will raise the issue of how to model these behaviours within our proposal.

Another line of research we are exploring is the experimentation with reputation systems in real networked execution environments, by deploying dedicated KLAIM code on each node. Specifically, we are developing a software tool for rapid prototyping and testing reputation system models. To this aim, we will implement the KLAIM model introduced in this paper by means of KLAVA [3], a Java library providing a run-time support for KLAIM actions within Java code. This would bring many benefits. Firstly, the Java implementations obtained with such framework would be driven by a formal model, which would be initially verified at an abstract level by means of formal tools. Then, the system would be analysed by means of practical experiments in order to take into account networking and other real world aspects.

Acknowledgments

We would like to thank Michele Loreti for the fruitful discussions and his support while using the SAM tool.

7. REFERENCES

[1] RePast. Web site: `http://repast.sourceforge.net`.

[2] Specification and Analysis of Reputation Systems in SAM, 2012. SAM source files available at `http://cse.lab.imtlucca.it/rep_sys_eval/SpecAndAnalysisOfRepSysSAM.zip`.

[3] L. Bettini, R. De Nicola, and R. Pugliese. Klava: a Java Package for Distributed and Mobile Applications. *Softw. - Pract. and Exper.*, 32(14):1365–1394, 2002.

[4] L. Bettini et al. The Klaim Project: Theory and Practice. In *Global Computing*, LNCS 2874, pages 88–150. Springer, 2003.

[5] F. Calzolai and M. Loreti. Simulation and Analysis of Distributed Systems in Klaim. In *COORDINATION*, LNCS 6116, pages 122–136. Springer, 2010.

[6] A. Celestini, R. De Nicola, and F. Tiezzi. Specifying and analysing reputation systems with coordination languages (full version). Technical report, IMT Advanced Studies Lucca, 2013. `http://cse.lab.imtlucca.it/rep_sys_eval/sac2013_TR.pdf`.

[7] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *Trans. on Software Engineering*, 24(5):315–330, 1998.

[8] R. De Nicola, J. Katoen, D. Latella, M. Loreti, and M. Massink. Model checking mobile stochastic logic. *Theor. Comput. Sci.*, 382(1):42–70, 2007.

[9] Z. Despotovic and K.Aberer. A Probabilistic Approach to Predict Peers' Performance in P2P Networks. In *CIA*, LNCS 3191, pages 62–76. Springer, 2004.

[10] D. Gambetta. *Trust: Making and Breaking Cooperative Relations*, chapter 13: Can We Trust Trust?, pages 213–237. Basil Blackwell, 1988.

[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] M. Loreti. SAM: Stochastic Analyser for Mobility, 2010. Available at `http://rap.dsi.unifi.it/SAM/`.

[14] A. Omicini and M. Viroli. Coordination models and languages: from parallel computing to self-organisation. *Knowledge Eng. Review*, 26(1):53–59, 2011.

[15] D. Rossi, G. Cabri, and E. Denti. Tuple-based technologies for coordination. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 83–109. Springer, 2001.

[16] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24:33–60, 2005.

[17] A. Schlosser, M. Voss, and L. Brückner. Comparing and Evaluating Metrics for Reputation Systems by Simulation. In *Work. on Reput. in Agent Soc.*, 2004.

[18] Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. In *P2P*, pages 150–157. IEEE, 2003.

[19] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer e-commerce communities. In *CEC*, pages 275–284. IEEE, 2003.