A Calculus for Trust and Reputation Systems

Alessandro Aldini

Department of Base Science and Fundamentals, University of Urbino, Italy
alessandro.aldini@uniurb.it

Abstract. Trust and reputation models provide soft-security mechanisms that can be used to induce cooperative behaviors in user-centric communities in which user-generated services and resources are shared. The effectiveness of such models depends on several, orthogonal aspects that make their analysis a challenging issue. This paper aims to provide support to the design of trust and reputation infrastructures and to verify their adequacy in the setting of software architectures and computer networks underlying online communities. This is done by proposing a formal framework encompassing a calculus of concurrent systems, a temporal logic for trust, and model checking techniques.

1 Introduction

Cooperation is a key factor for the success of service- and user-centric networks in which user-generated contents are exchanged, remote resources are shared, and services provided by third parties are executed online. Trust and reputation systems provide extrinsic motivations to favor cooperation in spite of selfishness, malicious behaviors, and mistrust towards unknown users. The metrics provided by these systems help to estimate quantitatively the subjective reliance on the ability, integrity, honesty and disposition of each user, to be shared within the community with the aim of making explicit a collective notion of reputation. Even more important, reputation is defined not only to give a perception of the public trustworthiness of users, but also to provide enabling conditions for participating actively in the community by exchanging services and resources.

The design and implementation of trust and reputation systems is not an easy task as it depends on several, orthogonal aspects. Solutions can be centralized or distributed, can rely (or not) on the presence of a trusted third party, can use first-hand or second-hand reputation systems using (non-)linear adjustment mechanisms, can involve explicit (based, e.g., on voting) or implicit evaluation means, and so on [14]. As a consequence, the analysis of the effectiveness of these systems against the typical obstacles to cooperation (lack of motivation, selfishness, free-riding, ...) and the variety of attacks (slandering, self-promoting, sybil, ...) is a challenging issue.

In this paper, we propose a framework for:

- the formal modeling of the behavior of cooperative, concurrent, and distributed systems;

J. Zhou et al. (Eds.): IFIPTM 2014, IFIP AICT 430, pp. 173–188, 2014.

- the formal specification of trust and reputation infrastructures governing the interactions in these systems;
- the formal verification of the effectiveness of the trust policies adopted by these infrastructures to stimulate cooperation and to contrast attacks.

These objectives are achieved by means of a process algebraic approach to software architecture design, in which functional modeling (through typical process algebraic operators) and specification of the trust model are defined separately at the syntactic level and joined automatically at the semantic level. This separation of concerns facilitates all the design issues and the execution of sensitivity analysis aimed at evaluating the effects of the chosen system architecture and trust model. The formal specification of trust-based properties relies on a temporal logic for trust that extends classical state-based and action-based logics, while the verification of such properties is supported by model checking techniques.

In the rest of the paper, we first introduce a real-world case study, which accompanies the presentation of the formal framework as a running example through which we show how to apply our approach in practice. Afterwards, we present the syntax for a calculus of concurrent processes and the syntax for a specification language of trust systems. We then define a unifying formal semantics, which subsumes the definition of specific labeled state-transition systems, and the temporal logic for specifying trust properties that can be model checked through standard techniques. Conclusions about related work and future directions terminate the paper.

1.1 Running Example

As a real-world example, we consider an incentive-based cooperation model for wireless and mobile user-centric environments recently proposed [6]. Basically, cooperative networks involve users providing services, called *requestees*, and recipients of such services, called *requesters*. For the sake of simplicity, we assume that each user behaves either as requester or as requestee. The cooperation model is based on soft-security mechanisms (like trust management and virtual currency) and a process entailing four phases: (*i*) the requester looks for a service in the community and then sends a request to the chosen requestee; (*ii*) the two parties negotiate parameters and cost of the transaction; (*iii*) if an agreement is reached, the requestee provides the negotiated service and the requester pays for it; (*iv*) both parties evaluate the quality of experience and provide feedback. In each phase, trust is used to govern choices and to provide incentives for both parties, e.g., by making offered quality of service and related cost directly dependent on trust. The objective consists of inducing a prosocial attitude to collaboration while isolating selfish and cheating behaviors.

In the following, we abstract away from the details of the specific incentive strategies and we concentrate on showing how to employ our approach in order to model a scenario like the one surveyed above, specify the underlying trust and reputation models, and perform model checking based sensitivity analysis aiming at demonstrating the influence of each policy and configuration parameter chosen by the involved parties.

2 Modeling Trust Systems

In this section we show how to define separately functional behavior of the system and trust infrastructure. In both cases, we present formal syntax, semantics, and examples related to our running case study.

By following principles inspired by architectural description languages [3], we distinguish between *process behaviors*, which describe behavioral patterns, and *process instances*, which represent specific entities exhibiting a certain behavioral pattern, as well as we separate the definition of individual entities from the specification of their parallel composition and communication interfaces. This separation of concerns is applied also to distinguish the description of a system of interacting entities from the specification of the reputation infrastructure governing any interaction based on trust. The objective is an improvement of usability concerning the modeling issues of the different aspects that come into play in the specification of trust-based distributed systems.

2.1 Modeling Individual Processes

We start by introducing a calculus for the specification of individual process terms, which represent process behaviors modeling behavioral patterns. We denote with *Name* the set of visible action names, ranged over by $a, b, \ldots$. Moreover, we assume a special name τ to denote invisible, internal actions.

The set of process terms of our calculus is generated through the following syntax:

$$P ::= \underline{0} \mid a\,.\,P \mid \tau\,.\,P \mid P + Q \mid a.P \mp b.Q \mid B$$

where:

- $\underline{0}$ represents the inactive, terminated process term.
- $a\,.\,P$ (resp., $\tau\,.\,P$) denotes the process term that executes a (resp., τ) followed by the behavior of P.
- $P + Q$ represents a nondeterministic choice between process terms P and Q.
- $a.P \mp b.Q$, which is called *trusted choice* operator, denotes an external, guarded choice based on trust.
- B represents a process constant equipped with a defining equation of the form $B \stackrel{\text{def}}{=} P$, which establishes that process constant B behaves as process term P, thus enabling the possibility of defining recursive behaviors.

In the following, we restrict ourselves to consider guarded process terms, i.e., all of the (finite) occurrences of process constants are immediately preceded by the action prefix operator. Before detailing the interpretation of these operators, we introduce the underlying semantic model, which is based on classical labeled transition systems.

Definition 1. *A labeled transition system (lts) is a tuple (Q, q_0, L, R) where Q is a finite set of states, of which q_0 represents the initial one, L is a finite set of labels, and $R \subseteq Q \times L \times Q$ is a finitely-branching transition relation, such that $(p, l, q) \in R$ is denoted by $p \stackrel{l}{\longrightarrow} q$.*

Given $Act = \{\tau\} \cup Name \cup \{a^- \mid a \in Name\}$, which is ranged over by $\alpha, \ldots$, the behavior of a process term P is defined by the smallest lts $[\![P]\!]$ such that the states in Q represent process terms (with P being the initial state q_0), the labels in L are given by actions in the set Act, and the transitions in R are obtained through the application of the following operational semantic rules:

$$\textit{prefix} \qquad a\,.\,P \xrightarrow{a} P \qquad\qquad \tau\,.\,P \xrightarrow{\tau} P$$

$$\textit{choice} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad\qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\textit{trusted choice} \qquad a.P \mp b.Q \xrightarrow{a} P \qquad\qquad a.P \mp b.Q \xrightarrow{b^-} Q$$

$$\textit{recursion} \qquad B \stackrel{\text{def}}{=} P \quad \frac{P \xrightarrow{\alpha} P'}{B \xrightarrow{\alpha} P'}$$

The rules for prefix, nondeterministic choice, and recursion are standard, while the trusted choice operator establishes that $a.P \mp b.Q$ executes either a followed by P or (a decorated version of) b followed by Q. The intuition is that this operator is used to communicate one of two possible actions to another process and that the choice will be guided by the trust towards such a process: if trust is beyond a certain threshold, then the offered action is a, otherwise it is b. The isolated semantics of this operator offers both actions, as the identity of the interacting process is still unknown, but it uses a decoration to distinguish which action is to be considered in the absence of sufficient trust.

Example 1. With respect to our running example, let us model the behavior of a generic (potentially dishonest) requester possibly interacting with n requestees and the behavior of a generic requestee possibly interacting with m requesters.

The process term describing the requester behavioral pattern is:

$$\begin{aligned}
Requester &\stackrel{\text{def}}{=} send_req_1.Wait_1 + \ldots + send_req_n.Wait_n \\
Wait_i &\stackrel{\text{def}}{=} rec_accept_i.Service_i + rec_refuse_i.Requester && 1 \leq i \leq n \\
Service_i &\stackrel{\text{def}}{=} pay_i.Requester + not_pay_i.Requester && 1 \leq i \leq n
\end{aligned}$$

while the requestee counterpart is as follows:

$$\begin{aligned}
Requestee &\stackrel{\text{def}}{=} rec_req_1.Decision_1 + \ldots + rec_req_m.Decision_m \\
Decision_i &\stackrel{\text{def}}{=} send_accept_i.\tau.Payment_i \mp send_refuse_i.Requestee && 1 \leq i \leq m \\
Payment_i &\stackrel{\text{def}}{=} rec_pay_i.Requestee + not_rec_pay_i.Requestee && 1 \leq i \leq m
\end{aligned}$$

□

A process instance, called entity, is an element exhibiting the behavior associated to a process term. The kernel $[\![I]\!]$ of the semantics of an entity I belonging to the behavioral pattern defined by process term P is given by the behavior of P, in which every action α is renamed to $I.\alpha$ [3]. With abuse of terminology, we say that I is of type P, and we write $I.B$ to specify that the behavior of I in the current state is given by the process term associated to B.

Example 2. In our running example, we consider a system with a single requester, modeled by the entity Req_A of type *Requester*, and three requestees, which are represented by the entities Req_1, Req_2, and Req_3, each one of type *Requestee*. □

2.2 Modeling Trust and Reputation

The execution of the interactions in which every entity in a system is involved depends strictly on the trust infrastructure regulating the communications within the community. Hence, before introducing the semantics for interaction, we first define formally such an infrastructure with respect to a set $\mathcal{S}$ of individual entities, by assuming that each entity name is unique to avoid ambiguity.

Let $IName = \{I.a \mid I \in \mathcal{S} \wedge a \in Name\}$ be the set of interacting action names. Moreover, $\mathbb{T}$ represents the domain of trust values. Even if in principle we may adopt any trust domain by adequately defining the semantics of the structures manipulating trust values, for the sake of presentation in the following we assume it to be a totally ordered set, the maximum (resp., minimum) value of which is denoted by $\top$ (resp., $\bot$).

A trust system is a tuple consisting of a set $\mathcal{S}$ of interacting processes and of the following structures:

- *Trust table* $tt : \mathcal{S} \times \mathcal{S} \to \mathbb{T}$, such that $tt[I; J]$ denotes the direct trust of entity I towards entity J as a result of previous interactions between them. Each row $tt[I; \_]$ is initialized with the dispositional trust of I, which is the initial willingness of I to trust unknown users.
- *Recommendation table* $rt : \mathcal{S} \times \mathcal{S} \to \mathbb{T}$, such that $rt[I; J]$ contains either the trust value recommended by I about J to other entities, or the special symbol δ to specify that I does not provide recommendations about J.
- *Trust threshold function* $tth : \mathcal{S} \to \mathbb{T}$, such that $tth(I)$ represents the minimum amount of trust (towards other entities) required by I to execute a trusted interaction.
- *Trust variation function* $tv : IName \to \mathbb{T}$, such that $tv(I.a)$ is the trust feedback that I associates to the execution of interactions through action a.
- *Trust function* $tf : \mathcal{S} \times \mathcal{S} \to \mathbb{T}$, such that $tf(I, J)$ computes the trust of I towards J according to a trust formula taking into account direct trust (deriving from the trust table) and reputation (deriving from the recommendation table).

We implicitly assume that the trust structures are parameterized with respect to a given type of service, and that several, mutual independent structures are needed if we intend to model a system offering different types of services, each one requiring separate trust information. In this case, every action must be parameterized as well with respect to the service type, in order to guide each interaction among entities according to the related trust information.

As far as the trust function *tf* is concerned, here we do not define it as its specification strictly depends on the chosen trust model and, as we will see,

it does not affect the definition of the semantics for interacting processes. Function tf may be based on several different methods [16,24,23], an example of which will be given with respect to our case study. We can argue similarly in the case of the specific relation existing between trust table and recommendation table and, in particular, the way in which an entity provides feedback to other entities on the basis of personal experience. However, some aspects of such a relation (that change depending whether the reputation system is centralized or distributed) deserve discussion here.

In a centralized scenario, we can envision a trusted third party collecting trust information from all the entities. Such a collection contributes to form the reputation of each entity as perceived by the community. Hence, it is reasonable to assume that every entity requiring a recommendation has access to such information in the same way and obtains the same feedback. From a semantics viewpoint, this scenario is captured by formalizing the relation between trust table and recommendation table.

For instance, in a very simple scenario, the recommendation provided by I about J is exactly the trust of I towards J, under the assumption that I had some direct experience with J (otherwise the suggested value would be simply the dispositional trust of I). Let $ct : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ be the *contact table*, such that $ct[I; J] = 1$ if and only if entities I and J interacted with each other (initially, $ct[I; J] = 0$ for each pair of entities in the set $\mathcal{S}$). Then, the relation between trust table and recommendation table is described by the following equation:

$$rt[I; J] = \begin{cases} tt[I; J] & \textit{if } \ ct[I; J] = 1 \\ \delta & \textit{otherwise} \end{cases} \tag{1}$$

thus assuming that all the entities recommend exactly the trust values resulting from their own experience, if any. Notice that this would not be the case in the presence, e.g., of entities providing inaccurate feedback or attackers cheating deliberately other entities. In order to model such a case, it is sufficient to alter some rows (or specific entries) of the recommendation table with respect to the trust table.

In a distributed scenario, the absence of a centralized trusted third party has two important effects. Firstly, different entities may have access to different information if they are in contact with different neighbors. Secondly, an entity may provide, for the same recommendation, different values to different entities. These situations are managed by adding a dimension to the recommendation table, such that each recommendation is specified by the identities of the recommender entity, the recommended entity, and the entity receiving the recommendation. Formally, if $ct[I; J] = ct[I; K] = 1$ then $rt[I; J; K]$ denotes the trust value recommended by I about J to K. In this case all the formulas depending on the recommendation table are changed accordingly.

Example 3. In our running example, let trust be a discrete metric such that $\mathbb{T} = [0..10]$. Initially, the trust table is as follows:

	Req_A	Req_1	Req_2	Req_3
Req_A		8	8	8
Req_1	2		2	2
Req_2	3	3		3
Req_3	5	5	5	

The recommendation table is calculated by means of Equation 1 (notice that we are not considering self-promoting behaviors, which, however, could be modeled). Even if we assume a distributed scenario, requester and requestees are connected without any restriction and can communicate with each other. The trust threshold function establishes that the requester issues requests without any reputation constraint, $tth(Req_A) = 0$, and that for each requestee the service trust threshold is equal to requestee's dispositional trust: $tth(Req_1) = 2$, $tth(Req_2) = 3$, and $tth(Req_3) = 5$.

The trust variation function establishes that the requester increases (resp., decreases) by one unit the trust towards any requestee accepting (resp., refusing) a request, namely $tv(Req_A.rec\_accept\_i) = 1$ and $tv(Req_A.rec\_refuse\_i) = -1$ for $1 \leq i \leq 3$. Each requestee increases trust towards the requester in case of paid service, $tv(Req_i.rec\_pay\_1) = 1$ for $1 \leq i \leq 3$. The first two requestees decrease trust by the same amount in case of unpaid service, $tv(Req_i.not\_rec\_pay\_1) = -1$ for $1 \leq i \leq 2$, while the third one is more cautious and applies the maximum penalty, $tv(Req_3.not\_rec\_pay\_1) = -10$. All the other actions do not imply any trust variation.

Finally, the trust formula is abstracted as follows. Let:

$$Rec_{I,J} = \mathcal{S} \backslash \{\{I, J\} \cup \{K \mid rt[K; J] = \delta\}\}$$

be the set of entities from which I receives recommendations about J. Then:

$$tf(I, J) = \begin{cases} tt[I; J] & \text{if } Rec_{I,J} = \emptyset \\ \rho_I \cdot tt[I; J] \; + \; (1 - \rho_I) \cdot \frac{\sum_{K \in Rec_{I,J}} rt[K;J]}{|Rec_{I,J}|} & \text{otherwise} \end{cases}$$

where ρ_I represents the risk factor for I, i.e., how much of its trust towards other entities depends on previous direct experience. The factor that is multiplied by $1 - \rho_I$ represents the average trust towards J resulting from recommendations provided by third entities. For the three requestees, in the following we assume that the risk factor is equal to 0.5, 0.8, and 0.8, respectively.

In general, notice that the most risky profile is adopted by the first requestee, while the third requestee is characterized by the most cautious behavior [1]. □

2.3 Modeling Interacting Processes

The semantics of interacting entities arises from the parallel composition of a set $\mathcal{S}$ of individual entities following the communication rules established by

a synchronization set SS, which is a set of names of the form $I.a\_to\_J.b$. In particular, $I.a\_to\_J.b$ denotes a synchronization between I and J in which I offers action a and J responds with action b. In other words, $I.a$ is the output part of the communication, $J.b$ represents the input counterpart, and $I.a\_to\_J.b$ is the name of the synchronized action.

Example 4. In our running example, the synchronization set for the group of entities $\{Req_A, Req_1, Req_2, Req_3\}$ includes the actions:

$$Req_A.send\_req\_i\_to\_Req_i.rec\_req\_1$$
$$Req_i.send\_accept\_1\_to\_Req_A.rec\_accept\_i$$
$$Req_i.send\_refuse\_1\_to\_Req_A.rec\_refuse\_i$$
$$Req_A.pay\_i\_to\_Req_i.rec\_pay\_1$$
$$Req_A.not\_pay\_i\_to\_Req_i.not\_rec\_pay\_1$$

where $1 \leq i \leq 3$.

The system topology resulting from such a synchronization set reveals that the requester may interact with every requestee, while communications among requestees do not occur, except for the potential exchange of recommendations. Notice that such an exchange is modeled implicitly through the definition of the recommendation policy. According to the trust infrastructure described in the previous section, the system topology has the following effect on the calculation of reputation. Each requestee receives recommendations from any other requestee if such a requestee has interacted with the requester, while the requester does not receive recommendations, meaning that $tf(Req_A, Req_i) = tt[Req_A; Req_i]$ for $1 \leq i \leq 3$ independently of the risk factor chosen by Req_A. □

The interacting semantics of $\mathcal{S}$ is given by the parallel composition of the semantics $[\![I]\!]$ of all the entities $I \in \mathcal{S}$. In the semantic rules for parallel composition, let $P, P', Q, Q', \ldots$ denote process terms representing the local behavior $[\![I]\!]$ of any entity $I \in \mathcal{S}$. Moreover, let $\mathcal{P}$ be a vector of local behaviors with as many elements as the number of entities in $\mathcal{S}$, each one expressing the current local behavior of the related entity. Then, $\mathcal{P}[P'/P]$ denotes the substitution of P with P' in $\mathcal{P}$. The semantic rule for internal actions is as follows:

$$\frac{P \in \mathcal{P} \quad P \xrightarrow{I.\tau} P'}{\mathcal{P} \xrightarrow{I.\tau} \mathcal{P}[P'/P]}$$

The rule establishes that every entity executes its internal actions independently from each other. Then, based on the trust information, interactions among entities occur (or do not occur) and their execution provides feedback. In order to emphasize the separation of concerns between trust modeling and behavior modeling, the rule premises concerned with the trust structures are specified syntactically as external side conditions. Hence, the semantic rules expressing interactions are:

$$\frac{P, Q \in \mathcal{P} \quad I.a\_to\_J.b \in SS \quad P \xrightarrow{I.a} P' \quad Q \xrightarrow{J.b} Q'}{\mathcal{P} \xrightarrow{I.a\_to\_J.b} \mathcal{P}[P'/P, Q'/Q]} \quad \begin{array}{l} tt[I;J]{=}update(tt[I;J], tv(I.a)) \\ tt[J;I]{=}update(tt[J;I], tv(J.b)) \\ tf(I,J) \geq tth(I) \end{array}$$

and:

$$\frac{P,Q \in \mathcal{P} \quad I.a\_to\_J.b \in SS \quad P \xrightarrow{I.a^-} P' \quad Q \xrightarrow{J.b} Q'}{\mathcal{P} \xrightarrow{I.a\_to\_J.b} \mathcal{P}[P'/P, Q'/Q]} \begin{array}{l} tt[I;J]{=}update(tt[I;J],tv(I.a)) \\ tt[J;I]{=}update(tt[J;I],tv(J.b)) \\ tf(I,J){<}tth(I) \end{array}$$

where function *update* formalizes the effect of the interaction upon the trust between the involved parties. For instance, if we assume $\mathbb{T}$ to be a finite set of integers and $tv(I.a)$ to denote the trust gain/loss, then we have:

$$update(v,k) = \begin{cases} max(\bot, v+k) \ \textit{if} \ k < 0 \\ min(\top, v+k) \ \textit{if} \ k > 0 \end{cases}$$

Intuitively, the first rule states that if entities I and J enable, respectively, the interacting actions $I.a$ and $J.b$, the communication guided by I is allowed, i.e., $I.a\_to\_J.b \in SS$, and J is trusted enough by I, i.e., $tf(I,J) \geq tth(I)$, then the interaction is executed and both I and J update their mutual trust accordingly. The second rule behaves essentially the same, except that it models the case in which the communication from I to J occurs if I does not trust J enough, see action $I.a^-$ and the premise $tf(I,J) < tth(I)$, in compliance with the use of the trusted choice operator. Notice that, in order to consider the case in which the contact table is necessary for the trust calculation, the update $ct[I;J] = 1$ must be added to the premises to keep track of the interaction.

The separation of concerns – between functional behavior modeling and trust representation – is realized at the syntax level and favors independent reasoning and control. All the information and policies concerning trust are not involved syntactically in the specification of the process terms modeling the functional behavior of systems. Instead, they are described in a separate infrastructure, thus facilitating modeling and then sensitivity analysis. Functional behavior and trust management are combined at the semantics level in a fully automatic way governed by the operational semantic rules.

As far as the resulting semantic model is concerned, if trust has a finite value domain, then a concrete treatment of semantics is applied, meaning that the actual instantiations of the trust parameters become part of the formal semantics by contributing to label the states of the labeled transition system expressing the system behavior. Such a condition is achieved easily whenever trust is a finite, discrete metric, as usual in several trust-based systems [14]. In order to define the formal semantics of a system of interacting entities, we need to extend the notion of lts in order to take into account in each state the trust information affecting the application of the semantic rules. In particular, it is worth noticing that the variables of the trust infrastructure needed to determine the enabled transitions are represented by the entries of the trust and recommendation tables. In the following, we limit ourselves to consider the case of the trust table, as the extension including both tables is straightforward.

Definition 2. *Given a domain V of trust variables and a domain $\mathbb{T}$ of trust values, a trust labeled transition system (tlts) is a tuple (Q, q_0, L, R, T, P) where*

(Q, q_0, L, R) is a lts, T is a finite set of trust predicates of the form $v = k$, with $v \in V$ and $k \in \mathbb{T}$, and $P : Q \to 2^T$ is a labeling function that associates a subset of T to each state of the tlts.

Hence, the semantics of a trust system made of a set $\{I_1, \ldots, I_n\}$ of entities obeying the synchronization set SS and the trust table tt is the smallest tlts such that the following conditions hold. Firstly, each state in Q represents a n-length vector of process terms modeling the local behavior of each entity I_j, $1 \leq j \leq n$; the initial state q_0 is associated to the vector modeling the initial local state of each entity; the trust predicates in T denote all the possible assignments in the trust table tt and the labeling function P associates a configuration of such a table to each state, by assuming that the initial state of the tlts is labeled by the initialization of tt according to the given trust infrastructure. Secondly, the transitions in R are obtained through the application of the semantic rules for parallel composition and, therefore, the set of labels L is given by the set $IAct$, ranged over by i, containing internal actions of the form $I_j.\tau$ and interactions in SS. Therefore, a transition $(p, i, q) \in R$ determines, depending on the global state p and the action i, both the vector of local states and the set of trust predicates labeling q.

Example 5. The initial state of the tlts related to our running example is associated to the vector of process terms:

$$[Req_A.Requester, Req_1.Requestee, Req_2.Requestee, Req_3.Requestee]$$

and is labeled with the trust predicates given by the trust table depicted in Example 3. The transitions departing from this state are three, labeled with $Req_A.send\_req\_i\_to\_Req_i.rec\_req\_1$, $1 \leq i \leq 3$, respectively. □

3 Model Checking Trust Properties

The formal semantics of a trust system of interacting processes is based on tlts, which is an instance of doubly labeled transition systems [21], and of Kripke transition systems [8]. Hence, it is possible to employ temporal logics for such systems in order to define a trust logic for specifying both conditions based on the actions labeling the transitions and requirements based on the trust information labeling the states. We call such a language trust temporal logic (TTL). In particular, TTL embodies features of the classical branching-time state-based Computation Tree Logic [11] and of its action-based variant ACTL [12].

TTL includes the definition of state formulas, which are applied to states of a tlts, and path formulas, which are applied to sequences of transitions of a tlts. The syntax of TTL is defined as follows:

$$\begin{array}{rcl} \Phi & ::= & true \mid i \mid v \geq k \mid \Phi \wedge \Phi \mid \neg\Phi \mid A\pi \mid E\pi \\ \pi & ::= & \Phi \, {}_{\mathcal{A}_1}U \, \Phi \mid \Phi \, {}_{\mathcal{A}_1}U_{\mathcal{A}_2} \, \Phi \end{array}$$

where $v = tt[I; J]$, with I and J entity names, $k \in \mathbb{T}$, $i \in IAct$, and $\mathcal{A}_1, \mathcal{A}_2 \subseteq IAct$. Inspired by other logics merging action/state-based predicates [4], atomic

propositions are either actions or trust predicates of the form $v \geq k$, where variable v denotes any entry of the trust table and k belongs to the trust domain. State formulas are ranged over by Φ. Intuitively, a state satisfies the atomic proposition i if it enables a transition labeled with i, while it satisfies the atomic proposition $v \geq k$ if it is labeled with a trust predicate that assigns to v a value greater than (or equal to) k. Composite state formulas are obtained through the classical connectives. The operators A and E denote the universal and existential path quantifiers. A state satisfies $A\pi$ (resp., $E\pi$) if *every* path (resp., *at least one* path) departing from such a state satisfies the path formula π. Path formulas are ranged over by π, while U is the indexed until operator. Intuitively, a path satisfies the until formula $\Phi\, {}_{\mathcal{A}_1}U\, \Phi'$ if the path visits a state satisfying Φ', and visits states satisfying Φ while performing only actions in $\mathcal{A}_1$ until that point. Similarly, the until formula $\Phi\, {}_{\mathcal{A}_1}U_{\mathcal{A}_2}\, \Phi'$ is satisfied by a path if the path visits a state satisfying Φ' after performing an action in $\mathcal{A}_2$, and visits states satisfying Φ while performing only actions in $\mathcal{A}_1$ until that point. We observe that a path satisfying $\Phi\, {}_{\mathcal{A}_1}U_{\mathcal{A}_2}\, \Phi'$ must include a transition to a state satisfying Φ', while this is not required for $\Phi\, {}_{\mathcal{A}_1}U\, \Phi'$ if the initial state of the path satisfies Φ'.

Similarly as argued in the previous section, if the states of the tlts include reputation-based information deriving from the recommendation table, we can enrich TTL with reputation-based state predicates.

Now, let us define formally some notion about paths with respect to a tlts (Q, q_0, L, R, T, P). A path σ is a (possibly infinite) sequence of transitions of the form: $p_0 \xrightarrow{i_0} p_1 \ldots p_{j-1} \xrightarrow{i_{j-1}} p_j \ldots$ where $p_{j-1} \xrightarrow{i_{j-1}} p_j \in R$ for each $j > 0$. Every p_j in the path is denoted by $\sigma(j)$. Moreover, let $p_j \xrightarrow{\mathcal{A}} p_{j+1}$ if and only if $i_j \in \mathcal{A} \subseteq L$. We denote with $Path(q)$ the set of paths starting in state $q \in Q$. Then, the formal semantics of TTL is as shown in Table 1.

Table 1. Semantics of TTL

$q \models true$	*holds always*
$q \models v \geq k$	*iff* $(v = k') \in P(q) \wedge k' \geq k$
$q \models i$	*iff* $\exists p : q \xrightarrow{i} p \in R$
$q \models \Phi \wedge \Phi'$	*iff* $q \models \Phi$ *and* $q \models \Phi'$
$q \models \neg\Phi$	*iff* $q \not\models \Phi$
$q \models A\pi$	*iff* $\forall \sigma \in Path(q) : \sigma \models \pi$
$q \models E\pi$	*iff* $\exists \sigma \in Path(q) : \sigma \models \pi$
$\sigma \models \Phi\, {}_{\mathcal{A}_1}U\, \Phi'$	*iff* $\exists k \geq 0$: $\sigma(k) \models \Phi' \wedge (for\ all\ 0 \leq i < k : \sigma(i) \models \Phi \wedge \sigma(i) \xrightarrow{\mathcal{A}_1} \sigma(i+1))$
$\sigma \models \Phi\, {}_{\mathcal{A}_1}U_{\mathcal{A}_2}\, \Phi'$	*iff* $\exists k > 0$: $\sigma(k) \models \Phi' \wedge (for\ all\ 0 \leq i < k-1 : \sigma(i) \models \Phi \wedge \sigma(i) \xrightarrow{\mathcal{A}_1} \sigma(i+1)) \wedge \sigma(k-1) \models \Phi \wedge \sigma(k-1) \xrightarrow{\mathcal{A}_2} \sigma(k)$

TTL can be mapped to the logic UCTL [21], for which an efficient on-the-fly model checking algorithm is implemented. The unique non-trivial difference between the two logics is that TTL allows for action-based atomic propositions, while UCTL does not. The atomic proposition i of TTL can be represented through the UCTL until operator as follows. Denoted with *false* the formula $\neg true$, then i is expressed by the formula $E(false\ _{\emptyset}U_{\{i\}}\ true)$, which establishes that from the current state a transition labeled with i is enabled that leads to a state satisfying the atomic formula *true*, i.e., given q the current state, it holds that $\exists p : q \xrightarrow{i} p \in R$.

Finally, we provide two flavors of classical operators like *next* (X), *eventually* (F), and *always* (G), depending on the kind of until operator used. To this end, we introduce the following notations:

$$\begin{array}{ll} X\Phi = false\ _{\emptyset}U_{IAct}\ \Phi & X_{\mathcal{A}_1}\Phi = false\ _{\emptyset}U_{\mathcal{A}_1}\ \Phi \\ EF\Phi = E(true\ _{IAct}U\ \Phi) & EF_{\mathcal{A}_1}\Phi = E(true\ _{IAct}U_{\mathcal{A}_1}\ \Phi) \\ AF\Phi = A(true\ _{IAct}U\ \Phi) & AF_{\mathcal{A}_1}\Phi = A(true\ _{IAct}U_{\mathcal{A}_1}\ \Phi) \\ EG\Phi = \neg AF\neg\Phi & EG_{\mathcal{A}_1} = \neg AF_{IAct-\mathcal{A}_1}\, true \\ AG\Phi = \neg EF\neg\Phi & AG_{\mathcal{A}_1} = \neg EF_{IAct-\mathcal{A}_1}\, true \end{array}$$

For instance, $EG\Phi$ holds in p if there exists a path in $Path(p)$ every state of which (including p) satisfies Φ, while $EG_{\mathcal{A}_1}$ holds in p if there exists a path in $Path(p)$ every transition of which is labeled with an action in $\mathcal{A}_1$.

Example 6. With respect to our running example, we focus on the comparison between the two limiting profiles, i.e., risky and cautious, which characterize the behavior of the requestees. After adequate translation of the model, the following properties have been recast and checked both in PRISM [17] and in NuSMV [10].

The first parameter under analysis is the risk factor and the related impact upon the capability of being influenced by recommendations. To this aim, we formulate the following condition to check. Can the risky requestee accept a request without sufficient direct trust towards the requester? The related property is stated formally as follows:

$$EF(tt[Req_1; Req_A] < 2 \wedge Req_1.send\_accept\_1\_to\_Req_A.rec\_accept\_1)$$

where the state predicate $tt[Req_1; Req_A] < 2$ describes the trust condition and the action predicate $Req_1.send\_accept\_1\_to\_Req_A.rec\_accept\_1$ represents the behavior to observe, while the formula schema expresses the eventuality of reaching a state satisfying both predicates. The property is satisfied, because by virtue of the assumption $\rho = 0.5$, positive recommendations provided to the risky requestee can balance (and overcome the effect of) negative direct experiences. The same property can be recast in the case of the cautious requestee:

$$EF(tt[Req_3; Req_A] < 5 \wedge Req_3.send\_accept\_1\_to\_Req_A.rec\_accept\_3)$$

which is not satisfied, thus confirming the prudent behavior of this requestee.

An interesting analysis concerns the consequences of a malicious behavior of the requester. The following property:

$$AG(\ Req_A.not\_pay\_3\_to\_Req_3.not\_rec\_pay\_1 \rightarrow \\ AG(\neg Req_3.send\_accept\_1\_to\_Req_A.rec\_accept\_3))$$

is satisfied, thus establishing that after experiencing a cheating behavior of the requester (action $Req_A.not\_pay\_3\_to\_Req_3.not\_rec\_pay\_1$) the cautious requestee does not trust the requester anymore (in every future state it holds that the action $Req_3.send\_accept\_1\_to\_Req_A.rec\_accept\_3$ cannot be enabled). By replacing the cautious requestee with the risky requestee, the corresponding property is violated. Actually, not very surprisingly, even the following property is satisfied:

$$EF(EG_{\mathcal{A}_1})$$

where $\mathcal{A}_1$ is the set of actions:

$$\{\ Req_A.send\_req\_1\_to\_Req_1.rec\_req\_1, \\ Req_1.send\_accept\_1\_to\_Req_A.rec\_accept\_1, \\ Req_1.\tau, \\ Req_A.not\_pay\_1\_to\_Req_1.not\_rec\_pay\_1\ \}.$$

This means that a certain point can be reached starting from which the requester can obtain services from the risky requestee infinitely often without paying for any of them. This situation is an immediate consequence of the first property, which demonstrates that the direct mistrust of the risky requestee towards the requester is not sufficient to exclude the cheating behavior.

On the other hand, let us now consider a completely honest requester. This variant can be obtained either by eliminating from requester's process terms any action not_pay_i or, even better, by removing the related actions from the synchronization set SS. In this scenario, we verify whether eventually a point is reached starting from which every issued request is accepted:

$$EF(AG_{IAct-\mathcal{A}_1})$$

where $\mathcal{A}_1$ is the set of actions:

$$\{\ Req_1.send\_refuse\_1\_to\_Req_A.rec\_refuse\_1, \\ Req_2.send\_refuse\_1\_to\_Req_A.rec\_refuse\_2, \\ Req_3.send\_refuse\_1\_to\_Req_A.rec\_refuse\_3\ \}.$$

Such a property holds as expected.

Separating functional behavior modeling and trust management specification allows for a clear verification of the impact of trust policies upon specific properties by simply adjusting the trust parameters of certain entities. For instance, let us replace the cautious requestee with a paranoid requestee characterized by strict trust requirements, and then let us consider the capability of such an entity of accepting services. To this aim, we adjust the trust infrastructure only,

by tuning ρ, tth, and dispositional trust for entity Req_3. As an example, with $\rho = 0.8$, $tth(Req_3) = 5$ (as for the cautious requestee), and dispositional trust less than 4, we obtain that the following property is not satisfied:

$$EF(Req_3.send\_accept\_1\_to\_Req_A.rec\_accept\_3)$$

meaning that the paranoid requestee does not serve any request. The property turns out to hold if the dispositional trust is set to 4, in which case we also observe that, given $\mathcal{A}_1 = \{Req_3.send\_accept\_1\_to\_Req_A.rec\_accept\_3\}$, the property:

$$E((tt[Req_1; Req_A] < 10 \wedge tt[Req_2; Req_A] < 10)\ {}_{IAct}U_{\mathcal{A}_1}\ true)$$

does not hold. More precisely, at least one of the other two requestees must recommend top trust towards the requester in order to allow the paranoid requestee to accept a request.

Finally, let us consider a coalition attack by two requestees against the third one. The condition of interest is formulated as follows. Can malicious requestees provide false feedback to the risky requestee thus avoiding her/him from accepting any request? To this aim, it is sufficient to extend the recommendation table by setting $rt[Req_2; Req_A; Req_1] = rt[Req_3; Req_A; Req_1] = 0$ (while all the other entries are as usual), and then check the TTL formula:

$$\neg EF(Req_1.send\_accept\_1\_to\_Req_A.rec\_accept\_1).$$

This property is satisfied, thus revealing the effectiveness of the attack. By tuning the dispositional trust of the risky requestee, we observe that the attack can be avoided if and only if such a parameter is set to at least 4. On the other hand, if the false feedback is provided by Req_3 only, Req_1 can accept requests (even without altering her/his dispositional trust), but only after a successful interaction between Req_2 and Req_A. In this case, we have also verified that extremely positive recommendations by Req_2 ($rt[Req_2; Req_A; Req_1] = top$) protect Req_1 from coalition attacks of (up to) 4 malicious requestees.

4 Related Work and Future Directions

In the literature, formal methods have been used successfully to model trust and trust relationships [20,13,15]. However, usually these techniques represent trust without an integration with formal approaches to the modeling and verification of concurrent/distributed systems. Theoretical analysis of cooperation strategies is proposed in formal frameworks like, e.g., game theory [18], and the theory of semirings [22]. The analysis of trust chains is investigated also in a process algebraic setting, either with a specific focus on access control policies [19], or by employing equivalence checking based analysis [7]. In this paper we have proposed a process algebraic framework in which trust modeling and system specification are combined and model checking techniques are applied to verify the effects of trust models and related parameters upon cooperation in concurrent and distributed systems.

An extension under development concerns the semantics, which is currently based on nondeterministic labeled transition systems. Without altering the syntax of the language, the idea is to employ quantitative information deriving from the trust infrastructure in order to implement nonfunctional trust-based choice policies at the level of the semantics for interacting processes. For instance, trust can be interpreted as a weight guiding the choice among concurrent trust-based interactions, which thus becomes either probabilistic or prioritized. In the former case, the semantics would be based on probabilistic tlts. Further extensions are concerned with the use of reward structures expressing metrics that can be related to trust. This is the case, e.g., of the service cost, as well as any other parameter related to the quality of experience that may be influenced by (or may affect) trust. As an example, every time an interaction modeling a payment from I to J occurs, a certain reward depending on the trust relation between I and J is cumulated to express the amount paid by I. Then, similarly as done in the setting of quantitative model checking [5,4,9], we can employ a version of TTL extended with probabilities and rewards to estimate the tradeoff existing between trust and other metrics, which is necessary to evaluate mixed cooperation incentive strategies [1,2].

Acknowledgment. This work has been partially supported by the MIUR project CINA (Compositionality, Interaction, Negotiation, Autonomicity for the future ICT society).

References

1. Aldini, A.: Formal Approach to Design and Automatic Verification of Cooperation-Based Networks. Journal on Advances in Internet Technology 6, 42–56 (2013)
2. Aldini, A., Bogliolo, A.: Modeling and Verification of Cooperation Incentive Mechanisms in User-Centric Wireless Communications. In: Rawat, D., Bista, B., Yan, G. (eds.) Security, Privacy, Trust, and Resource Management in Mobile and Wireless Communications, pp. 432–461. IGI Global (2014)
3. Aldini, A., Bernardo, M., Corradini, F.: A Process Algebraic Approach to Software Architecture Design. Springer (2010)
4. Aldini, A., Bernardo, M., Sproston, J.: Performability Measure Specification: Combining CSRL and MSL. In: Salaün, G., Schätz, B. (eds.) FMICS 2011. LNCS, vol. 6959, pp. 165–179. Springer, Heidelberg (2011)
5. Baier, C., Cloth, L., Haverkort, B., Hermanns, H., Katoen, J.-P.: Performability Assessment by Model Checking of Markov Reward Models. Formal Methods in System Design 36, 1–36 (2010)
6. Bogliolo, A., Polidori, P., Aldini, A., Moreira, W., Mendes, P., Yildiz, M., Ballester, C., Seigneur, J.-M.: Virtual Currency and Reputation-Based Cooperation Incentives in User-Centric Networks. In: Proc. 8th Wireless Communications and Mobile Computing Conf (IWCMC 2012), pp. 895–900. IEEE (2012)
7. Carbone, M., Nielsen, M., Sassone, V.: A Calculus for Trust Management. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 161–173. Springer, Heidelberg (2004)

8. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 128–147. Springer, Heidelberg (2004)
9. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic Verification of Competitive Stochastic Systems. Formal Methods in System Design 43(1), 61–92 (2013)
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
11. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems 8(2), 244–263 (1986)
12. De Nicola, R., Vaandrager, F.W.: Actions versus State based Logics for Transition Systems. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
13. Huang, J., Nicol, D.: A Calculus of Trust and Its Application to PKI and Identity Management. In: Proc. 8th Symposium on Identity and Trust on the Internet (IDtrust 2009), pp. 23–37. ACM (2009)
14. Jøsang, A.: Trust and Reputation Systems. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 4677, pp. 209–245. Springer, Heidelberg (2007)
15. Jøsang, A.: Subjective logic book, draft (2013), `http://folk.uio.no/josang/papers/subjective_logic.pdf`
16. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The Eigentrust Algorithm for Reputation Management in P2P Networks. In: Proc. 12th Conf. on World Wide Web (WWW 2003), pp. 640–651. ACM (2003)
17. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
18. Li, Z., Shen, H.: Game-Theoretic Analysis of Cooperation Incentives Strategies in Mobile Ad Hoc Networks. IEEE Transactions on Mobile Computing (2012)
19. Martinelli, F.: Towards an Integrated Formal Analysis for Security and Trust. In: Steffen, M., Zavattaro, G. (eds.) FMOODS 2005. LNCS, vol. 3535, pp. 115–130. Springer, Heidelberg (2005)
20. Nielsen, M., Krukow, K.: Towards a formal notion of trust. In: Proc. 5th ACM SIGPLAN Conf. on Principles and Practice of Declaritive Programming (PPDP 2003), pp. 4–7 (2003)
21. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An Action/State-Based Model-Checking Approach for the Analysis of Communication Protocols for Service-Oriented Applications. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 133–148. Springer, Heidelberg (2008)
22. Theodorakopoulos, G., Baras, J.S.: On Trust Models and Trust Evaluation Metrics for Ad Hoc Networks. IEEE Journal on Selected Areas in Communications 24, 318–328 (2006)
23. Zhang, Y., Lin, L., Huai, J.: Balancing Trust and Incentive in Peer-to-Peer Collaborative Systems. Journal of Network Security 5, 73–81 (2007)
24. Zhou, R., Hwang, K.: PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. IEEE Transactions on Parallel and Distributed Systems 18(4), 460–473 (2007)