Temporal Logic-Based Specification and Verification of Trust Models

Peter Herrmann

Norwegian University of Science and Technology (NTNU)
Telematics Department, 7491 Trondheim, Norway
Email: herrmann@item.ntnu.no

Abstract. Mutual trust is essential in performing economical transactions. In modern internet-based businesses, however, traditional trust gaining mechanisms cannot be used and new ways to build trust between e-business partners have to be found. In consequence, a lot of models describing trust and the mechanisms to build it were developed. Unfortunately, most of these models neither provide the right formalism to model relevant aspects of the trust gaining process (e.g., context and time of a trust-related interaction), nor they allow refinement proofs verifying that a trust management tool implements a certain trust model. Therefore, we propose the temporal logic-based specification and verification technique cTLA which provides a formalism enabling to model context- and time-related aspects of a trust building process. Moreover, cTLA facilitates formal refinement proofs. In this paper, we discuss the application of cTLA to describe trust purposes by means of simple example systems which are used to decide about the application of certain policies based on the reputation of a party. In particular, we introduce a basic and a refined reputation system and sketch the proof that the refined system is a correct realization of the simple one.

1 Introduction

Since the very beginnings of mankind, trust is an essential ingredient for human cooperation. In a society based on the division of labor, people often are willing to rely on others, even though they might face negative consequences. According to McKnight and Chervany [1], however, this is exactly the definition of trust. Ways to build trust include personal experience with the party, one relies on, recommendation by third parties as well as the reduction of the negative consequences in the case of unfounded trust (e.g., by using an insurance covering financial losses in case of malicious behavior of a trusted party).

These traditional trust gaining mechanisms, however, can hardly be applied in modern internet-based businesses. Here, one performs transactions on an ad-hoc basis with often changing anonymous partners living in other countries with different legal systems. Thus, it is difficult to gain any personal experience about transaction partners and trustworthy third persons which can give meaningful recommendation are also hardly available. Furthermore, suing for ones personal

rights in another legal system is stressful. In some cases, security mechanisms (e.g., authentication, non-repudiation techniques) can be applied to facilitate cooperations. But especially in an electronic environment providing ad hoc cooperations between anonymous parties, security mechanisms are often of limited benefit. For example, it is tedious to use a traditional password-based authentication system with proof of identity and exchange of passwords if the user wants to perform a business transaction only once. Instead, a party needs mechanisms to foster the building of well-founded trust to its partner based, for instance, on recommendations by trustworthy third parties or by the partner's reputation. The creation of this kind of mechanisms is a goal followed up by the Trust Management research discipline.

To support the trust building process by a computer, one has to find suitable representations of trust, so-called trust values (cf. [2, 3]). Moreover, one needs mechanisms to compute the trust values in a way that the natural way to build up trust is modeled fairly realistically. Thus, these mechanisms should be able to consider experiences and recommendations of computer users to compute suitable trust values. As pointed out by Falcone and Castelfranchi [4], new trust is influenced by already existing trust in rather complex ways which should also be reflected by a trust value computation mechanism. Furthermore, the mechanism has to consider the context in which building of trust takes place (cf. [1, 5]). Relevant aspects of a trust context are according to Jøsang et al. [6] the utility of possible outcomes, environmental factors (e.g, law enforcement, contracts, security mechanisms) as well as the risk attitude of the trusting party.

Another important aspect of a trust building mechanism is time. Falcone and Castelfranchi, for instance, call in [4] trust as a very dynamic phenomenon evolving in time and having a history. Likewise, a Cheskin Research study [7] examining trust concepts of e-commerce cites describes trust as "function of time and specific formal characteristics of sites". Also Mezzetti [8] states that trust values may be changed in the course of time. He considers recent events more relevant than older events for building trust, "since obsolete information is not considered to accurately describe more recent behaviors". In consequence, his trust building mechanism uses a decay function reducing a trust value in the course of time.

In between, several trust models describing formats for trust values as well as trust building mechanisms were proposed (cf. Sec. 2). Some of these models focus on the description of relevant aspects of the human trust building process per se which may be viewed from a rather formal mathematical or philosophical perspective (e.g. [9, 10]) as well as from a more sociological-cognitive view (e.g. [5, 8, 11]). Other models are devoted to computer-implementable solutions fostering the gaining of the trust [12–16]. While these approaches offer a variety of very useful concepts to specify and compute the generation of trust, most of them, however, miss the sufficient formalism to model all relevant aspects of trust building including context and time. Moreover, they do not allow to carry out deduction proofs that an implementation of a trust management system fulfills a trust model and particular trust properties. For this reason, we propose the

temporal logic cTLA (compositional Temporal Logic of Actions [17]) as a method to model and to verify trust mechanisms. It provides the formalism to define trust values and to model context- and time-dependent building of trust. Furthermore, it enables to carry out refinement proofs in a relatively suitable way. Yet, we do not intend to create a completely new trust model but adapt existing approaches like Jøsang's Subjective Logic [9] or Mezzetti's work [8] to cTLA.

cTLA is based on Lamport's Temporal Logic of Actions (TLA, [18]). It supports the modular description of processes which, in contrast to TLA, can be coupled to system models both in a resource-oriented and a constraint-oriented specification style (cf. [19]). In contrast to a resource-oriented style, where a process describes a physical system resource in its entirety, processes in the constraint-oriented style model certain functional properties of a system which may be realized by several cooperating resources. This specification style facilitates system descriptions by composing models of the various system constraints which reflects the logical connections and dependencies of a system very well. cTLA also enables the description of continuous flows [20]. Thus, it is possible to model the dynamic trust building process as a continuous process which is influenced by discrete events (e.g., the selection of an access policy based on positive or negative valuations of a party). In consequence, we can specify trust creation by means of continuous-discrete models similar to those describing computer-driven technical processes like a chemical plant.

The composition of cTLA processes to a system has the character of superposition (cf. [21]) guaranteeing that all relevant properties of a process or a subsystem are also properties of the systems embedding it. Therefore, one can simplify formal deduction proofs of properties by considering only a subsystem guaranteeing the property to be verified. In combination with the constraint-oriented specification style, one can define often very small subsystems which can easily been proven to realize a certain property. Thus, the structuring of a verification process into relatively simple proof steps is supported.

In the remainder, we discuss several trust models (Sec. 2) followed by an introduction to cTLA (Sec. 3). Thereafter we will point out the specification of trust management systems by means of two example systems. A more abstract system introduced in Sec. 4 describes a simple reputation system collecting good and bad experiences based on which one of two policies is selected. In Sec. 5 we introduce a refined system collecting the experiences from two separate users which have to be combined in a fair manner. Afterwards, in Sec. 6 we sketch the carrying out of refinement proofs in cTLA by outlining the verification that the more complex system correctly implements the more abstract one. The cTLA processes and proofs can also be looked at in the WWW (URL: `http://www.item.ntnu.no/~herrmann/specs/trust`).

2 Trust Model Survey

Trust is a rather complex human emotion and the models describing it tend to be complex as well. To reduce the complexity, however, most models consider

only certain aspects of trust building. A class of trust models specifies relevant issues of trust very realistically without being devoted directly to implementation purposes. These models tend to be relatively formal and describe trust gaining from a mathematical-philosophical or from a sociological-cognitive perspective.

An approach to describe trust by uncertain probabilities is Jøsang's Subjective Logic [9]. Here, a trust value is modeled by a so-called opinion which consists of three values[1] in the interval between 0 and 1 modeling the belief resp. disbelief in the honesty of a party as well as the uncertainty about it. The sum of these values have always to be 1. The trust values can be computed by means of a metric [22] which is outlined to more detail in Sec. 4. Operators of the logic enable various combinations of trust values. Unfortunately, it does not allow to model time-dependent behavior yet. Relevant aspects of this logic, however, can be easily implemented (cf. e.g., [23]). In contrast, Jones and Firozabadi [10] use a modal logic of action to describe that a party a receiving a piece of information by a party b has to decide based on b's credibility if the information is true. Based on this, they discuss a species of deception in which b exploits a's trusting nature to make him to believe something which b does not believe himself. The trust gaining processes are modeled in a rather descriptive way and can hardly be realized on a computer.

Falcone and Castelfranchi [5] provide an extensive model to describe trust building from a sociological and cognitive-psychological view. Their so-called Socio-Cognitive Model of Trust considers various forms of social dependence between parties which lead to different forms of beliefs (i.e., ability/competence, disposition/availability, unharmfulness, opportunity and danger beliefs). Based on these forms of beliefs, methods to rate the degree of trusts and to perform decisions are discussed. A formalization of the model based on fuzzy cognitive maps is introduced in [11]. Mezzetti [8] defined a simpler model consisting of a set of rules which specify the building of trust from experience and recommendations based on aspects like competence, willingness, and dependence. As already mentioned, this model reflects time aspects of experiences and uses a decay function.

Other approaches concentrate more on an easy realization of trust values and trust building mechanisms on computers. An early approach to integrate trust issues into computer software emerges from the field of access control. Since traditional access control models are not adequate for the Internet with a large number of fluctuating participants, so-called credential-based systems like PolicyMaker, REFEREE or KeyNote were designed (cf. [12]). Parties interested to access a resource have to pass credentials to the resource provider stating that the credential issuer considers the credential owner as trustworthy. Based on his own trust in the recommendations of the credential issuers, the resource owner decides to provide access or not. Another framework for trust-based policies was developed by Grandison and Sloman [13]. It enables descriptions of trust policies

[1] A forth value to describe the expected probability of an event is less important in our context.

by means of Prolog statements which may be evaluated in order to support trust-based decisions about granting access to certain resources.

Abdul-Rahman and Hailes [14] designed a trust management system to be implemented on the base of mobile agents with strict resource constraints. It enables to rate experiences with a party by different values. The trust in recommendations of a party is computed by the so-called semantic distance describing the difference between the values of a recommendation and the later evaluation of the recommended party. Thus, the semantic distance describes the bias of the recommender towards the recommended party. Later recommendations of the recommender are adapted by adding the semantic distance to it. Azzedin and Maheswaran [15] developed a system to negotiate trust in grid computing systems. They rate recommenders based on advice of so-called trusted allies and compute the direct trust in a party from recommendations by considering the recommender's trust values. The dependency on the trusted allies seems to be a major drawback of this approach. Another contribution to the use of trust management in completely decentralized environments is provided by Aberer and Despotovic [16] who define trustworthiness of a party by the absence of complaints on it. The core of model consists of methods to store and retrieve complaints on a party in a distributed way. The problem of this approach is that the ratio between complaints and the overall number of transactions cannot be retrieved. Thus, uncertainty about a party is handled as high trust which seems not to be a good perception of the reality.

Of these interesting and innovative models we consider Jøsang's Subjective Logic, the formalization of Falcone's and Castelfranchi's Socio-Cognitive Model of Trust, Mezzetti's work as well as the more practical-oriented approaches as well suited to be integrated into our cTLA-based approach. One may even think to combine different models in order to get a better description of the actual building of trust. For instance, a combination of different approaches is used in our example models in which we combine elements of Jøsang's Subjective Logic [9] with a decay function similar to that proposed by Mezzetti (cf. [8]).

3 cTLA

TLA [18] is a linear time temporal logic describing properties of state transition systems by means of often lengthy and complex canonical formulas. To provide a better understanding of specifications, in contrast, cTLA [17, 20] omits the canonical parts of TLA formulas. It is oriented at programming languages and introduces the notion of processes. A specification is structured into modular definitions of process types. An instantiation of a process type forms a process which either has the form of a simple process or that of a process composition. Simple processes, which directly refer to state transition systems, are used to model single system resources or system constraints.

Fig. 1 depicts the example of a simple process type used to model a part of our example trust systems. The header declares the process type name (e.g., *PolicyDecider* and generic module parameters (e.g., *beliefThreshold*). These pa-

```
PROCESS PolicyDecider (beliefThreshold : real;
                       disbeliefThreshold : real)
CONSTANTS
  TrustValues ≜ [[ b : real; d : real; u : real ]];
BODY
  VARIABLES
    policy : {"lowTrust","highTrust"};
  INIT ≜ policy = "lowTrust";
  ACTIONS
    retrievePolicy (p : {"lowTrust", "highTrust"}) ≜
      p = policy ∧ policy′ = policy;
    CONT (INPUT i : TrustValues) ≜
      policy′ = IF i.b ≥ beliefThreshold ∧ i.d ≤ disbeliefThreshold
                THEN "highTrust"
                ELSE "lowTrust";
END
```

Fig. 1. cTLA Process Type *Policy Decider*

rameters facilitate the modeling of similar but not identical processes by a single process type specification. The part headed by the keyword *CONSTANTS* enables the definition of constant expressions (e.g., the record type *TrustValues*).

The process type body defines the state transition system. The state space is specified by state variables (e.g., *policy*) and the subset of initial states is modeled by the predicate *INIT*. Moreover, the body contains actions. An action (e.g., *retrievePolicy*) is a predicate on pairs of current and next states and specifies a set of state transitions. The state variables referring to the current state are noted in simple form (e.g., *policy*) while variables describing the successor state occur in the primed form (e.g., *policy'*). An action may have action parameters enabling to specify different actions by a single representation. The disjunction of the actions forms the next state relation of the process. In the course of time, a process may perform action steps (i.e., it changes its state in accordance with an action) or stuttering steps (i.e., it does not change its state while the environment performs a state transition).

Following [24], real-time is represented by means of a real-valued state variable *now* which is incremented lively by a clock action *tick*. Unlike other variables, which are private in exactly one process, *now* can be read by all processes of a system. Additional real-time constructs specify minimum waiting times and maximum reaction times for actions.

Continuous properties of a process are expressed by means of the special action type *CONT*. The *CONT*-actions of all processes modeling a system and the *tick*-action of the clock occur simultaneously. A *CONT*-action specifies difference equations and, since an execution corresponds to a very small time step, continuous behavior is approximated well. In the difference equations, we express the time steps by *now'-now*. The inputs and outputs of continuous processes are

```
PROCESS OneUserReputationSystem
CONSTANTS
  TrustValues ≜ [[ b : real; d : real; u : real ]];
PROCESSES
  E  : TrustValueEngine(0.01,0.04,0.001,0.004);
  PD : PolicyDecider(0.99,0);
ACTIONS
  reportGoodExperience ≜
    E.reportGoodExperience ∧ PD.stutter;
  reportBadExperience ≜
    E.reportBadExperience ∧ PD.stutter;
  retrievePolicy (p : {"lowTrust", "highTrust"}) ≜
    PD.retrievePolicy(p) ∧ E.stutter;
  CONT (OUTPUT o : TrustValues) ≜
    E.CONT(; o) ∧ PD.CONT(o; );
END
```

Fig. 2. System *One User Reputation System*

modeled by action parameters. In Fig. 1, the variable *policy* is set according to the *IF-THEN-ELSE*-statement depending on input value *i*.

Systems and subsystems are described as compositions of concurrent process instances. The coupling of the processes is specified by synchronously executed process actions while, with exception of *now*, the process variables are encapsulated and cannot be read or modified by other processes. In consequence, a system state is the vector of the process variables. The system transitions are modeled by system actions and each process contributes to a system action by either exactly one process action or a stuttering step. In consequence, a system action is a conjunction of process actions and process stuttering steps. Fig. 2 shows an example of a system specification. In the part *PROCESSES*, the processes of the system are listed as instantiations of process types (e.g., process E of the type *TrustValueEngine* and process *PD* of the process type *PolicyDecider* depicted in Fig. 1). In addition, the instantiations of the module parameters are listed (e.g., the module parameters *beliefThreshold* and *disbeliefThreshold* of process *PD* are replaced by the values 0.99 resp. 0).

In the part headed by *ACTIONS*, the system actions are defined as conjunctions of process actions and stuttering steps. For instance, the system action *retrievePolicy* models that process *PD* performs its process action *retrievePolicy* while E carries out a stuttering step.

4 Simple Trust Management Model

As already mentioned, we introduce the application of cTLA to specify trust models by means of two example systems. The first trust management model describes a very simple reputation-based policy decision system. In particular, users report positive and negative experience reports about a party in question from which trust values are computed. Based on the trust values, the system

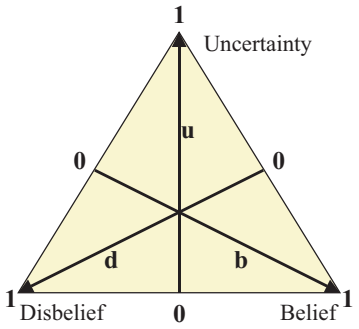

Fig. 3. Opinion Triangle (taken from [25])

selects one out of two trust-based policies to be used, for instance, to decide about granting access to a resource. The system specification is a composition of two cTLA processes. One of them models the collection of experience reports and the computation of the trust values while the other specifies the trust policy selection.

The specification and computation of the trust values is based on Jøsang's Subjective Logic [9]. There, trust values are described as so-called opinions which are triples of real values b, d and u in the range between 0 and 1. b and d state the belief resp. disbelief in a party while u describes uncertainty. Thus, one can distinguish if the lack of trust results from malicious experience or from missing knowledge about a party. Since a trust value fulfills the constraint $b+d+u=1$, it can be modeled by a point in the so-called opinion triangle (cf. Fig. 3). A trust value stating a high degree of uncertainty is described by a point close to the top of the triangle while points on the right or left bottom state great belief resp. disbelief based on a lot of experience with a party.

Trust values are used to describe both the direct trust in a party itself and the trust in the recommendation of a party about another one. Jøsang and Knapskog introduce the following metric [22] to compute trust values from the number p of positive valuations and n of negative valuations of the party in question:

$$b = \frac{p}{p+n+1} \qquad d = \frac{n}{p+n+1} \qquad u = \frac{1}{p+n+1} \tag{1}$$

Unfortunately, this metric does not reflect the time when an experience report was handed over. In reality, the trust resp. distrust in a party is definitely higher if it is based on more recent experience in comparison to older impressions which leads to a higher degree of uncertainty (cf. e.g., [5]). Therefore, we combine the trust value-computation metric with a decay function reducing the numbers p of positive and n of negative experience reports in the cause of time. In order to model the decay per time-unit in a flexible way, we do not describe it by a fixed function but enforce that it has to stay within certain borders. Therefore we define four values *pDMin*, *pDMax*, *nDMin* and *pDMax* describing the minimum resp. maximum decay rates of positive and negative valuations as stated in the

```
PROCESS TrustValueEngine (pDMin, pDMax, nDMin, nDMax : real)
CONSTANTS
  TrustValues ≜ [[ b : real; d : real; u : real ]];
BODY
  VARIABLES
    p : real;
    n : real;
  INIT ≜ p = 0 ∧ n = 0;
  ACTIONS
    reportGoodExperience ≜
      p′ = p + 1 ∧ n′ = n;
    reportBadExperience ≜
      n′ = n + 1 ∧ p′ = p;
    CONT (OUTPUT o : TrustValues) ≜
      o = [[ b ↦ p / (1 + p + n); d ↦ n / (1 + p + n);
             u ↦ 1 / (1 + p + n) ]] ∧
      p′ ≤ max(0,p - (now′-now) · pDMin) ∧
      p′ ≥ max(0,p - (now′-now) · pDMax) ∧
      n′ ≤ max(0,n - (now′-now) · nDMin) ∧
      n′ ≥ max(0,n - (now′-now) · nDMax);
END
```

Fig. 4. Process Type *Trust Value Engine*

following formula[2]:

$$\begin{aligned} p - \Delta t \cdot pDMax \leq p' \leq p - \Delta t \cdot pDMin \\ n - \Delta t \cdot nDMax \leq n' \leq n - \Delta t \cdot nDMin \end{aligned} \tag{2}$$

The time-related generation of trust values is modeled by the cTLA process type *TrustValueEngine* depicted in Fig. 4. Here, the values *pDMin*, *pDMax*, *nDMin* and *pDMax* determining the minimum resp. maximum decays of the experiences are specified as module parameters. The constant expression *TrustValues* defines the three tuple used to model trust values as a record. Moreover, we specify the numbers p of positive resp. n of negative experiences by two variables of the type *real* which both carry the value 0 initially.

The state changes modeled by instances of this process type are specified by means of two atomic actions describing discrete state changes and the special action *CONT* defining continuous behavior. The action *reportGoodExperience* models the reception of a positive valuation. It increments the variable p by 1 while n remains unchanged. In similar, the action *reportBadExperience* describes the submission of a negative experience report resulting in an increment of n by 1. As discussed in the introduction, we consider the computation of the trust values and the "forgetting" of older experience reports as a continuous process. Therefore the corresponding behavior is specified by the action *CONT* modeling very small time steps *now'-now* (cf. Sec. 3). The action contains an output parameter o describing the current trust value which may be used as an input for

[2] Following the cTLA style, p' and n' refer to the next state.

other cTLA processes. The calculation of o from the variables p and n according to the metric introduced in formula 1 is described by the first conjunct of the action. The other conjuncts model the decay of p and n by the inequations listed in formula 2. Since neither p nor n must get values below 0, we added the function *max* to the inequations.

The selection of a low trust resp. a high trust policy based on the current trust value is specified by the cTLA process type *PolicyDecider* listed in Fig. 1. The policy selection is guided by two thresholds *beliefThreshold* and *disbeliefThreshold* which are introduced by means of module parameters. The variable *policy* describes the currently active policy. Initially it is set to *"lowTrust"* stating that in the first state the low trust policy is active. The action *retrievePolicy* enables external cTLA processes to read the current policy. It contains a parameter p describing the current value of the variable *policy* which is not changed by the action. The adjustment of the currently active policy based on the trust value is a continuous process and therefore modeled by the action *CONT*. We assume that the high trust policy is only available if the belief element b of the current trust value, which is modeled by the import parameter i, is not lower than *beliefThreshold* while the disbelief element d must not exceed *disbeliefThreshold*.

The example system is modeled by the process type *OneUserReputation System* depicted in Fig. 2. It consists of instances E of the process type *TrustValueEngine* and PD of *PolicyDecider*. The module parameters of E are instantiated in a way that the decay limits of positive experiences are 0.01 and 0.04 while those of the negative experiences are 0.001 resp. 0.004. Thus, a positive experience is "forgotten" in between 25 and 100 time units while a negative one is lost after between 250 and 1000 time units[3]. The parameter instantiations of PD state that the high trust policy is only used if the belief b in the current trust value is at least 0.99 while the disbelief has to be 0. Thus, one needs at least 99 positive but no negative valuations to run the high trust policy.

The couplings of the process actions to system actions are straightforward. The system actions *reportGoodExperience* and *reportBadExperience* are conjunctions of the corresponding process actions of E and stuttering steps of PD while *retrievePolicy* is coupled the other way round. The interaction between the two process instances is basically an exchange of the current trust value which is modeled by the coupling of the two process actions *CONT* where the output parameter of action $E.CONT$ and the input parameter of $PD.CONT$ are identical. In consequence, in the system action *CONT*, the parameters of both process actions are set to the system action parameter o.

5 Refined Trust Management Model

The simple trust management model introduced above includes only one single trust value engine and, thus, does not distinguish whether the experience reports are submitted by one or more users. In contrast, the model introduced below uses

[3] These settings do not reflect well-founded experience about trust gaining processes but are only used to exemplify the application of cTLA.

```
PROCESS ConsensusOperator
CONSTANTS
  TrustValues ≜ [[ b : real; d : real; u : real ]];
BODY
  INIT ≜ True;
  ACTIONS
    CONT (INPUT i1, i2 : TrustValues; OUTPUT o : TrustValues) ≜
      LET κ ≜ i1.u + i2.u - i1.u · i2.u;
      IN o = [[ b ↦ (i1.b · i2.u + i2.b · i1.u) / κ;
                d ↦ (i1.d · i2.u + i2.d · i1.u) / κ;
                u ↦ (i1.u · i2.u) / κ ]];
END
```

Fig. 5. Process Type *Consensus Operator*

two trust value engines in order to distinguish between two separate sources for valuations. Thus, two trust values ω_1 and ω_2 are created and have to be combined to a third trust value ω in order to determine the active trust policy. An adequate means to compute ω fairly from ω_1 and ω_2 is the consensus operator $\oplus$ introduced in Jøsang's Subjective Logic [9]. If $\omega_1 = (b_1, d_1, u_1)$ and $\omega_2 = (b_2, d_2, u_2)$ are two trust values stating trust in a party based on two separate sources, one can describe by this operator a consensus of these two opinions. The operator is specified by the formula

$$\omega = \omega_1 \oplus \omega_2 \mathrel{\widehat{=}} ((b_1u_2 + b_2u_1)/\kappa, (d_1u_2 + d_2u_1)/\kappa, (u_1u_2)/\kappa) \tag{3}$$

in which κ is defined as $u_1 + u_2 - u_1u_2$. Since the consensus-operator is commutative and associative, it can be used to combine trust values from various sources.

The operator is specified by the cTLA process type *ConsensusOperator* listed in Fig. 5. Since the process type models only stateless behavior, it does not contain variables. The action *CONT* specifies formula 3 in which the operands are specified by the input parameters *i1* and *i2* while the result corresponds to the output parameter *o*. The *LET-IN*-construct enables the definition of constant expressions (e.g., κ) in the *LET*-section which can be used in the formula listed behind the keyword *IN*.

The refined system is modeled by the cTLA process type *TwoUserReputationSystem* depicted in Fig. 6. The system contains two trust value engines specified by the processes *E1* and *E2*. In contrast to the simple system in Sec. 4, the engines uses maximum decay rates of only 0.02 resp. 0.002. Thus, a positive experience is lost not before 50 and a negative one not not before 500 time units passed. The system also contains a process *CO* modeling the consensus operator. Finally, we use a policy decider which is modeled by the process *PD* and uses the same parameter instantiations as in the simple system.

Due to the use of two trust value engines, the system specification defines each two actions to model the reception of positive resp. negative experiences. Moreover, it declares an action to retrieve trust policies and the action *CONT*

```
PROCESS TwoUserReputationSystem
CONSTANTS
  TrustValues ≜ [[ b : real; d : real; u : real ]];
PROCESSES
  E1 : TrustValueEngine(0.01,0.02,0.001,0.002);
  E2 : TrustValueEngine(0.01,0.02,0.001,0.002);
  CO : ConsensusOperator;
  PD : PolicyDecider(0.99,0);
ACTIONS
  reportGoodExperience1 ≜
    E1.reportGoodExperience ∧ E2.stutter ∧ CO.stutter ∧ PD.stutter;
  reportGoodExperience2 ≜
    E2.reportGoodExperience ∧ E1.stutter ∧ CO.stutter ∧ PD.stutter;
  reportBadExperience1 ≜
    E1.reportBadExperience ∧ E2.stutter ∧ CO.stutter ∧ PD.stutter;
  reportBadExperience2 ≜
    E2.reportBadExperience ∧ E1.stutter ∧ CO.stutter ∧ PD.stutter;
  retrievePolicy (p : {"lowTrust", "highTrust"}) ≜
    PD.retrievePolicy(p) ∧ E1.stutter ∧ E2.stutter ∧ CO.stutter;
  CONT (OUTPUT o1, o2, o : TrustValues) ≜
    E1.CONT(; o1) ∧ E2.CONT(; o2) ∧ CO.CONT(o1, o2; o) ∧ PD.CONT(o; );
END
```

Fig. 6. System *Two User Reputation System*

defining the continuous trust value computation. Here, the outputs *o1* and *o2* of the trust value engines are the operands of the consensus operator. The result o of this operator forms the input of the policy decider. Thus, we model that the active trust policy is determined based on the experience reports of both trust value engines, the generated trust values of which are combined by means of the consensus operator.

6 Example Proof

The verification, that the more complex trust management system introduced in Sec. 5 is a correct implementation of the simpler model discussed in Sec. 4, is performed by means of a temporal logic implication proof. In particular, we verify that an instance T of process type *TwoUserReputationSystem* (cf. Fig. 6) implies an instance O of *OneUserReputationSystem* (cf. Fig. 2). Due to the superposition property of cTLA, this verification can be reduced to three separate proof steps:

1. A subsystem $\widehat{T}$ of T implies the process E of O.
2. The process PD of T implies the process PD of O.
3. The actions of the processes in T are coupled in consistence with those in O.

The subsystem $\widehat{T}$ used in the first proof step consists of the processes and action couplings of *TwoUserReputationSystem* with the exception of the process

PD. In consequence, we omitted the system action *retrievePolicy* and the other system actions include the same conjuncts as their counterparts in *TwoUserReputationSystem* except for those referring to *PD*.

A problem of the proof $\widehat{T} \Rightarrow O.E$ is that $\widehat{T}$ and $O.E$ contain different state types. Therefore we define a so-called refinement mapping (i.e., a function mapping the state space of $\widehat{T}$ to that of $O.E$ which has to contain some side properties to be proven below; cf. [26]). Here, we define the refinement mapping by the following formulas:

$$O.E.p \mathrel{\widehat{=}} T.E1.p + T.E2.p \qquad O.E.n \mathrel{\widehat{=}} T.E1.n + T.E2.n \tag{4}$$

Thus, the numbers of positive and negative valuations in the simple model corresponds to the sum of the numbers of experience reports stored by the two trust value engines in the refined model. Now we can start to carry out the first proof step, for which we have to verify that the initial states of $\widehat{T}$ are mapped to initial states of $O.E$ and that each action of $\widehat{T}$ implies either an action of $O.E$ or a stuttering step.

The first proof $\widehat{T}.INIT \Rightarrow O.E.INIT$ is merely trivial since $\widehat{T}.INIT$ implies $T.E1.p = 0 \wedge T.E1.n = 0 \wedge T.E2.p = 0 \wedge T.E2.n = 0$ which according to the refinement mapping in formula 4 implies $O.E.p = 0 \wedge O.E.n = 0$. This, however, is exactly the definition of $O.E.INIT$.

At next, we verify that the action $\widehat{T}.reportGoodExperience1$ implies $O.E.reportGoodExperience$. From $\widehat{T}.reportGoodExperience1$ we can infer $T.E1.p' = T.E1.p + 1$ while all other variables do not change their values. Due to the refinement mapping, however, $O.E.p' = O.E.p + 1 \wedge O.E.n' = O.E.n$ holds which implies $O.E.reportGoodExperience$. Similarly, we can prove that $\widehat{T}.reportGoodExperience2$ implements $O.E.reportGoodExperience$ and that the two actions, modeling reception of negative experience reports in $\widehat{T}$, imply $O.E.reportBadExperience$.

In the next step, which is the most complex of the overall proof, we verify that the action $CONT$ of $\widehat{T}$ implies $O.E.CONT$ (i.e., $\forall o1, o2, o \in TrustValues : \widehat{T}.CONT(; o1, o2, o) \Rightarrow O.E.CONT(; o)$). In particular, we prove for each conjunct of $O.E.CONT$ that it is fulfilled by $\widehat{T}.CONT$. To verify the first conjunct of $O.E.CONT$, we prove firstly that $\widehat{T}.CONT$ implies the setting of the record element $o.b$ (i.e., $o.b = O.E.p/(1 + O.E.p + O.E.n)$). For clarity, we use in this proof the two auxiliary constants $\lambda_1 = 1 + \widehat{T}.E1.p + \widehat{T}.E1.n$ and $\lambda_2 = 1 + \widehat{T}.E2.p + \widehat{T}.E2.n$. From $\widehat{T}.CONT$, we can infer the following formulas:

$$o1.b = \frac{\widehat{T}.E1.p}{\lambda_1} \qquad o2.b = \frac{\widehat{T}.E2.p}{\lambda_2} \qquad o.b = \frac{o1.b \cdot o2.u + o2.b \cdot o1.u}{o1.u + o2.u - o1.u \cdot o2.u} \tag{5}$$

By inserting the first two of these formulas into the third one, we get the following result:

$$o.b = \frac{\frac{\widehat{T}.E1.p}{\lambda_1} \cdot \frac{1}{\lambda_2} + \frac{\widehat{T}.E2.p}{\lambda_2} \cdot \frac{1}{\lambda_1}}{\frac{1}{\lambda_1} + \frac{1}{\lambda_2} - \frac{1}{\lambda_1 \lambda_2}} = \frac{\widehat{T}.E1.p + \widehat{T}.E2.p}{\lambda_1 + \lambda_2 - 1} \tag{6}$$

By application of the refinement mapping in formula 4 we can, however, infer that the right term of formula 6 implies $o.b = O.E.p/(1 + O.E.p + O.E.n)$ which

was the goal of this partial proof. In a very similar way, we can also prove that $\widehat{T}$ implies the settings of $o.d$ and $o.u$ which guarantees that the first conjunct of $O.E.CONT$ is correctly implemented by $\widehat{T}$.

To verify the four other conjuncts of $O.E.CONT$ describing the decay of the variables $O.E.p$ and $O.E.n$, we use the following inequations which can easily be proven by means of a case separation:

$$\begin{aligned}&\forall a, b, k \in real : a, b, k \geq 0 \Rightarrow \\ &\quad max(0, a + b - 2k) \leq max\ (0, a - k) + max(0, b - k) \leq max(0, a + b - k)\end{aligned} \tag{7}$$

To prove, for instance, the third conjunct of $O.E.CONT$, we use the fact that $\widehat{T}.CONT$ implies both $\widehat{T}.E1.p' \geq max(0, \widehat{T}.E1.p - (now' - now) \cdot 0.02$ and $\widehat{T}.E2.p' \geq max(0, \widehat{T}.E2.p - (now' - now) \cdot 0.02$. By a simple invariant proof, we can show that both $\widehat{T}.E1.p$ and $\widehat{T}.E2.p$ are never smaller than 0. Therefore we can apply the left inequation of formula 7 and verify that $\widehat{T}.E1.p' + \widehat{T}.E2.p' \geq max(0, \widehat{T}.E1.p + \widehat{T}.E2.p - (now' - now) \cdot 0.04$ which, according to the refinement mapping, corresponds to the conjunct to be proven. After verifying the remaining conjuncts of $O.E.CONT$ in a rather similar way, we finished the formal proof that $O.E.CONT$ is fulfilled by $\widehat{T}.CONT$. Since, moreover, stuttering steps of $\widehat{T}$ trivially imply stuttering steps of $O.E$, we now achieved the first proof step $\widehat{T} \Rightarrow O.E$ entirely.

The second proof step stating $T.PD \Rightarrow O.PD$ is very simple since both $T.PD$ and $O.PD$ use the same module parameter instantiation. Therefore, the two cTLA process instances are identical and $T.PD$ implies $O.PD$ trivially.

The third proof step states that the actions of the two systems are being consistently coupled. A consistent coupling is guaranteed if the process actions participating in a system action of T are mapped to process actions all being coupled to the same system action of O. This proof, however, is also merely trivial since we can apply the intermediate results of the first two proof steps. For instance to verify $T.reportGoodExperience1 \Rightarrow O.reportGoodExperience$ we use the already carried out implication proofs $\widehat{T}.reportGoodExperience1 \Rightarrow O.E.reportGoodExperience$ and $T.PD.stutter \Rightarrow O.PD.stutter$ which directly implies the assertion. By finishing the third proof step, however, we achieved the overall refinement proof stating that the complex system is a correct implementation of the simple one. A sketch of the complete proof is listed on the WWW (URL: `http://www.item.ntnu.no/~herrmann/specs/trust`).

7 Concluding Remarks

Above, we introduced the use of the temporal logic cTLA to specify trust models and to perform refinement proofs. We can also apply cTLA to specify more detailed specifications of computer implementations and to verify that the realizations fulfill the trust models. For instance, the more complex example trust model can be refined to a model which does not specify continuous behavior. Here, the actions *CONT* are replaced by discrete system actions which are repeatedly executed and computes the decay of the experience numbers and calculates

the trust values based on the time-step since the last execution. The cTLA processes of the implementation and the refinement proof could not be presented here due to the page limit but can be retrieved from our web page.

Due to the compositionality of cTLA, specifications can be designed and refinement proofs can be carried out in a relatively simple way. Nevertheless, one can facilitate the use of cTLA even more by creating so-called specification frameworks [17]. Here, cTLA process types describing aspects of a certain application domain are collected in repositories. The framework user creates system specifications by simply taking suitable process types from the framework, instantiating their module parameters and composing them to a system model. Moreover, a specification framework contains repositories of theorems. A theorem is proven by the framework designer and states that an instance of a cTLA framework process type is fulfilled by a certain subsystem consisting of other framework process instances. The framework user can reduce a refinement proof into proof steps which correspond directly to the framework theorems. Thus, the verification effort is reduced to some simple checks guaranteeing that a certain theorem can be applied in a particular proof. These checks can be automated and tool support is available. cTLA-based specification frameworks were realized for telecommunication protocols [17], hazard analysis of hybrid technical systems [20] and security proofs of component-structured software [27]. Another framework to describe trust-based systems and trust models is under development and will be presented elsewhere.

References

1. McKnight, D.H., Chervany, N.L.: The Meanings of Trust. Working Paper Series 96–04, University of Minnesota — Carlson School of Management (1996)
2. Gambetta, D.: Can We Trust Trust? In Gambetta, D., ed.: Trust: Making and Breaking Cooperative Relations. Basil Blackwell (1990) 213–238
3. Jøsang, A.: The right type of trust for distributed systems. In: Proc. UCLA New Security Paradigms Workshop, Lake Arrowhead, ACM (1996) 119–131
4. Falcone, R., Castelfranchi, C.: The socio-cognitive dynamics of trust: Does trust create trust? In Falcone, R. et al., eds.: Proc. 4th Workshop on Agents — Trust in Cyber-Societies. LNCS 2246, Barcelona, Springer-Verlag (2001) 55–72
5. Falcone, R., Castelfranchi, C.: Social Trust: A Cognitive Approach. In Castelfranchi, C., Tan, Y.H., eds.: Trust and Deception in Virtual Societies. Kluwer Academic Publishers (2001) 55–90
6. Jøsang, A., Keser, C., Dimitrakos, T.: Can We Manage Trust? In Herrmann, P. et al., eds.: Proc. 3rd International Conference on Trust Management (iTrust 2005). LNCS 3477, Paris, Springer-Verlag (2005) 93–107
7. Cheskin Research and Studio Archetype/Sapient: eCommerce Trust Study. (1999)
8. Mezzetti, N.: A Socially Inspired Reputation Model. In Katsikas, S.K. et al., eds.: 1st European Workshop on Public Key Infrastructure (EuroPKI 2004). LNCS 3093, Samos Island, Springer-Verlag (2004) 191–204
9. Jøsang, A.: A Logic for Uncertain Probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9** (2001) 279–311

10. Jones, A.J.I., Firozabadi, B.S.: On the Characterisation of a Trusting Agent — Aspects of a Formal Approach. In Castelfranchi, C., Tan, Y.H., eds.: Trust and Deception in Virtual Societies. Kluwer Academic Publishers (2001) 157–168
11. Falcone, R., Pezzulo, G., Castelfranchi, C.: A Fuzzy Approach to a Belief-Based Trust Computation. In: AAMAS 2002 International Workshop on Trust, Reputation, and Security. LNCS 2631, Bologna, Springer-Verlag (2003)
12. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proc. 17th Symposium on Security and Privacy, Oakland, IEEE (1996) 164–173
13. Grandison, T., Sloman, M.: Specifying and Analysing Trust for Internet Applications. In: Proc. 2nd IFIP Conference on E-Commerce, E-Business & E-Government (I3E), Lisbon, Kluwer Academic Publisher (2002) 145–157
14. Abdul-Rahman, A., Hailes, S.: Supporting Trust in Virtual Communities. In: Proc. 33rd Hawaii International Conference. Volume 6., Maui, Hawaii, IEEE Computer Society Press (2000)
15. Azzedin, F., Maheswaran, M.: A TrustBrokering System and Its Application to Resource Management in Public-Resource Grids. In: Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, IEEE Computer Society Press (2004)
16. Aberer, K., Despotovic, Z.: Managing Trust in a Peer-2-Peer Information System. In Paques, H. et al., eds.: Proc. 10th International Conference on Information and Knowledge Management (CIKM'01), New York, ACM Press (2001) 310–317
17. Herrmann, P., Krumm, H.: A Framework for Modeling Transfer Protocols. Computer Networks **34** (2000) 317–337
18. Lamport, L.: Specifying Systems. Addison-Wesley (2002)
19. Vissers, C.A., Scollo, G., van Sinderen, M.: Architecture and specification style in formal descriptions of distributed systems. In Agarwal, S., Sabnani, K., eds.: Proc. 8th IFIP International Conference on Protocol Specification, Testing and Verification (PSTV'88), Elsevier (1988) 189–204
20. Herrmann, P., Krumm, H.: A Framework for the Hazard Analysis of Chemical Plants. In: Proc. 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD2000), Anchorage, IEEE CSS, Omnipress (2000) 35–41
21. Kurki-Suonio, R.: A Practical Theory of Reactive Systems — Incremental Modeling of Dynamic Behaviors. Springer-Verlag (2005)
22. Jøsang, A., Knapskog, S.J.: A metric for trusted systems. In: Proc. 21st National Security Conference, NSA (1998)
23. Herrmann, P.: Trust-Based Protection of Software Component Users and Designers. In Nixon, P., Terzis, S., eds.: Proc. 1st International Conference on Trust Management. LNCS 2692, Heraklion, Springer-Verlag (2003) 75–90
24. Abadi, M., Lamport, L.: An old-fashioned recipe for real time. In de Bakker et al., eds.: Real-Time: Theory in Practice. LNCS 600, Springer-Verlag (1991)
25. Jøsang, A.: An Algebra for Assessing Trust in Certification Chains. In Kochmar, J., ed.: Proc. Network and Distributed Systems Security Symposium (NDSS'99), The Internet Society (1999)
26. Abadi, M., Lamport, L.: The Existence of Refinement Mappings. Theoretical Computer Science **82** (1991) 253–284
27. Herrmann, P.: Formal Security Policy Verification of Distributed Component-Structured Software. In König, H. et al., eds.: Proc. 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'2003). LNCS 2767, Berlin, Springer-Verlag (2003) 257–272