Design of Trusted Systems with Reusable Collaboration Models

Peter Herrmann and Frank Alexander Kraemer

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

Abstract. We describe the application of our collaboration-oriented software engineering approach for the design of trust-aware systems. In this model-based technique, a specification does not describe a physical system component but the collaboration between various components which achieve system functions by cooperation. A system model is composed from these collaboration specifications and, by a set of transformations, executable code can be automatically generated. As a modeling language, we use UML 2.0 collaborations and activities, for which we defined a semantics based on temporal logic. Thus, formal refinement and property proofs can be provided by applying model checkers as well. We consider our approach as well-suited for the development of trust-based systems since the trust relations between different parties can be nicely modeled by the collaborations. This ability facilitates also a tight cooperation between trust management and software engineering experts which are both needed to create scalable trust-aware applications. The engineering approach is introduced by means of an electronic auction system executing different policies which are guided by the mutual trust of its principals. As a trust model we apply Jøsang's Subjective Logic.

1 Introduction

Since the turn of the millenium, the management of trust gained more and more momentum. While this field is inherently multi-disciplinary and researchers from psychology, sociology, philosophy, law, and economics work on trust issues for many years, the computer science seems to be the driving force behind the current advances. An important reason for that is the maturing of the internet-based consumer commerce [1]. The acceptance of e-commerce services depends directly on the trust, the different parties involved in it can build up in each other. In the internet, however, commerce partners are often unknown, live in another country with a different legal system, and are selected on an ad hoc basis guided by the best offer. Therefore, traditional trust building mechanisms like personal experience of oneself or relatives, recommendations by friends, or the general reputation "in town" cannot be used in the same way as in traditional commerce. The trust management community started to overcome this deficiency of the internet by developing trust models consisting of both representations for trust in computers and related mechanisms specifying the building of trust. Some of these models describe trust in a more general way from either a mathematical-philosophical perspective (e.g., [2, 3]) or from a sociological-cognitive view (e.g., [4, 5]). Other approaches are devoted to realize trust building mechanisms which take the practical limits of computer systems and networks into account [6, 7, 8, 9, 10].

The invention of computer-readable trust mechanisms facilitates the design of applications incorporating trust. Most approaches enhance or replace traditional security mechanisms at points where they are not suitable for modern ad hoc-networks. In particular, a number of solutions were developed for access control of both peer-to-peer networks [11, 12, 13] and business processes for web services [14, 15, 16] while other tools approach authorization [17], authentication and identity management [18] as well as privacy [19]. A second field of application design is devoted to federate systems combined of separate partners and, in particular, to determine the kind of mutual protection of the partners. Here, a wide field starting at security-protecting routing algorithms [20] via the formation of virtual organizations [21] to the trust-based protection of component-structured software [22, 23] and the protection of collaborations of pervasive devices [24] is covered. It does not require prophetic skills to expect that there will be a lot more trust-encompassing systems to come in various application domains.

The development of trust-aware systems will, of course, face all the challenges other software systems have (e.g., to guarantee their functional correctness, robustness, and fault-tolerance). As trust-based systems can be quite complex, the application design process has to incorporate typical software engineering techniques. The application of these techniques is normally so difficult, that experienced software engineers are required. In consequence, to design a trust-aware system, we need experts both for the trust management and for software engineering who have to cooperate very closely since the trust management functions of a system are tightly interwoven with the rest of the system logic. This poses the question, how to perform a design of a trust-aware system combining the strengths of trust management system design and software engineering techniques best without affording that one expert needs to have a deep insight into the other's competence. Thus, the trust management developer should be able to integrate trust models into a system design process without necessarily understanding the full application logic while the software designer should be capable to make the general software engineering decisions without comprehending the complete functionality of the underlying trust management model.

We consider our software engineering approach based on collaboration-oriented formal system models [25] well-suited to answer this question. Most modeling techniques combine system specifications from models specifying a separate physical software component each. In contrast, in our technique a specification building block describes a partial system functionality which is provided by the joint effort of several components cooperating with each other. Every component

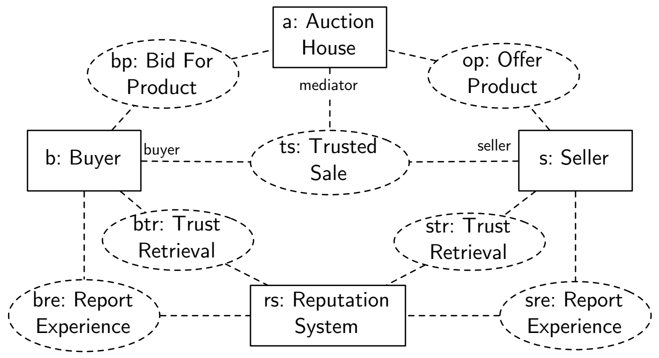

Fig. 1. Collaboration of the *Trusted Auction System*

taking part in a collaboration is represented in the form of a so-called collaboration role. The behavior models of collaborations specify both the interactions between the collaboration roles as well as local behavior of collaboration roles needed to provide the modeled functionality. Instances of collaborations (called collaboration uses) may be composed with each other to more comprehensive collaborations and the system model is the most global one. Thus, hierarchical system models are possible.

As an example, we depict in Fig. 1 the collaboration uses of the outmost hierarchical level to model a trusted electronic auction system which will be introduced in detail in sections 3 and 4. The system specifies a fully automatic internet-based auction system which could, for instance, be built upon the web services offered by eBay. From a trust management perspective, the major problem of such a system is the sale between the winning buyer and the seller after the auction since the reluctance of one party to pay resp. to deliver the product may cause damage to the other side. As a solution, we provide a trust-encompassing application based on a reputation system (e.g., the eBay feedback forum). According to their mutual trust, both parties can decide how to carry out the sale. As a consequence, the example system incorporates four major components, the winning buyer, the seller, the reputation system, and the auction house. Its functionality is expressed by means of seven collaboration uses depicted in Fig. 1. The collaboration use *btr* models the access to the reputation system by the buyer in order to retrieve the current trust of the community in the seller. We will see in Sec. 4 that this retrieval is done before bidding for the product. Likewise, the collaboration use *str* describes the retrieval of the buyer's trust value by the seller which takes place after the auction. According to the mutual trust, the buyer and seller perform the sale which is modeled by *ts*. Indeed, this collaboration is a composition from more basic collaborations specifying four different modes which depend on the trust of the participants in each other. After finishing the sale, both parties report their mutual experiences to the reputation system which is expressed by the collaboration uses *bre* and *sre*. The remaining collaboration uses *op* and *bp* describe the offering of goods by the seller and the bidding of the buyer. As these two collaboration uses are not relevant from a trust management perspective, they are not discussed further.

Fig. 1 is a collaboration in the popular graphical modeling language UML 2.0 (Unified Modeling Language [26, 27]). These diagrams are used to describe the basic structure of a collaboration (i.e., the collaboration uses forming it and the relation between the roles of the collaboration uses and those of the comprehensive collaboration). To specify the behavior of the collaborations and the logic combining collaboration uses is described by UML activities which are introduced in Sec. 3.

As trust relations are inherently collaborative and always comprise at least a trustor and a trustee, we consider the collaboration-oriented specification style very helpful to develop trust-based systems. The reduction of systems to sub-functionalities supports their understanding to a high degree (cf. [25, 28, 29, 30]). As discussed in Sec. 2, we consider this property useful to answer our basic question, i.e., to provide trust management experts and software developers with a fundament for tightly interwoven cooperation. In addition, the model structure enables a higher reuse of collaborations. In many distributed application domains, the system components cooperate with each other by means of a relatively small number of recurrent sub-functionalities which can be specified once and thereafter stored in a library. System developers can create their specifications in a relatively simple way by selecting collaborations from the library, instantiating them, and composing them to a system description. In our example, *btr*, *str*, *bre*, and *sre* are instantiations of the collaborations *Trust Retrieval* resp. *Report Experience* which are suitable building blocks to create applications using reputation systems.

By means of an algorithm [31], we can automatically transform the collaboration-oriented models into executable state machines from which in a second step executable code can be generated [32]. Moreover, we currently develop a transformation to TLA$^+$ [33], the input syntax of the model checker TLC [34] which facilitates formal proofs of system properties. This will be further discussed in Sec. 5. Before that, we discuss in Sec. 2 the benefit of our approach for the generation of trust management-based systems. Thereafter, the specification of collaborations by UML collaboration diagrams and activities is introduced by means of the trusted auction example in Sec. 3. The coupling of collaboration uses to more comprehensive collaborations is outlined in Sec. 4.

2 Trust Management Aspects

In the recent years, a lot of definitions for trust were published. A significant one was introduced by Jøsang [35] who distinguishes between trust in humans and trust in computers. He calls humans as well as organizations formed by humans with a free will *passionate entities*. In contrast, computers and other entities without a free will are named *rational entities*. Then trust in a passionate entity is defined as *“the belief that it will behave without malicious intent”* while trust in a rational entity is *“the belief that it will resist attacks from malicious agents”*. Both definitions have in common that a trustor can only be a passionate entity since trust needs a free will. Nevertheless, in specific application domains both the building of trust and its deployment selecting different policies to deal with the trustee is so rational that it can be handed over to a computer. A good example is the decision making process of banks whether to provide loans or not. A bank’s behavior is basically guided by its trust in a debtor that he will be able to pay a loan back. In consequence, typical trust building mechanisms as the debtor’s behavior in previous cases (i.e., the debtor’s reputation) are taken into account and the decision is made according to fixed policies. Of course, these policies can be implemented on a computer as already applied in some banks.

For the representation of trust one can apply trust values. For instance, Jøsang introduces so-called opinion triangles [2, 36]. These are effectively triples of probability values, the sum of which is always 1. Two of these values describe the belief resp. disbelief in the trustee while the third one states the uncertainty based on missing knowledge on the trustee. The building of trust is, in consequence, described by traces of changing trust values. In between, a lot of trust models being suited for computers were developed [2, 5, 6, 7, 8, 9, 10]. For instance, Jøsang and Knapskog use a certain metric to calculate opinion triangles from a number of positive and negative behaviors [37] while the so-called Subjective Logic [2] can be used to compute trust values from other ones in certain ways. The utilization of trust in dealing with a trustee can also be realized on a computer by defining trust-related policies. The actual policy can then be selected based on the current trust value.

Our collaboration-oriented software development approach is well-suited to model the mechanisms used to describe the building of trust. A collaboration is an appropriate means to describe the various functions of a trust model since every function affects more than one partner. Moreover, the collaborations can be used as building blocks for trust-encompassing applications. For instance, the collaborations *Trust Retrieval* and *Report Experience* used in the trusted auction model (see Fig. 1) describe the two aspects typically used in dealing with a reputation system, i.e., the decision about how to deal with the trustee depending on its current trust value as well as improving the trustee’s assessment by sending the reputation system a positive or negative experience report. Similar collaborations can be defined to model other trust gaining mechanisms as considering the own experience or the recommendation by third parties. In addition, to support the design of more complex trust building mechanisms, one can add special building blocks enabling to combine different trust values (e.g., by specifying the operators of the Subjective Logic).

The method is also useful to tackle the question mentioned beforehand how to simplify the cooperation between the trust management experts and the software engineers. A trust expert can specify the trust building functions of the system on its own by utilizing collaborations from a library. The outcome will be a set of collaboration uses that the software engineers can integrate into the overall system model without fully understanding their internal behavior. They only need to recognize that different trust-based policies are possible but not the steps to decide which actual policy should be used.

Somehow more difficult is the support of the cooperation between the two expert groups in modeling the enforcement of the different trust policies. Here, aspects of the general application functionality and special trust-related properties have to be combined. This can be achieved by a twofold proceeding. First, characteristic trust-based functions may be used to enforce policies. These functions can also be modeled by collaborations and used in several system models. For instance, a sale between two parties with a low degree of trust in each other can be performed by including a trusted third party which mediates the sale by guaranteeing that a buyer cannot receive the product before sending the money while the seller must send the product before receiving the payment. It is easy to model this as a collaboration which can be used by the software engineer without understanding the exact functionality (see also Sec. 4).

Second, the trust expert can inform the software engineer about trust-related functionalities, the application has to follow. For instance, a requirement of the trusted sale should be that the buyer only issues the money transfer to the seller without having evidence of receiving the product in time if her trust in the seller is high. The software engineer considers these properties in the system development. Afterwards, the trust expert can check that the system complies with the properties by, for instance, proving them with the model checker TLC [34]. In the following, we will clarify how trust-based systems like the trusted auction example can be developed using the collaboration-oriented specification style.

3 Activity-Based Collaboration Models

As depicted in Fig. 1, we use UML collaborations to specify the overall structure of system models composed from collaboration uses. In particular, a collaboration describes the different components forming a system and the assignment of the roles of the collaboration uses to the components. To model the behavior of a collaboration, UML offers various diagram types like state machines, sequence diagrams, and activities [27]. We decided to use activities mainly for two reasons: First, activities are based on Petri Nets and model behavior as flows of tokens passing nodes and edges of a graph. This proved to represent flows of behavior quite naturally and is therefore easy to understand (cf. [25]). Second, activities are self-contained. Sequence diagrams, for instance, typically describe in one diagram only a set of system scenarios rather than the complete behavior. In contrast, activities allow to specify the full behavior of a collaboration within one diagram.

A typical example for an activity is *Trust Retrieval* which models the behavior of the collaborations *btr* and *str* in the trusted auction example (see Fig. 1).

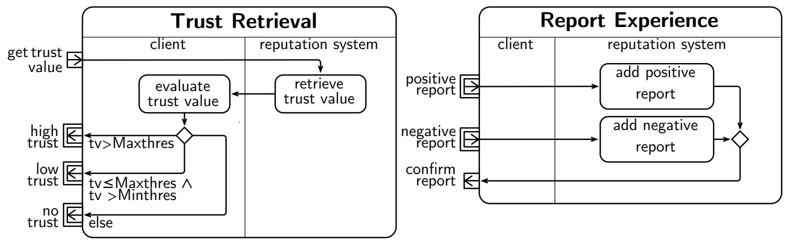

Fig. 2. Activities *Trust Retrieval* and *Report Experience*

It is listed on the left side of Fig. 2 and describes both the access of a caller to a reputation system in order to retrieve a trustee's reputation and the decision about a certain level of trust which may lead to different trust policies. Since the collaboration comprises two different roles, the client of the reputation system and the reputation system itself, we use two activity partitions in the diagram which are named by the role identifiers. The interface of the collaboration to its environment is located at the activity partition of the client and consists of three output pins each describing a certain level of trust[1].

The behavior of the activity is described by a token flow which is started at the input node in the partition of the client. It passes a token from the client via the partition border to the reputation system. The token contains an identifier of the trustee which is computed in the call operation action *retrieve trust value*. This call operation action contains the logic (not further specified here) to access the number of good and bad experiences with the trustee and to generate an opinion triangle based on Jøsang's and Knapskog's metric [37]. The constructed trust value is thereafter forwarded as a token back to the caller and evaluated in the call operation action *evaluate trust value* (i.e., the trust value is copied to the auxiliary collaboration variable *tv*). Thereafter, the token proceeds to a decision node ($\diamond$). The token proceeds on exactly one of the output edges guided by the conditions linked to it. The trust expert defines two thresholds (e.g., for *Maxthres* a belief value greater than 0.95 with a disbelief value of less than 0.02 while for *Minthres* a disbelief value between 0.02 and 0.3 or a disbelief value smaller than 0.95 could be reasonable) and the token is forwarded to the activity environment via one of the output pins *high trust*, *low trust*, or *no trust*. By passing one of the output pins, the overall activity is terminated. A trust management expert can instantiate *Trust Retrieval* simply by defining suitable thresholds.

Activity *Report Experience* (on the right side of Fig. 2) models the report of positive or negative experiences with a trustee to the reputation system adjusting the trustee's reputation. It is started with a token passing one of the input pins *positive report* or *negative report*. The tokens are forwarded to the reputation system which adapts the trustee's data base entry in the call

[1] As these output pins are mutual exclusive, they belong to different parameter sets shown by the additional box around them.

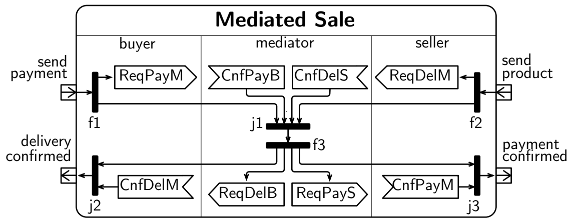

Fig. 3. Activity *Mediated Sale*

operation actions. The edges leaving the two call operation actions lead to a merge node (⋄) that merges its incoming flows by forwarding all incoming tokens to the only outgoing edge. In this way, after registering either a positive or negative report, the token is passed back to the client's output pin *confirm report* describing the confirmation of the experience report.

The activity *Mediated Sale* introduced in Fig. 3 expresses a functionality with several parallel flows. As discussed before, a mediator acts here as a trusted third party which assures a fair sale by collecting the payment and the product which are delivered to their recipients not before both are received by the mediator. The activity consists of three partitions for the buyer, the seller, and the mediator. It is started by two separate tokens coming in from the buyer through the input pin *send payment* and from the seller via *send product*. The token from the buyer heads to the fork node f_1. In a fork node every incoming token is reproduced and one copy is sent via every outgoing edge. One of the tokens leaving f_1 reaches the send action *ReqPayM*. We use send actions to model the transfer of signals to external applications which are not an inherent part of the modeled application. For instance, the accounting unit of the buyer is such an external system which is notified by *ReqPayM* to issue the payment to the mediator. The other token leaving f_1 is forwarded to the mediator which is notified thereby about the start of the payment. Likewise, the seller calls its delivery unit to send the product to the mediator which is expressed by the send action *RegDelM* and notifies the mediator as well. When the payment arrives at the mediator, it is notified by its accounting unit using the receive action *CnfPayM* while *CnfDelS* reports the reception of the product. Similar to send actions, we use receive actions to model incoming signals from the environment. All tokens coming from the two receive actions and from the buyer resp. seller lead to the join node[2] j_1. A flow may only leave a join if tokens arrived on all of its incoming edges before. During the execution of the join, all but one token are removed and only one token leaves it via its outgoing edge. The token leaving j_1 continues to the fork f_3 from which both deliveries to the final recipients and the notifications are issued. Thus, by the combination of j_1 and f_3 we guarantee

[2] UML uses identical symbols for join and fork nodes. They can be distinguished by the number of incoming and outgoing edges. Fork nodes have exactly one incoming edge while join nodes have exactly one outgoing edge.

that deliveries are only carried out if both the payment and the product arrived at the mediator before.

The notification for the buyer heads to the join node j_2 and can only be forwarded if the buyer's delivery unit reports the product's reception which is specified by the receive action $CnfDelM$. The token passing j_2 leaves the activity via the output pin *delivery confirmed*. Likewise, the seller sends a confirmation of the payment via *payment confirmed* after receiving the money. As the two activities introduced before, *Mediated Sale* can be provided by the trust management expert. The only necessary cooperation with the software engineer is to agree about the formats of the transmissions with the various accounting and delivery units.

4 Coupling Activities

Activities are especially powerful for the composition of behaviors from existing ones. This is done by means of call behavior actions that can refer to other activities which are invoked in place of the call behavior action. In this way, activities may be combined to more comprehensive activities. The events of the activities may be coupled using all kinds of control nodes and edges, so that arbitrary dependencies between the sub-activities may be described. As activities are used in our approach to describe the behavior of collaborations, this technique is applied to compose the collaborations behaviorally (while the UML collaboration in Fig. 1 shows the structural aspect of this composition.) An example for a composed activity is *Trusted Sale* in Fig. 4 which is composed from the call behavior actions *ms* and *pc* referring to the behavior of subordinate activities (resp. collaborations).

Trusted Sale describes the functionality of selling a product between a buyer and a seller after finishing an auction. The two parties in the sale may either have a high or a low degree of trust in the other one which is modeled by the two input pins in both the buyer and the seller partition. If the buyer has a high degree of trust in the seller, she is willing to send the payment immediately without waiting for the partner. That is described by the send action *ReqPayS* to which a token is forwarded directly after entering the activity via *buy trusted*. By this send action, the accounting unit of the buyer is notified to start the payment to the seller. Likewise, the seller is ready to send the product to the buyer immediately if he has a high level of trust which is expressed by the flow to the send action *ReqDelB*.

Since both parties may either have high or low trust in the other one, four different trust relations between the two parties are possible and for each one a separate sale policy is defined. Nevertheless, to decide about a sale policy, both parties have to know the partner's trust in themselves. As a mutual distributed combination of policies is a quite common function in many networked systems, we have a collaboration and a corresponding activity *2×2 Policy Combination* available from our general pattern library which can be applied here in the

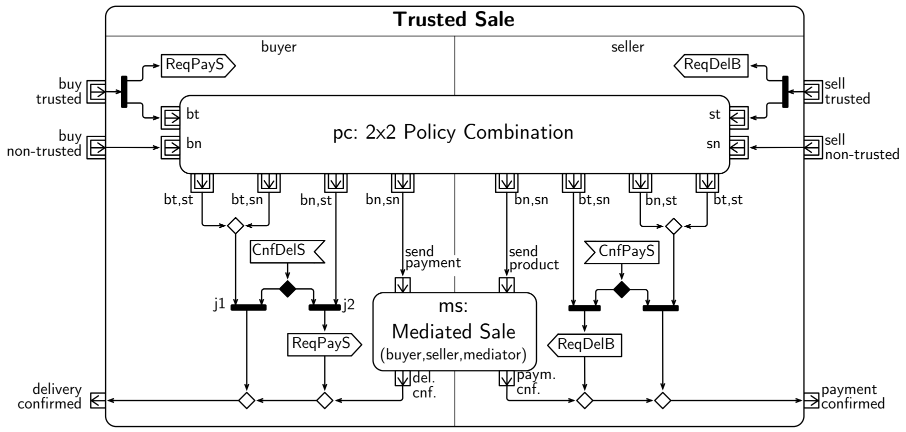

Fig. 4. Activity *Trusted Sale*

form of the call behavior action *pc*. This activity has two input pins and four output pins on each side. The two parties define the selected input policy by transferring a token via the corresponding input pin which causes the delivery of tokens through those output pins describing the combination of the two policies (e.g., if the buyer sends a token via input pin *bt* (for buy trusted) and the seller via *sn* (for sell non-trusted), the tokens will eventually arrive at the output pins *bt,sn*). The input nodes of *Trusted Sale* are connected with the corresponding ones of *pc* and its output pins can be used as the starting points to model the four sale policies (*bt,st; bt,sn; bn,st; bn,sn*):

- If both partners have a high degree of mutual trust (*bt,st*), they simply send the payment resp. the product without waiting for the other. Each partner completes the sale after the delivery arrived. As the payment was already started, the buyer has to wait for a token arriving via output pin *bt,st* in join j_1 for the delivery of the product. The reception of the product is described by the accept signal action *ConfDelS* forwarding a token to j_1 as well[3]. Thus, j_1 can be triggered and a token leaves the activity *Trusted Sale* via the output pin *delivery confirmed* which specifies the completion of the sale on the buyer's side. The behavior in the partition of the seller is similar.
- If the buyer has only a low trust in the seller but the seller a high one in the buyer (*bn,st*), we use a policy in which the seller transfers the product first and the buyer initiates the payment not before receiving the product. Thus, the buyer does not send the payment initially, but waits for the delivery of the product which is expressed by the token in join j_2. After the delivery is notified as modeled by a token heading from *ConfDelS* to j_2, the buyer initiates the payment, which is described by the send action *ReqPayS*, and

[3] Indeed, the token leaving *ConfDelS* is stored first in a so-called waiting decision node (◆, cf. [31]) which forwards it either to join j_1 or j_2 depending on which join can be executed earlier.

finishes the sale. The handling of this policy on the seller's side is identical to the first one since it behaves similarly in both policies.

- If the buyer has a high degree of trust in the seller which, however, trusts the buyer only lowly (*bt,sn*), we use the reciprocal policy to that listed before. Here, the seller does not send the product before receiving the payment. As the effective behavior for the buyer is the same as for the policy (*bt,st*), the flow from *bt,sn* is simply merged into the behavior for *bt,st*.
- If both partners have a low degree of trust in each other (*bn,sn*), they decide to rely on a mediator. This can be modeled by applying the activity *Mediated Sale* introduced in Sec. 3. The pins *bn,sn* are simply connected with the input pins of *Mediated Sale* and its output pins with the output pins of *Trusted Sale*.

When one of the partners cheat by not sending anything, the activity is not finished correctly but stops somewhere. We will see below that this case leads to a negative rating of the partner.

The activity *Trusted Sale* exemplifies the interplay between both groups of competence. The trust management expert provides the software engineer with the activity *Mediated Sale* and describes the four sale policies. Based on this information, the software engineer accomplishes the overall model of the trusted sale which can be added to the library of building blocks for trusted systems facilitating a later usage in other applications.

The last activity introduced here is *Trusted Auction* depicted in Fig. 5 which describes the behavior of the overall system. Each of the collaboration uses, it is composed of (see Fig. 1), are represented by the call behavior actions *btr*, *str*, *bre*, *sre*, and *ts*. While an electronic auction encompasses an arbitrary number of buyers and sellers, we laid out the activity in a way that only the relation between exactly one buyer and one seller is modeled by the activity. In consequence, the whole application is described by multiple instances of *Trusted Auction*. For the sake of brevity, we omitted the part in which the seller registers the product since that is not relevant for trust management. Thus, the activity is started by the buyer, who becomes active if she finds an interesting product. This is expressed by the initial node i_1 from which, at first, the trust level of the seller is retrieved by accessing *btr*. If the reputation of the seller is so bad that there is merely no trust, the buyer decides not to bid and the activity is terminated by a final node (◉). If the buyer trusts the seller at least to some degree, she makes a bid[4] which is modeled by the send action *MakeBid* and waits in the receive node *WinBid* for the end of the bidding. If the bid is not sufficient, a token is received via the accept signal action *LoseBid* and the activity is terminated since no further action is necessary. If the bid won, a token leaves *WinBid* and the trusted sale is started by forwarding a token to *ts*. Moreover, the instance *bto* of activity *Timeliness Observer* is started. It specifies a timeout process to detect late deliveries of the product which will be discussed below.

[4] For brevity, we assume that a buyer makes only one bid in an auction.

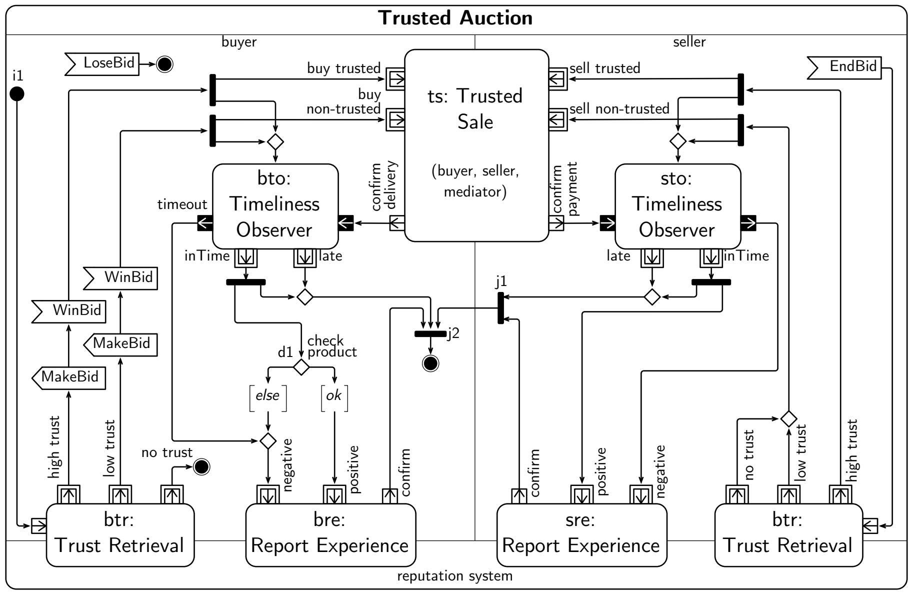

Fig. 5. Activity *Trusted Auction*

On the seller's side, a flow is started after the end of the action as expressed by *EndBid*. Thereafter, the reputation of the buyer is retrieved in *str* and the trusted sale is started as well. Due to the nature of an electronic auction system, the seller has to start the sale process even if he does not trust the buyer at all. Furthermore, *sto* is initiated starting a timer as well. In the case of a timeout, a token leaves the output pin *timeout* immediately, meaning that the payment did not arrive in due time, and via *sre* a negative report on the buyer is sent to the reputation system. The confirmation is forwarded to the join node j_1 used to synchronize the activity termination in the seller partition. If the payment is confirmed, a token proceeds from *ts* to *sto*. If this confirmation arrives at *sto* after the timeout, a token is issued at the output pin *late* which is forwarded to j_1. If the negative report was already confirmed, j_1 can fire which notifies the buyer's side that the seller can accept to terminate the activity. If the payment confirmation arrives timely, a token leaves the output pin *in Time* of *sto* issuing a positive report about the buyer. In addition, a token is forwarded to j_1 such that the buyer can be notified about the readiness for termination after the experience report was confirmed.

The behavior after finishing the sale on the buyer's side is similar except for the decision d_1. We assume that the delivery unit of the buyer attaches information to the token sent to the activity *Trusted Sale* describing if the quality of the product is sufficient. In that case, a positive report is triggered while a bad condition of the product leads to a negative report. The join j_2 can only be executed if both the delivery of the product was confirmed, the report about the seller was attested and the seller reported that it is ready to terminate. The execution of j_2 causes the termination of the activity.

As in the activity *Trusted Sale*, this activity can be developed combining the competence of the two expert groups. The trust management expert delivers the activities describing the access to the reputation system as well as some policies defining, for instance, which reports have to be issued to the reputation system under which circumstances. This provides the software engineer with the sufficient knowledge to develop the behavioral model specified by the activity.

5 Implementation and Verification

The fact that activities render a complete system behavior facilitates automatic generation of code from the collaboration-oriented model which is performed in a series of steps: At first, we apply the algorithm introduced in [31] which transforms the activities into a set of UML state machines each describing a system component. As we defined both the semantics of the activities and the state machines based on the compositional Temporal Logic of Actions (cTLA) [38], the correctness of the transformation could be verified by a cTLA refinement proof sketch (cf. [31]). For our example, the algorithm creates separate state machines modeling the behavior of the buyer, the seller, the reputation system, and the auction house acting as mediator. Due to the varying complexity of the four components, the state machines have a quite different size. Since the behavior of the reputation system is stateless, its state machine consists only of one control state and three transitions modeling the retrieval of trust values as well as the addition of positive and negative experience report. In contrast, the state machine of the mediator consists of 15 control states and 33 transitions while that of the buyer models the most complex functionality and consists of 70 states and 93 transitions.

The state machines have a special "executable" form in which, except for the initialization, all transitions are triggered by incoming signals from the environment or from local timers. Since, in addition, the enabling condition of a transition depends only on the control state of the state machine but not on its auxiliary variables, very efficient executable code can be generated. This kind of code generators is built at our department for nearly 30 years now (see, for instance, [39, 40]). To implement our example, we used a generator creating Java code which is executed on the middleware platform JavaFrame [41]. During testing the application, we could not detect any significant overhead. The application of the code generators, the related middleware platforms, and a cTLA-based correctness proof are described in [32].

The trust expert can check if the produced collaboration-oriented model fulfills the trust-related properties passed to the software engineer by applying an animation tool. Moreover, due to defining the semantics of the activities by cTLA, formal refinement and invariant proofs are also facilitated. For instance, the property, that the buyer may only start a payment to the seller immediately if she has high trust in him, can be expressed by an invariant excluding a state in which (1) the trust level is low, (2) the payment was already sent to the seller, (3) and the product is not yet delivered. By a cTLA proof, one can verify that the cTLA formula specifying the activity *Trusted Sale* always fulfills the invariant. In the context of trusted systems, this kind of proofs was introduced in [42]. We currently develop a tool transforming activities directly into the input syntax TLA$^+$ [33] of the model checker TLC [34] carrying out the proofs automatically. Of course, model checkers are subject to the state space explosion problem according to which for a scalable system the number of states to be inspected is too large to be handled by the checker. cTLA, however, supports a coupling style reflecting the activity combinations in a quite natural way. For each activity, a separate cTLA model is created and, in a proof, only those models need to be considered which realize the verified property. For instance, to prove the invariant listed above, only the states of the cTLA model representing the activity *Trusted Sale* must be checked. This quality of cTLA makes our approach not only well-suited for the design and implementation of realistic trust-based systems but also enables formal property proofs in a relatively user-friendly way.

6 Concluding Remarks

In this paper we introduced our collaboration-oriented software development approach which facilitates system modeling by specifying the various cooperations between the system components separately. We consider the approach well-suited for the design of trust-aware systems since trust relations between principals can be directly modeled as collaborations. This property enables the tight cooperation of trust management experts and software engineers without affording a too close insight in the competence of the other expert group. The collaboration-oriented development approach was awarded by the Norwegian research organization Norges Forskningsrådet (NFR) that approved the research and development project ISIS (Infrastructure for Integrated Services). ISIS is mainly devoted to the creation of a tool set supporting the suitable design of collaboration-oriented systems. Besides home automation, the development of trust-based systems shall also play a significant role as an application domain in this project. Moreover, we currently prepare the application to NFR for an add-on research project devoted to combine the methodologies of collaboration-oriented software design and security protocol composition. As a result of this project, we expect methods facilitating the engineering and deployment of secure and trust-aware distributed systems. The work presented above is considered as a major cornerstone for these research goals.

References

1. Cheskin Research and Studio Archetype/Sapient: eCommerce Trust Study. (1999)

2. Jøsang, A.: A Logic for Uncertain Probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9** (2001) 279–311
3. 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
4. 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
5. Mezzetti, N.: A Socially Inspired Reputation Model. In Katsikas, S.K., Gritzalis, S., Lopez, J., eds.: 1st European Workshop on Public Key Infrastructure (EuroPKI 2004). LNCS 3093, Samos Island, Springer-Verlag (2004) 191–204
6. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proc. 17th Symposium on Security and Privacy, Oakland, IEEE (1996) 164–173
7. 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
8. 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)
9. 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
10. 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)
11. Xiong, L., Liu, L.: Building Trust in Decentralized Peer-to-Peer Electronic Communities. In: Proc. 5th International Conference on Electronic Commerce Research (ICECR-5), Dallas, ATSMA, IFIP (2002)
12. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The EigenTrust Algorithm for Reputation Management in P2P Networks. In: Proc. 12th International World Wide Web Conference, Budapest, ACM (2003)
13. Ingram, D.: An Evidence Based Architecture for Efficient, Attack-Resistant Computational Trust Dissemination in Peer-to-Peer Networks. In Herrmann, P. et al., eds.: Proc. 3rd International Conference on Trust Management. LNCS 3477, Paris, Springer-Verlag (2005) 273–288
14. Bonatti, P., Samarati, P.: A Unified Framework for Regulating Access and Information Release on the Web. Journal of Computer Security **10** (2002) 241–272
15. Yu, T., Winslett, M., Seamons, K.E.: Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. ACM Transactions on Information and System Security **6** (2003) 1–42
16. Koshutanski, H., Massacci, F.: Interactive Access Control for Web Services. In: Proc. 19th IFIP Information Security Conference (SEC 2004), Toulouse, Kluwer Academic Publisher (2004) 151–166
17. Lee, A.J., Winslett, M., Basney, J., Welch, V.: Traust: A Trust Negotiation Based Authorization Service. In Stølen, K. et al., eds.: Proc. 4th International Conference on Trust Management. LNCS 3986, Pisa, Springer-Verlag (2006) 458–462
18. Pearson, S., Mont, M.C.: Provision of Trusted Identity Management Using Trust Credentials. In Stølen, K. et al., eds.: Proc. 4th International Conference on Trust Management. LNCS 3986, Pisa, Springer-Verlag (2006) 267–282

19. Pearson, S.: Trusted Computing: Strengths, Weaknesses and Further Opportunities for Enhancing Privacy. In Herrmann, P. et al., eds.: Proc. 3rd International Conference on Trust Management. LNCS 3477, Paris, Springer-Verlag (2005) 305–320
20. Jensen, C.D., O Connell, P.: Trust-Based Route Selection in Dynamic Source Routing. In Stølen et al., eds.: Proc. 4th International Conference on Trust Management. LNCS 3986, Pisa, Springer-Verlag (2006) 150–163
21. Kerschbaum, F., Haller, J., Karabulut, Y., Robinson, P.: PathTrust: A Trust-Based Reputation Service for Virtual Organization Formation. In Stølen et al., eds.: Proc. 4th International Conference on Trust Management. LNCS 3986, Pisa, Springer-Verlag (2006) 193–205
22. 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
23. Lenzini, G., Tokmakoff, A., Muskens, J.: Managing Trustworthiness in Component-Based Embedded Systems. In: Proc. 2nd International Workshop on Security and Trust Management, Hamburg (2006)
24. Quercia, D., Hailes, S., Capra, L.: B- Trust: Bayesian Trust Framework for Pervasive Computing. In Stølen, K. et al., eds.: Proc. 4th International Conference on Trust Management. LNCS 3986, Pisa, Springer-Verlag (2006) 298–312
25. Kraemer, F.A., Herrmann, P.: Service Specification by Composition of Collaborations — An Example. In: 2006 IEEE/WIC/ACM International Conference on Web Intelligence. IEEE Computer Society Press, Hong Kong (2006)
26. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley Longman (1999)
27. Object Management Group: Unified Modeling Language: Superstructure (2006)
28. Sanders, R.T., Castejón, H.N., Kraemer, F.A., Bræk, R.: Using UML 2.0 Collaborations for Compositional Service Specification. In: ACM / IEEE 8th International Conference on Model Driven Engineering Languages and Systems. (2005)
29. Rossebø, J.E.Y., Bræk, R.: Towards a Framework of Authentication and Authorization Patterns for Ensuring Availability in Service Composition. In: Proc. 1st International Conference on Availability, Reliability and Security (ARES'06), IEEE Computer Society Press (2006) 206–215
30. Castejón, H.N., Bræk, R.: A Collaboration-based Approach to Service Specification and Detection of Implied Scenarios. In: ICSE's 5th Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'06). (2006)
31. Kraemer, F.A., Herrmann, P.: Transforming Collaborative Service Specifications into Efficiently Executable State Machines. Electronic Communications of the EASST (2007) To appear.
32. Kraemer, F.A., Herrmann, P., Bræk, R.: Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In Meersmann, R., Tari, Z., eds.: Proc. 8th International Symposium on Distributed Objects and Applications (DOA). LNCS 4276, Montpellier, Springer-Verlag (2006) 1613–1632
33. Lamport, L.: Specifying Systems. Addison-Wesley (2002)
34. Yu, Y., Manolios, P., Lamport, L.: Model Checking TLA+ Specifications. In Pierre, L., Kropf, T., eds.: Correct Hardware Design and Verification Methods (CHARME '99). LNCS 1703, Springer-Verlag (1999) 54–66
35. Jøsang, A.: The right type of trust for distributed systems. In: Proc. UCLA conference on New security paradigms workshops, Lake Arrowhead, ACM (1996) 119–131

36. 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)
37. Jøsang, A., Knapskog, S.J.: A metric for trusted systems. In: Proc. 21st National Security Conference, NSA (1998)
38. Herrmann, P., Krumm, H.: A Framework for Modeling Transfer Protocols. Computer Networks **34** (2000) 317–337
39. Bræk, R.: Unified System Modelling and Implementation. In: International Switching Symposium, Paris (1979) 1180–1187
40. Bræk, R., Gorman, J., Haugen, Ø., Melby, G., Møller-Pedersen, B., Sanders, R.T.: Quality by Construction Exemplified by TIMe — The Integrated Methodology. Telektronikk **95** (1997) 73–82
41. Haugen, Ø., Møller-Pedersen, B.: JavaFrame — Framework for Java Enabled Modelling. In: Proc. Ericsson Conference on Software Engineering, Stockholm, Ericsson (2000)
42. Herrmann, P.: Temporal Logic-Based Specification and Verification of Trust Models. In Stølen, K. et al., eds.: Proc. 4th International Conference on Trust Management (iTrust 2006). LNCS 3986, Pisa, Springer–Verlag (2006) 105–119