Specification and Verification of Security Properties of e-Contracts

Anas Abou El Kalam¹, Jérôme Ermont² and Yves Deswarte³

¹,² Université de Toulouse, IRIT, INP / ENSEEIHT: 2 rue Camichel Toulouse France
³ LAAS-CNRS; Université de Toulouse; 7 avenue du Colonel Roche, F-31077 Toulouse, France

¹ anas.abouelkalam@enseeiht.fr, ² jerome.ermont@enseeiht.fr, ³ yves.deswarte@laas.fr

Abstract

Converting a conventional contract to an electronic one that can be enforced, queried and verified by computers is a challenging task. The difficulties are mainly caused by the ambiguities that the original human oriented text is likely to contain. In this paper, we present new templates to specify the requirements of e-contracts, to securely check the well-execution of their clauses and to verify some security properties. To achieve this goal, we first analyze the contract of an illustrative e-commerce example. Then, through this example, we derive the most relevant security requirements of e-contracts. In particular, we characterize e-contract security rules / clauses by defining obligations, prohibitions, permissions, temporal constraints, responsibilities and disputes. We demonstrate how this kind of security requirements can be described using a timed automata formalism. Moreover, we show how verifying methods, such as model-checking, can be applied to this kind of models to check some security properties.

Keywords: e-contract security, timed automata and model-checking.

I. Introduction

In the century of globalization, emerging applications such as e-Business, e-Commerce, e-Science, e-Health, e-Government, are becoming more and more complex, virtualized and internationalized. However, in order to ensure legality and to protect interests of the involved parties, electronic interactions should be regulated by contracts.

This requirement implies that not only the original natural language contract has to undergo a conversion process from its original format into an electronic one, but also it must be possible to check if the interactions between the parties are compliant with the contract, and to detect, audit and notify violations when they occur.

To realize this conversion process, our approach is the following:

1. We identify the security requirements generally found in contracts; e.g., rights, obligations, sanctions, temporal constraints, etc.
2. We suggest a template to model each of these requirements; e.g., how to model an obligation, a forbidden action, etc. In this way, by instantiating this template according to the studied application, it is easy to capture and model all the security requirements of a contract.
3. We propose mechanisms to query the system (e.g., to know under which conditions the contract will be ended), to reason on the contract and to verify the security properties that interactions should satisfy.

Actually, several models have already been proposed for this purpose, but the challenge that still arises is to find a balance between a rich model/logic and its capacity to validate/verify the security properties. The main objective of this paper is to propose a model that is rich enough to cover the variety of e-contract clauses on the one hand, and that provides powerful verification mechanisms on the other hand.

To achieve these goals, we present in Section II an example of e-commerce as well as the contract that governs it. This use case is supposed to be simple but as representative as possible to cover the needs generally found in e-contracts. Then, in Section III, we use this use case to derive the main security needs and requirements of e-contracts. Afterwards, in Section IV, we use timed automata and Computational Tree Logic (CTL) to capture and model the requirements identified before. In particular, for every kind of requirement, we present a template that specifies it. Then, in Section V we show how our formalism can be used to query and to reason on the e-contract security protocol. After that, Section VI applies our model to our case study. Section VII discusses related works and finally, Section VIII presents our conclusions and perspectives.
II. A use case

In this section we will discuss an example of a simplified but representative business contract that stipulates the interactions between a buyer and a seller for the purchase of goods. The contract contains clauses of the following form:

- At his discretion, the buyer may send a purchase order to the seller.
- The seller is obliged to confirm acceptance/rejection of the purchase order within 24 hrs.
- The seller is obliged to send an invoice to the buyer within 7 days of accepting the purchase order.
- The buyer and the seller are forbidden to send invalid messages.
- Failures to honor obligations and prohibitions will result in financial compensations equal to 20% of the value of the item.
- If one of the contractual parties detects a technical failure that prevents them from continuing the normal course of a transaction, this party is obliged to send a failure notification message to the other party as well as to the administrator/third party.
- In case of failure, the e-contract is ended and the involved parties are informed.

III. General security requirements

Based on the scenario described above, this section progressively derives the main concepts used to specify an e-contract security policy. Actually, as contracts use large subsets of natural language, their expressiveness is very rich. But globally, the e-contract security policy should be expressed in term of the following entities:

- **Actors**: who are the actors (organizations, roles) involved in the contract? Who can carry out actions according to the contract clauses?
- **Actions / workflows**: as the contract should identify the activities (tasks/e-services to be executed during the process), the e-contract security policy should handle the concepts of actions and workflows (sequential, cyclic, ordered, …).
- **Deontic modalities**: an e-contract can be seen as a legally enforceable agreement in which two or more parties commit to certain obligations. Consequently, classical deontic concepts, such as obligations, permissions, prohibitions are important in modeling the e-contract security policy.
- **Temporal modalities**: temporal constraints are rules that regulate the order, timing and duration of actions. It is thus obvious that e-contracts should identify the duration (e.g., number of days / hours) of certain actions, the synchronization requirements as well as when (at, before, after, during …) an action can/must/cannot be carried out.
- **Context**: in order to provide fine-grained access control, it is necessary to take the context into account, e.g., the requester/resource location, the separation of duty, delegations, exceptions.
- **Heterogeneity**: e-contracts may have complex structures based on bilateral (e.g., buyer-seller), or multiparty (e.g., house building contract) interactions. Moreover, the e-contract dimension can be composite (e.g., textile value chain contracts). These requirements naturally imply heterogeneity (e.g., different views, structures, and implementation).
- **Sanctions**: as e-contracts contain deontic modalities (e.g., obligations, prohibitions), it is necessary to handle the cases where these modalities are not respected. In particular, a failure to execute an obligation and an attempt to execute a prohibition are considered as contract violations and the offending actor may be subject to sanctions.
- **Auditing**: when implementing an e-contract tool, it is necessary to keep an audit log that can be analyzed at runtime and/or be used later as evidence in case of disputes.

IV. A security model for e-contracts

The previous section presented the most relevant entities and requirements (mainly related to temporal constraints, actions / workflows, deontic modalities and sanctions) that should be handled by an e-contract security policy. The challenge now is to find a convenient framework that captures all these aspects. Actually, we believe that most of these requirements (except deontic modalities) can be specified by timed automata. Our choice is also motivated by the possibility of checking the correctness of the automata behavior and by the availability of several tools dedicated to this issue.

In this context, our methodology consists in: (1) showing how timed automata can capture the e-contract security requirements; (2) trying to homogeneously extend timed automata to capture
the deontic modalities. But before explaining these steps, let us first present the most relevant notions (regarding our study) of timed automata.

V. Timed automata

Timed automata have been proposed by Alur and Dill to describe systems behavior with time [1]. Basically, a timed automaton is a finite automaton with a set of clocks, i.e. real and positive variables increasing uniformly with time. Transition labels are: a guard, i.e. a condition on clock values, actions and updates, which assign new values to clocks.

Composition of timed automata is obtained by synchronous product: each action $a$ executed by a timed automaton corresponds to an action with the same name $a$ executed in parallel by a second timed automaton. In other words, a transition that executes the action $a$ can only be triggered in one automaton if the transition labeled $a$ can also be triggered in the other automaton. The two transitions are performed simultaneously and communications use the rendez-vous mechanism.

Note that performing transitions is instantaneous; conversely, time can be consumed in nodes.

Besides, each node is labeled by an invariant, that is a Boolean condition on clocks. Node occupation is invariant-dependent. The node is occupied if the invariant is true.

Finally, it is important to note that a system modeled with timed automata can be verified using model-checking. In particular, a reachability analysis can be performed by model-checking. It consists in encoding a certain property in terms of reachability of a given node (of one of the automata). In this respect, the property is verified by a node reachability if and only if the node is reachable from an initial configuration.

In the following subsections, we model the different requirements. The properties will be verified using the UPPAAL model-checker [2,3].

V.1. Modeling permissions

Permissions mean actions that are allowed by the contract' clauses. In our timed automata model, permitted actions are actually specified by transitions. For instance, in Figure 1, the system can execute the action $a$ at any time and then, behaves like the automaton $A$.

V.2. Modeling prohibitions.

We distinguish two kinds of prohibitions:

- Implicit prohibitions: the idea is to only specify permissions, which means that prohibited actions (i.e., actions that are not in accordance with the contract clauses) will not be specified in the automata. The states, actions and transitions not represented in the automata are by essence not possible because the system will not recognize them. This policy is actually similar to “by default-policies” used in some firewall configurations (i.e., in the context of such firewalls, actions that are not explicitly specified as permitted are actually prohibited).

- Explicit prohibitions: explicit prohibitions can be particularly useful in the management of decentralized policies / contracts where each administrator does not have details about the other parts of the system. Moreover, explicit prohibitions can also specify exceptions or limit the propagation of permissions in case of hierarchies. In our model, we specify explicit prohibitions by adding a “failure state” where the user will be (automatically) leaded if he/she misuses the system (Figure 2).

In Figure 2, as the $a$ action is forbidden, its execution automatically leads to the failure state described by an unhappy face.
V.3. Modeling obligations

Obligations are actions that « must » be carried out; otherwise the concerned entity will be subject to sanctions. In particular, obligations can be useful to impose some internal or external, manual or automatic actions.

In this work, we distinguish two kinds of obligations: *internal* obligations and *external* obligations.

- An *internal obligation* is a set of mandatory actions that must be performed by local entities (possibly constrained by a maximum delay). An obligation is automatically triggered by an event such as a change in the context or a particular message exchanged between the contractual entities (e.g., the buyer and the seller).

- An *external obligation* is a set of mandatory actions that must be performed by remote entities, but checked by local entities.

Note that the difference between these two kind of obligations is related to the target of the action (i.e., the entity subject to the obligation), not to the way the action is carried out.

This distinction is important in the context of e-contracts as each contractual party has its own automaton, which checks / verifies if the functioning of the other party respects the contract clauses. In fact, in each automaton, it would be important to distinguish between actions that must be done internally and those that must be performed by the other party.

After defining obligations, let us now explain how we model these notions with timed automata. First, as every obligation is also a permission (Obligation $\rightarrow$ Permission), obligations will be specified by particular transitions (in the same way as permissions); however, as obligations are stronger than permissions, we should add other symbols to capture this semantic and to distinguish between what is mandatory and what is permitted but not mandatory.

Actually, to model obligations, we use transition time-outs and invariants.

In this respect, an internal obligation is considered as a simple transition, and if a maximum delay is assigned to the obligation, a time-out (noted by $d$ in Figure 3) is set for the delay. When the obligation is fulfilled by local entities, this event resets the time-out and the system behaves like $A_1$. On the contrary, if the time-out expires, an exception is raised, which is another internal obligation, and the system behaves like $A_2$ (which can be considered as an exception).

![Figure 3: Modeling internal obligations.](image)

Besides, external obligations are represented by a transition with a time-out (an alternative temporized transition). When the remote entity has fulfilled its obligation, it sends a message carrying a proof of obligation completion, which resets the time-out. Alternatively, if the time-out expires, an internal obligation is triggered, corresponding to an exception processing that must be performed by local entities.

In Figure 4, the two automata are running in parallel and should be synchronized on the $a$ action. Actually, $a$ is an external obligation that must be carried out by the right hand automaton of Figure 4. Besides, the left hand automaton should receive a signal proving the well-execution of $a$ by the other automaton on the expected delay ($d$). Otherwise the external obligation is not fulfilled (by the right hand party).

![Figure 4: Modeling external obligations.](image)

V.4. Representing disputes in our model

Naturally, a conflicting situation may arise if one of the contract parties does not comply with the contract clauses (e.g., fails to fulfill one of its obligations, or performs a prohibited action). Of course, these dispute situations should be described in the e-contract in order to rigorously state how to
solve them and which sanctions to apply. Moreover, each contractual party should maintain an audit log in order to be able to present evidences to the judge in case of dispute. According to these evidences and to the contract clauses, the judge will decide and enforce the corresponding suitable sanctions.

To model these notions, we use two notions: a dispute state (noted by unhappy face state) and variables. Basically, when the conflicting situation is detected by one of the contract’ parties, the automata automatically makes a transition to a dispute situation (i.e., to the unhappy state). Contrarily to prohibitions, disputes are stronger (more sensitive) and automatically lead to the end of the contract.

Moreover, as disputes have different severities / sensitivities and as they are not all subject to the same sanctions, we use variables (i.e., labels on the unhappy state) to distinguish the different kinds of disputes as well as the corresponding sanctions.

**VI. The verification process**

Once we have defined the timed automata of our contract, it would be interesting to check its well-execution.

Actually, the idea is to verify that the automata will never reach the dispute state. In timed automata, the reachability analysis can be carried out by model-checking. Basically, we should first specify the reachability property (e.g., reaching the unhappy state); then, we can verify if from the initial configuration, there exists a possible execution of the system that leads to a node where this property is verified (i.e., reaching the unhappy state). In this respect, the contract is respected if none of the possible executions of the system will lead to the unhappy state (specified through the reachability property).

Note that the reachability problem is decidable and that the reachability properties are generally simple to verify by model checkers [4, 5].

**VII. Example of contract modeled by timed automata-based model**

After defining our templates as well as our verification process, let us apply our model to the use case presented in Section II.

In the timed automaton of Figure 6, the first transition specifies that the buyer may (has the permission to) send a purchase order to the seller. When he does it, he reaches a state where he is waiting for the accept signal from the buyer. Actually this signal corresponds to an external obligation that must be fulfilled by the seller and checked by the buyer automaton. Consequently, we represent it by a transition with a time-out initialized to 1 (1 day = 24 hours) as the seller is obliged to confirm the acceptance / rejection within 1 day. When the remote entity (seller) has fulfilled its obligation (sending the acceptance within 1 day), it sends a message carrying a proof of obligation completion, which resets the time-out. Alternatively, if the time-out expires, an exception is triggered, leading to the dispute state.
Similarly, an external obligation for the remote entity (i.e., seller) to send an invoice within 7 days (after a confirmed order) can be represented by a time-out (set for a 7 day duration). If the message corresponding to the invoice is received, the time-out is reset. On the contrary, if the time-out expires, it triggers an exception (dispute state: the e-contract is canceled and the buyer informs a judge).

Beside that, in the timed automaton of Figure 7, the seller receives the purchase order from the buyer. Actually, it is the matter of a synchronization action (see also Figure 6). After receiving this signal it reaches a state with a “true” invariant. This means that it can reach and stay in this state without condition. From this state, three transitions are possible. If he/she accepts it within 1 day, or if he/she rejects it within 1 day, he/she fulfills its obligation. Else, he/she will receive a signal (non conformance to the obligation) from the buyer, leading the system to the dispute state.

![Figure 7: The seller e-contract model.](image)

The same reasoning goes for the rest of the automaton (the obligation to send an invoice within 7 days).

Now, once we have modeled the contractual parties' behavior, we can verify if the system can reach a dispute state. In fact, proving that all the possible executions of the system will never lead to a conflicting situation is equivalent to prove that the exchange protocol can be run according to the contract clauses.

In our implementation, the automata are modeled by the UPPAAL model checker [2, 3]. The reachability properties are modeled by a subset of the Computational Tree Logic (CTL) [4]. For example, the following property "E<>Buyer.Dispute" stands for it exists at least one execution where the buyer reaches the dispute state.

This property is true and thanks to the model checker (UPPAAL in our case), we have obtained two possible executions:
- the seller does not respect the acceptance delay (1 day);
- the seller does not respect the invoice delay (7 days);

Inversely, the following property means that none of the possible executions will lead the buyer to a dispute state.

A[\] not Buyer.Dispute

VIII. Related works

Several works have been devoted to logics and theories for e-contracts. The most relevant ones are based on predicate logic, first order logic and speech act theory, deontic logic, model action logic, temporal logic, Petri nets and event calculus.

In the SeCo project, Gisler et al. have presented secure e-contracts based on three levels: logic, information and communication levels [6].

Jimenez et al. have defined a method for specifying contract mediated interactions [7]. Their model is based on deontic modalities. The general form of a contract is: nsi: δ → θs,b(α<ψ), where ns_i ( i >= 1 ) is a label that identifies the i\textsuperscript{th} normative statement of the contract; δ stands for a condition that might eventually become true; θ stands for permission, obligation or prohibition; s and b are the subject and beneficiary of θ, respectively; α is an action to be performed by s for the benefit of b; ψ is a deadline.

O.Perrin and . Godard presented in [8] an approach (similar to [7]) where permissions, obligations and prohibitions are mapped into ECA (even-condition-actions). An executable contract becomes a set of ECA rules deployed within a trusted third party and placed between two business partners to drive their interactions.

Kumar et al. [9] proposed an interesting work that couple RBAC (Role-Based Access Control) with TBAC (Task-BAC) and enforces sequential and temporal constraints over them so that process participants get only “Need to know information” with less administrative overhead.

Another interesting work is described by Marjanovic and Milosevic in [10]. They present a specification of deontic constraints and verification of deontic consistency associated with roles in a
contract. Their model defines also temporal constraints. In their model, a duration constraint has the Duration(ai, <=, d, h) form: action ai must be completed in no more than d time. Concerning the deontic modalities, they give examples such as O(Ri, ai, e, <=, Date, t1, t2): role Ri is obliged to finish action ai no later than Date. This obligation is valid from time t1 to t2.

However, up to our knowledge, most of these works have combined several logics to gain in the expressiveness, while these logics / models are not necessarily homogeneous. As a result, the verification mechanisms are either rarely presented or often too complex.

Moreover, several other works based on deontic logic are not expressive enough to describe situations where neither actions are assigned to specific agents ("it ought to be that the payment is sent by Alice"), nor permissions, obligations and prohibitions that become and cease to be in effect depending on the occurrence of time and other events (deontic logic is actually static).

To overcome these limits, some works mix deontic logic with constructs from Modal Logic, Temporal Logic, Logic of Action or from their combinations. Such hybrid logic systems can certainly express complex situations; however such systems have not yet been thoroughly studied and understood. Moreover, the logical rigor of a contract expressed in such notations is questionable; and generally, the resulting language is not really able to automatically verify the correctness of his notation by proof-theoretical means or model checking.

In our work, we based our model on well-known mechanisms ans tools: timed automata, CTL and UPPAAL. To be as rich as possible while remaining verification-homogeneous, we gave a precise semantic to specify and distinguish between the different deontic modalities (permissions, obligations, prohibitions), the dispute situations and sanctions.

IX. Conclusions and perspectives

In this paper we have presented a new model for specifying e-contract security requirements. Our model is able to specify deontic modalities, actions and events, temporal constraints and disputes. We have also presented how our formalism can be used to query the policy and to check its consistency as well as some of the contract properties (e.g., the reachability property).

Note that even if we have experimented our model only in the context of an e-commerce protocol, our results could perfectly be applied to any kind of e-contracts.

Moreover, even if the main weakness of the model checking is the state explosion problem, the complexity of automata system for e-contracts is generally smaller than in other contexts where model checking has been applied successfully [11].

Now, we are looking for enhancing our model to take into account other interesting requirements such as delegations. Furthermore, liability as well as privacy issues are also serious challenges that need to be addressed in this field.

References