Conformance Verification of Normative Specifications using C-O Diagrams

Gregorio Díaz[1], Luis Llana[2], Valentín Valero[1] and Jose A. Mateo[1]
1. Computer Science Dept. University of Castilla-La Mancha
[gregorio.diaz,valentin.valero,joseantonio.mateo]@uclm.es
2. Computer Science Dept. Complutensis University of Madrid
llana@fdi.ucm.es

C-O Diagrams have been introduced as a means to have a visual representation of normative texts and electronic contracts, where it is possible to represent the obligations, permissions and prohibitions of the different signatories, as well as what are the penalties in case of not fulfillment of their obligations and prohibitions. In such diagrams we are also able to represent absolute and relative timing constrains. In this paper we consider a formal semantics for C-O Diagrams based on a network of timed automata and we present several relations to check the consistency of a contract in terms of realizability, to analyze whether an implementation satisfies the requirements defined on its contract, and to compare several implementations using the executed permissions as criteria.

1 Introduction

In the software context, the term *contract* has traditionally been used as a metaphor to represent limited kinds of "agreements" between software elements at different levels of abstraction. The first use of the term in connection with software programming and design was done by Meyer in the context of the language Eiffel (*programming-by-contracts*, or *design-by-contract*). This notion of contracts basically relies on the Hoare notion of pre and post-conditions and invariants. Though this paradigm has proved to be useful for developing object oriented systems, it seems to have shortcomings for novel development paradigms such as service-oriented computing and component-based development. These new applications have a more involved interaction and therefore require a more sophisticated notion of contracts. As a response, behavioural interfaces have been proposed to capture richer properties than simple pre and post-conditions [5]. Here it is possible to express contracts on the history of events, including causality properties. In the context of SOA, there are different service contract specification languages, like ebXML, WSLA, and WS-Agreement. These standards and specification languages suffer from one or more of the following problems: They are restricted to bilateral contracts, lack of formal semantics (so it is difficult to reason about them), their treatment of functional behaviour is rather limited and the sub-languages used to specify, for instance, security constraints are usually limited to small application-specific domains. The lack of suitable languages for contracts in the context of SOA is a clear conclusion of the survey [13] where a taxonomy is presented.

In [10] *C-O Diagrams* were introduced, a graphical representation not only for electronic contracts but also for the specification of any kind of normative text (Web service composition behaviour, software product lines engineering, requirements engineering, ...). *C-O Diagrams* allow the representation of complex clauses describing the obligations, permissions, and prohibitions of different signatories (as defined in deontic logic [12]), as well as *reparations* describing contractual clauses in case of not fulfillment of obligations and prohibitions. Besides, *C-O Diagrams* permit to define real-time constraints. In [9] some of the satisfaction rules needed to check if a timed automaton satisfies a *C-O Diagram* specification

G. J. Pace and A. P. Ravn:
Proceedings of Sixth Workshop on Formal Languages
and Analysis of Contract-Oriented Software (FLACOS'12)
EPTCS 94, 2012, pp. 1–10, doi:10.4204/EPTCS.94.1

were defined. In [11], *C-O Diagrams* are equipped with a formal semantics based on a transformation of these diagrams into a network of timed automata (NTA). The contribution of this work pursues the further development of our previous work. This time we will focus on the development of different relations to check the consistency of contracts, to seek whether an implementation conforms a given contract and to compare several implementations. To achieve this goal, we consider a semantics in terms of NTAs and we establish relations with the implementations also written in terms of NTAs.

2 Related Work

The use of deontic logic for reasoning about contracts is widely spread in the literature since it was proposed in [3] for modelling communication processes. In [8] Marjanovic and Milosevic present their initial ideas for formal modelling of e-contracts based on deontic constraints and verification of deontic consistency, including temporal constraints. In [4] Governatori et al. go a step further providing a mechanism to check whether business processes are compliant with business contracts. They introduce the logic FCL to reason about the contracts, based again on deontic logic. In [7] Lomuscio et al. provides another methodology to check whether service compositions are compliant with e-contracts, using WS-BPEL to specify both, all the possible behaviours of each service and the contractually correct behaviours, translating these specifications into automata supported by the MCMAS model checker to verify the behaviours automatically.

None of the previous works provides a visual model for the definition of contracts. However, there are several works that define a meta-model for the specification of e-contracts which purpose is their enactment or enforcement. In [2] Chiu et al. present a meta-model for e-contract templates written in UML, where a template consists of a set of contract clauses of three different types: obligations, permissions and prohibitions. These clauses are later mapped into ECA rules for contract enforcement purposes, but the templates do not include any kind of reparation or recovery associated to the clauses. In [6] Krishna et al. another meta-model based on entity-relationship diagrams is proposed to generate workflows supporting e-contract enactment. This meta-model includes clauses, activities, parties and the possibility of specifying exceptional behaviour, but this approach is not based on deontic logic and says nothing about including real-time aspects natively.

3 C-O Diagrams Syntax and Semantics

We first introduce a motivation example to understand the diagrams in an easy way. Figure 1 consists of three sub-figures, a) depicting a basic structure of a clause, and, b) and c) depicting our running example. This example consists in the payment and shipment of an item previously sold during an online auction. Thus the action starts after the auction has finished, that is, if the bid placed by the *buyer* is the highest one, then the activities concerning the payment and the shipment of the item start. First, the *buyer* has **three days** to perform the payment, which can be done by means of credit card or PayPal. After the payment has been performed, the *seller* has **fourteen days** to send the item to the *buyer*. If the item is not received within this period of time, the *auction service* has **seven days** to refund the payment to the *buyer* and can penalize the *seller* in some way.

At first sight, the figures are top down hierarchical structures with several boxes and branches. In Figure 1, we can observe several examples. At the top-left hand side of this figure, Figure 1-a, we can observe the basic construction element called **box**, also referred as proposition or clause. It is divided into four fields. The *guard* **g** specifies the conditions under which the contract clause must be taken into account (boolean expression). The *time restriction* **tr** specifies the time frame during which the contract

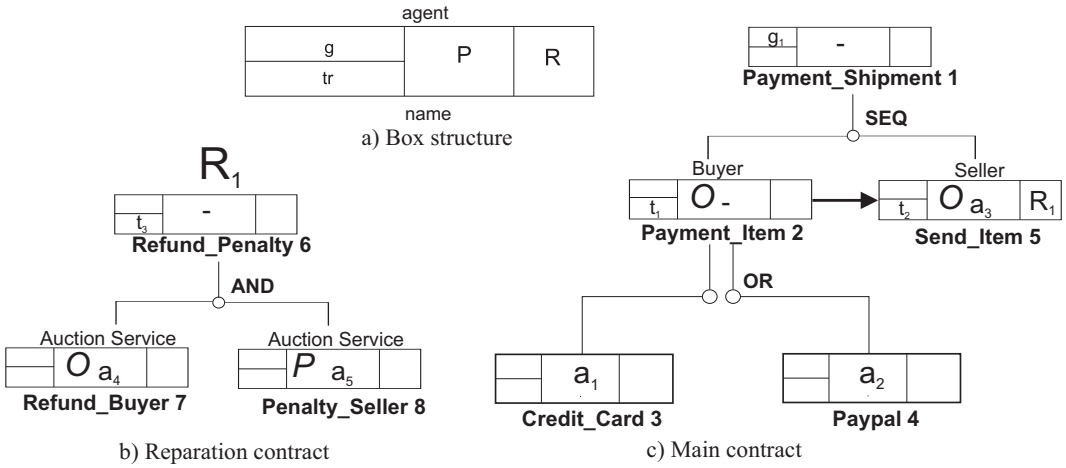

Figure 1: *C-O Diagrams* examples

clause must be satisfied (deadlines, timeouts, etc.). The *propositional content* **P**, on the centre, is the main field of the box, and it is used to specify normative aspects such as obligations (**O**), permissions (**P**) and prohibitions (**F**), that are applied over actions, and/or the specification of the actions themselves. The last field of these boxes, on the right-hand side, is the *reparation* **R**. This reparation, if specified by the contract clause, is a reference to another contract that must be satisfied in case the main norm is not satisfied (a *prohibition* is violated or an *obligation* is not fulfilled, there is no reparation for *permission*), considering the clause eventually satisfied if this reparation is satisfied. Each box has also a name at the bottom part and an agent at the top part.

These are the basic boxes, which can be composed by using some refinements. Refinements are classified into three types: joining *AND-refinements*, disjunctive *OR-refinements* and sequential *SEQ-refinement*. Joining refinements require that all the hanging propositions should be accomplished to declare the upper proposition accomplished; on the contrary, disjunctive propositions only require one to be accomplished; whereas, sequential propositions require a left-to-right ordered sequential satisfaction of every proposition to obtain the same result. In Figure 1-c, the root box, which only shows the name and guard g_1 (this guard checks if the buyer is the auction winner) is decomposed into two sub-clauses via sequential composition, that is, first the one on the left hand side, *Payment_Item* and, afterwards, the one on the right hand side, *Send_Item*. The first one is the **obligation** (*O*) of payment with the temporal restriction t_1, three days in this case, then this obligation is decomposed via an *OR-refinement* into **Clause 3** and **Clause 4** composing the actions of paying by credit card or PayPal by means of an *OR-refinement*. On the right-hand side we have the **obligation** (*O*) specified in **Clause 5**, which has been called *Send_Item*, including the real-time constraint t_2 14 days and a reference to reparation R_1.

Since reparations are references to new contracts, in Figure 1-b we can see the diagram corresponding to reparation R_1. It has been called *Refund_Penalty*, including the real-time constraint t_3, and it is decomposed into two subclauses by means of an *AND-refinement*. The subclause on the left corresponds to the **obligation** (*O*) specified in **Clause 7**, which has been called *Refund_Buyer*, and the subclause on the right corresponds to the **permission** (*P*) specified in **Clause 8**, which has been called *Penalty_Seller* regarding the possibility of performing some kind of penalization over the seller by the *Auction Service*.

The **syntax** of *C-O Diagrams* was first presented in [10]. Next, we just present a brief description of the EBNF grammar followed in the diagrams:

$$
\begin{array}{rcl}
C & := & (agent, name, g, tr, O(C_2), R) \mid \\
 & & (agent, name, g, tr, P(C_2), \varepsilon) \mid \\
 & & (agent, name, g, tr, F(C_2), R) \mid \\
 & & (\varepsilon, name, g, tr, C_1, \varepsilon)
\end{array}
\qquad
\begin{array}{rcl}
C_1 & := & C\,(And\ C)^+ \mid C\,(Or\ C)^+ \mid C\,(Seq\ C)^+ \\
C_2 & := & a \mid C_3\,(And\ C_3)^+ \mid C_3\,(Or\ C_3)^+ \mid C_3\,(Seq\ C_3)^+ \\
C_3 & := & (\varepsilon, name, \varepsilon, \varepsilon, C_2, \varepsilon) \\
R & := & C \mid \varepsilon
\end{array}
$$

The C-O diagram **semantics** is defined by using NTAs (*Network of Timed Automata*) [1] as semantic objects. Here we omit this formal translation and the technical definitions can be found in [11]. Instead, we present an informal interpretation of the NTA behaviors. When transforming a C-O diagram into a network of timed automata, the nodes of the generated automata are decorated with the set of contractual obligations, prohibitions and permissions that are either violated or satisfied.

Definition 1 *(Violation, Satisfaction and Permissions Sets) Let us consider the set of contractual obligations and prohibitions CN ranged over cn, cn′,... standing for identifiers of obligations and prohibitions and the set of contractual permissions CP ranged over cp, cp′, ...*

Definition 2 *(Decorated timed automaton)*
A decorated timed automaton is a timed automaton (N, n_0, E, I) *(see [1]) where for each* $n \in N$ *we have defined the following sets* $V(n) \subseteq CN$ *(the set of the obligations* violated *in n),* $S(n) \subseteq CN$ *(the set of the obligations* satisfied *in n), and* $P(n) \subseteq CP$ *(the set of permissions granted in n).*

Graphically, when we draw a timed automaton extended with these three sets, we write under each node n (between braces) its violation set $V(n)$ on the left, its satisfaction set $S(n)$ on the centre and its permission set $P(n)$ on the right. These sets are initially empty, and they do not change except in two cases, a) when either a obligation or a prohibition is violated or satisfied, b) when a permission is performed.

Let us recall that the intuitive meaning of an NTA is the parallel composition of several timed automata. We consider a set of actions ACT, in which we have the following actions:

- An internal action $\tau \in ACT$.
- An input action $m? \in ACT$.
- An output action $m! \in ACT$.
- A synchronization action $m \in ACT$ that comes from a synchronization of an input action $m?$ and an output action $m!$.

The semantics of timed automata is well known [1]. It is based on a timed labelled system, where states are pairs $s = (n, v)$ where n is a node of the automaton and v is a valuation of the clocks. There are two types of transition:

- timed transitions[1] $s \xrightarrow{d} s' (d \in \mathbb{R}^+)$
- and action transitions $s \xrightarrow{a} s' (a \in ACT)$.

A *Network of Timed Automata* (NTA) is then defined as a set of timed automata that run simultaneously, using the same set of clocks, and synchronizing on the common actions. Internal actions can be executed by the corresponding automata independently, and they will be ranged over the letters $a, b, \ldots$ whereas synchronization actions must be executed simultaneously by two automata. Synchronization actions are ranged over letters $m, m', \ldots$ and they come from the synchronization of two actions $m!$ and $m?$, executed from two different automata[2].

The operational semantics of a network of timed automata has the following transitions:

- A delay transition of d time units requires that all the involved automata are able to perform this delay individually.
- Autonomous action transitions that correspond to the evolution of a single timed automaton.
- Synchronization transitions that require two automata to perform two complementary actions, $m!$ and $m?$, respectively.

[1] Timed transitions only change the valuation of clocks.

[2] In the original definition the only internal action is τ, and synchronizations always yield internal actions.

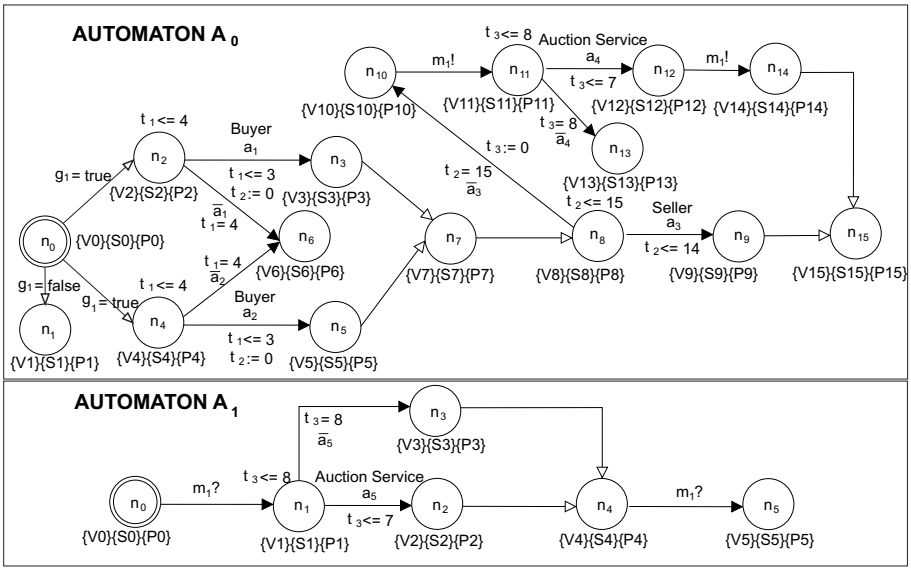

Figure 2: Automata for the Payment_Shipment example, A_0 and A_1

Definition 3 *(Semantics of an NTA)*
Let $N = (A_1, , \ldots, A_k)$ be an NTA. A state *of N is a tuple $\overline{s} = (s_1, \ldots, s_k)$, where s_i is a state of the automaton A_i (for $i = 1, \ldots, k$). We have the following transitions:*

- *Timed transitions. If* $\forall 1 \leq j \leq k : s_j \xrightarrow{d} s'_j$, *then:* $(s_1, \ldots, s_k) \xrightarrow{d} (s'_1, \ldots, s'_k)$ *with* $d \in \mathbb{R}^+$.
- *Autonomous transitions. If* $\exists 1 \leq j \leq k : s_j \xrightarrow{a} s'_j$ *for* $a \in ACT$, $a \neq m!$ *and* $a \neq m?$, *then:* $(s_1, \ldots, s_j, \ldots s_k) \xrightarrow{a} (s_1, \ldots, s'_j, \ldots s_k)$.
- *Synchronization transitions.* $\exists 1 \leq i, j \leq k : s_j \xrightarrow{m?} s'_j,\ s_i \xrightarrow{m!} s'_i$ *for* $m?, m! \in ACT$, *then:* $(\ldots, s_j, \ldots, s_i, \ldots,) \xrightarrow{m} (\ldots, s'_j, \ldots, s'_i, \ldots)$, *assuming that* $j \leq i$, *the other case is similar.*

The complete semantics for *C-O Diagrams* in terms of NTAs translation can be found in [11]. Figure 2 shows the resulting NTA once these transformations are applied over the *Payment_Shipment* example. This NTA consists of two automata running in parallel, that is, $NTA_{P\&S} = \{A_0, A_1\}$. Automaton A_0 is where the main part is translated and the starting point of this example. The main translated structures we can observe here are the three kind of refinements and the reparation of a violated clause. Besides these main structures, we can see how guards and time restrictions are translated.

In A_0, this contract starts with a SEQ-refinement of two clauses 2 and 5, which assemble in sequence via the transition between nodes n_7 and n_8, that is, the end of clause 2 and the beginning of clause 5, respectively. From node n_0, where clause 2 starts, we may reach either n_2 or n_4, which correspond to an OR-refinement representing the payment made either by credit card or paypal. Node n_6 only captures termination in the event that that the time for the payment expires without performing any of these actions. Notice that once the payment has been done (nodes n_3 or n_5) we move into node n_7, from which the "sending item action" clause 5 starts, which corresponds to action a_3. In this case we have 14 time units. If this time expires and the client has not received the item the reparation clause is activated (node n_{10}). In this case we have an AND-refinement, so a second timed automaton (A_1) is created, which corresponds to the right-hand side part of the AND-refinement (the left-hand side is performed by A_0). Both automata synchronize at their beginning and at their termination in order to be executed simultaneously. The obligation to refund the money is captured by action a_4 in A_0, whereas the

permission to penalize the seller is captured by action a_5 in A_1. Over-line actions label those transitions enabled when the main action is not performed.

Guards are here translated as guards in the transitions and time restrictions are used to denote the invariants of certain states and some guards in transitions, which determine whether a clause is satisfied in time. In reference to the different violation, satisfaction and permission sets, we will comment the most significant ones, which correspond with the maximal paths except when a reparation is defined. Violation sets $V6$, $V10$ and $V13$ consist of the violated clauses: either clause 3 or 4, clause 5 and clause 7, respectively. The satisfaction set $S15$ will consist of clauses 3 or 4 (depending on the payment is made either by credit card or paypal), and either clause 5 (if the item has been sent on time) or clause 7 (if the clause 5 has been repaired). Finally, permission set $P15$ is either empty or clause 8 (if we have followed the reparation and the permission to penalize the seller has been performed).

4 Conformance relations

In this section we define a set of conformance relations to establish whether an implementation of a contract conforms to the contract we want to satisfy. We will consider a semantic relation inspired in the *conformance testing* relation given in [14]. We take as starting point a normative document written in terms of a C-O Diagram, which is then translated into a network of timed automata. We also consider an implementation I of this contract which is also provided as an NTA, with at least the same actions we had in the contract. We intend to define a black box conformance relation, which means that we do not know how the implementation has been done, so we can only use the information about the actions it performs.

Definition 4 *A* timed trace *is a sequence* $[a_1 d_1 a_2 d_2 \cdots a_n d_n] \in (ACT \times \mathbb{R}^+)^*$. *We will use the symbols* t, t_1, t_2, t_n,... *to denote traces. The empty trace is denoted by* $[]$. *The concatenation of* t_1 *and* t_2 *will be denoted by* $t_1 \cdot t_2$. *We will say that* t_1 *is a subtrace of* t_2, *written* $t_1 \leq t_2$, *if there is a trace* t *such that* $t_2 = t_1 \cdot t$.

Let N be an NTA, where we define the timed computations of *N as follows:*

- $\overline{s} \xrightarrow{[]} \overline{s}$.
- $\overline{s} \xrightarrow{t\cdot[ad]} \overline{s'}$ *for* $a \in ACT$ *and* $d \in \mathbb{R}^+$ *if there exist states* $\overline{s_1}, \overline{s'_1}, \ldots, \overline{s_l}, \overline{s'_l}$ *of N with* $l \geq 1$ *such that* $\overline{s} \stackrel{t}{\Rightarrow} \overline{s_1} \xrightarrow{d_1} \overline{s'_1} \xrightarrow{\tau} \overline{s_2} \xrightarrow{d_2} \overline{s'_2} \cdots \overline{s_{l-1}} \xrightarrow{d_{l-1}} \overline{s'_{l-1}} \xrightarrow{\tau} \overline{s_l} \xrightarrow{d_l} \overline{s'_l} \xrightarrow{a} \overline{s'}$ *and* $d = \sum_{1 \leq i \leq l} d_i$

We define the set of timed traces *of N as* $\mathsf{tr}(N) = \{t \mid \exists \overline{s} : \ \overline{s_0} \stackrel{t}{\Rightarrow} \overline{s}\}$, *being* $\overline{s_0}$ *the initial state of* N.

The following definition extends the sets V, S and P to traces, by accumulating the contents of the respective sets V, S, P over the traversed nodes until reaching the final node of the trace.

Definition 5 *Let* $N = (A_1, \ldots A_k)$ *be an NTA and* $t \in \mathsf{tr}(N)$, *we define the sets of violation (denoted* $\mathsf{V}(N,t)$*), satisfaction (denoted* $\mathsf{S}(N,t)$*), and permission (denoted* $\mathsf{P}(N,t)$*) as follows:*

- $\mathsf{V}(N,t) = \{\bigcup_{1 \leq i \leq k} V(n_i) \mid \overline{s_0} \stackrel{t}{\Rightarrow} (s'_1, \ldots, s'_k),\ s'_i = (n_i, v_i)\}$
- $\mathsf{S}(N,t) = \{\bigcup_{1 \leq i \leq k} S(n_i) \mid \overline{s_0} \stackrel{t}{\Rightarrow} (s'_1, \ldots, s'_k),\ s'_i = (n_i, v_i)\}$
- $\mathsf{P}(N,t) = \{\bigcup_{1 \leq i \leq k} P(n_i) \mid \overline{s_0} \stackrel{t}{\Rightarrow} (s'_1, \ldots, s'_k),\ s'_i = (n_i, v_i)\}$

Where $\overline{s_0}$ *is the initial state of N. We say that t is a* good *trace, denoted by* $t \in \mathsf{good}(N)$ *if it is maximal*[3], $\forall S \in \mathsf{S}(N,t) : \ S \neq \varnothing$, *and* $\forall V \in \mathsf{V}(N,t) : \ V = \varnothing$.

We say that t is a clean *trace, denoted by* $t \in \mathsf{clean}(N)$, *if* $\forall t' \leq t : \ \mathsf{V}(N,t') = \{\varnothing\}$.

Trace	Description	Nodes	V	S	P
$t_0 = [\overline{a_1}4]$	4 days without paying.	(n_6, n_0)	2	$\varnothing$	$\varnothing$
$t_1 = [a_1 3 a_3 8]$	Credit card payment in 3 days and then item shipped in 8 days.	(n_{15}, n_0)	$\varnothing$	3, 5	$\varnothing$
$t_2 = [a_1 3\overline{a_3}15\overline{a_4}8]$	Similar to t_1 but the item is not shipped.	(n_{13}, n_4)	5	3	$\varnothing$
$t_3 = [a_1 3\overline{a_3}15a_5 2\overline{a_4}6]$	Similar to t_2 but with a penalization.	(n_{13}, n_4)	5	3	8
$t_4 = [a_2 2\overline{a_3}15a_4 4]$	Paypal payment in 2 days, item not received but refunded in 19 days.	(n_{15}, n_5)	$\varnothing$	4, 7, 5	$\varnothing$
$t_5 = [a_2 2\overline{a_3}15a_4 4a_5 1]$	Similar to t_4 but a penalization is made.	(n_{15}, n_5)	$\varnothing$	4, 7, 5	8

Table 1: Trace examples for $NTA_{P\&S}$.

Comming back to our running example $NTA_{P\&S}$, let us analyse the following maximal traces of Table 1. The *good* traces will be t_1, t_4 and t_5 since their violation sets are empty but not their satisfaction sets. From these traces only t_1 corresponds to a *clean* trace since t_4 and t_5 have violated the shipment clause 5, however they have been recovered via R_1.

Definition 6 *Let C be an NTA corresponding to a C-O diagram. We say that C is consistent if the following conditions hold:*

- $\mathsf{clean}(C) \cap \mathsf{good}(C) \neq \varnothing$. *This means that there is a way to meet contracts without making any violations.*
- $\forall cn \in CN\ \exists t \in \mathsf{clean}(C) \cap \mathsf{good}(C) :\ \exists S \in \mathsf{S}(C,t) :\ cn \in S$. *That is there is a way to meet all obligations and prohibitions without making any violation.*

Our $NTA_{P\&S}$ example satisfies both conditions since trace t_1 is a *good* and *clean* trace that meets both obligations, the payment and the shipment.

As we have indicated previously, we assume that implementations are given as networks of timed automata. Implementations usually need to implement a single action by making several simple actions. For instance let us suppose that a contract specifies that a payment can be done by credit card. When implementing the payment procedure, several invisible steps like connecting with the bank or checking the credit card should be performed. All these actions are not considered in the specification of the contract and they should not be taken into account. All we need in this case is the amount of time required to perform these actions. Thus, these implementation traces may contain actions that are not considered in the contract, so we need to *hide* these actions.

Definition 7 *Let us consider $ACT \subseteq ACT'$ and $t \in (ACT' \times \mathbb{R}^+)^*$. We consider the operator hide_{ACT} defined as follows:*

- $\mathsf{hide}_{ACT}([]) = []$
- $\mathsf{hide}_{ACT}([ad] \cdot t) = [ad] \cdot \mathsf{hide}_{ACT}(t)$ *for $a \in ACT$, $a \neq \tau$*
- $\mathsf{hide}_{ACT}([ad] \cdot t) = d + \mathsf{hide}_{ACT}(t)$ *for $a \notin ACT$ or $a = \tau$, where the operator $+$ adds d units of time to the last action of t. Formally it is defined as follows:*
  - $d + [] = []$
  - $d + ([ad_1] \cdot t) = [a(d_1 + d)] \cdot t$

[3] A maximal trace is a trace that cannot be extended anymore: if $t \in \mathsf{tr}(N)$ but $t \cdot [ad] \notin \mathsf{tr}(N)$ for all $a \in ACT$ and $d \in \mathbb{R}^+$.

Let us consider the following trace $t_6 = [a_1 3 a_3' 2 a_3'' 2 a_3 4]$ belonging to a possible implementation of our contract. The actions a_3' and a_3'' are internal actions of the implementation (for instance the seller obtains the deliver company list related to the shipment address a_3' and sends the shipment info to the deliverer a_3''). Therefore, the result of $\mathsf{hide}_{ACT}(t_6) = [a_1 3 a_3 8]$, where the internal actions have been omitted and the intermediate time delays are 2 + 2 + 4 = 8.

Now, we have all the machinery needed to define our conformance relation. We will consider that an implementation satisfies a contract if a) there is at least one trace that execute all the actions expressed in the obligations in due time, and not any actions from the prohibitions; that is, satisfying all the obligations and prohibitions expressed in the contract, and b) if at any time a violation occurs, then it will be repaired in the future. In our example, the ideal implementation should be able to "allow the user to at least pay with either credit card or paypal in 3 days, and then, the seller send the item in time". This ideal behavior is represented by condition a), since it gathers all contract obligations and prohibitions. However we should be most realistic and think that all systems are prone to errors, then implementations can as well fail in some occasions. But if they do, then they should been able to recover somehow. That is the idea behind the second condition, that is, if a seller does not send the item, he should at least refund the buyer.

Definition 8 *Let us consider a consistent contract specification C and an implementation I, we say that I conforms C, written I* conf *C, iff*

- *For any cn* $\in$ *CN there exists* $t \in \mathsf{tr}(I)$ *such that* $\mathsf{hide}_{ACT}(t) \in \mathsf{clean}(C) \cup \mathsf{good}(C)$ *and* $\exists S \in \mathsf{S}(I, \mathsf{hide}_{ACT}(t)) :$ *cn* $\in S$.
- *If there exists* $t \in \mathsf{tr}(I)$ *and cn* $\in$ *CN with* $\exists V \in \mathsf{V}(C, \mathsf{hide}_{ACT}(t)) :$ *cn* $\in V$, *there exists* t' *such that* $t \cdot t' \in \mathsf{tr}(I)$ *such that* $\mathsf{hide}_{ACT}(t \cdot t') \in \mathsf{tr}(C)$ *and* $\forall V' \in \mathsf{V}(C, \mathsf{hide}_{ACT}(t \cdot t')) :$ *cn* $\notin V'$.

Let us consider the following implementations I_1, I_2 and I_3 where $\mathsf{tr}(I_1) = \{t_1, t_2\}$, $\mathsf{tr}(I_2) = \{t_4\}$ and $\mathsf{tr}(I_3) = \{t_1, t_4\}$ of our running example $NTA_{P\&S}$. The implementation I_1 satisfies the first condition since t_1 is good and clean and satisfies all the $cn \in CN$, although it does not satisfies the second because t_2 violates clause 5, which is never repaired. Thus implementation I_1 does not conform the given contract. Regarding to I_2, we have the opposite situation, here trace t_4 violates the clause 5, but reparation R_1 is now applied to refund the buyer. Therefore this trace satisfies the second condition but not the first one because it does not includes all the $cn \in CN$. Finally, implementation I_3 is the only one that conforms the contract written as I_3 conf $NTA_{P\&S}$, since it includes t_1 and t_2, which fulfil both conditions.

We are now interested in a comparison of different implementations of a consistent contract, taking into account the permissions allowed for each implementation. This comparison will be based on the permissions performed by an implementation in such a way that an implementation will be considered *better* than other if it is able to perform more permissions. In our example we can consider two implementations, one that after the seller refunds the buyer (because the item has not been sent), allows him to penalize the seller; and other implementation, which does not allow penalizations. In this case, we will say that the first one is better than the former one.

Definition 9 *Let us consider a consistent contract specification C and two implementations* I_1 *and* I_2 *such that* I_1 conf *C and* I_2 conf *C. We say that* I_1 *is better with respect to the permissions than* I_2, *written* $I_2 \leq_P I_1$ *iff for any* $t_2 \in \mathsf{tr}(I_2)$ *such that* $\forall V \in \mathsf{V}(C, \mathsf{hide}_{ACT}(t_2)) : V = \varnothing$ *there is a trace* $t_1 \in \mathsf{tr}(I_1)$ *such that* $\forall V \in \mathsf{V}(C, \mathsf{hide}_{ACT}(t_1)) : V = \varnothing$ *and for any* $P_1 \in P(C, \mathsf{hide}_{ACT}(t_1))$ *there exists* $P_2 \in P(C, \mathsf{hide}_{ACT}(t_2))$ *such that* $P_2 \subseteq P_1$.

Let us consider two new implementations I_4, I_5, where $\mathsf{tr}(I_4) = [t_1, t_4]$ and $\mathsf{tr}(I_5) = [t_1, t_5]$. Both I_4 and I_5 conform to C, as they have at least one trace (t_1) fulfilling all the obligations and prohibitions, and traces t_4 for I_4 and t_5 for I_5 violate a clause, but the corresponding reparation is performed on time. That

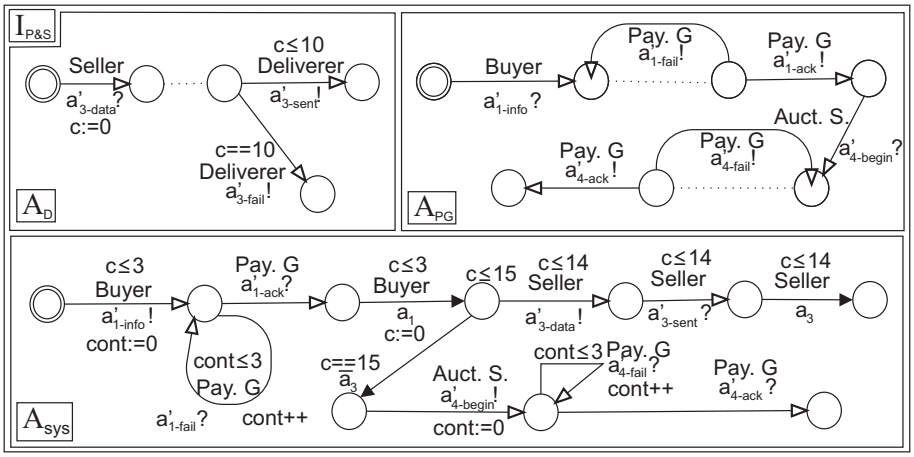

Figure 3: Payment_Shipment implementation example.

implies that both I_4 and I_5 conf $NTA_{P\&S}$. However, the permission set for I_4 is empty, whereas for I_5 clause 8 is the permission set. Thus, $P_4 \subseteq P_5$, implying that $I_4 \leq_P I_5$.

In Figure 3, an implementation of the *Paymen_Shipment* example $I_{P\&S}$ is presented where $I_{P\&S} = \{A_D, A_{PG}, A_{SYS}\}$. A_{SYS} is the main automata where the main actions concerning to the contract are implemented. A_D and A_{PG} implement the behaviors of a deliverer and a payment gateway, both automata present some doted lines to describe a set of internal actions that we abstract to simplify the example. All the actions described in these two automata are synchronization actions whose counterparts are defined in the main automata. The deliverer automaton consists of three actions: the first one is used to start the deliverer process by receiving the item data and delivery address, then, the second one and the third are used to inform the seller whether the delivery has succeed within a time window of 10 days. The payment gateway is in charge of performing two processes: charging the buyers credit card and perform the refund if needed. They are performed via actions $a'_{1-\ldots}$ and actions $a'_{4-\ldots}$ where fail and ack actions are used to communicate whether the operation has succeed or not, respectively.

Let us now analyze this implementation. The main answer to decipher is if $I_{P\&S}$ conf $NTA_{P\&S}$. We can observe that the above defined trace t_1 is obtained hiding the following trace $t'_1 = [a'_{1-info}0 a'_{1-ack}3\ a_1 0 a'_{3-data}0\ a'_{3-ack}8\ a_3 0]$, that is, $t_1 = \mathsf{hide}_{ACT}(t'_1)$, and $t'_1 \in \mathsf{tr}(I_{P\&S})$. As we have shown before, t_1 satisfies the first condition. Regarding to the second condition, note that when a contract is broken it is not necessary that the contract is always repaired, but it should exist at least one trace allowing it[4]. This occurs in trace t_4, which can be obtained hiding the $I_{P\&S}$ trace $t'_4 = [a'_{1-info}0\ a'_{1-ack}2\ a_1 0\ \overline{a_3}15\ a'_{1-begin}0\ a'_{1-ack}4\ a_4 0]$ and substituting a_2 by a_1, that is, substituting the equivalent actions "paypal" payment for a "credit card" payment. Thus, we show that the conformance relation is held by our example, since it fulfils both criteria.

5 Conclusions

In this paper we have used the formal semantics based on NTAs (Network of Timed Automata) for normative contracts written in terms of C-O diagrams introduced in [10] in order to define a conformance relation between a contract and an implementation. We have introduced the notion of *consistent* contracts

[4]Cont variable is used to force the refund for three times. If the refund is not feasible then a fail action is executed.

on the basis on their corresponding NTA, as those NTAs that allow to find final traces without violating any clauses. Then, implementations of contracts are also NTAs, which must satisfy all the obligations and prohibitions, or in the event of a violation, implement the corresponding reparation. These implementations are said to be conforming to the contract. We have also presented a first comparison relation between implementations, on the basis of the permissions allowed for each one. We intend to define a set of implementation comparisons, taking into account the number of clauses that have been violated, or assigning a weight to some clauses, thus considering some clauses as more important.

References

[1] R. Alur & D.L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] D. Chiu, S. Cheung & S. Till (2003): *A Three-Layer Architecture for E-Contract Enforcement in an E-Service Environment*. *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36)*, pp. 74–83, doi:10.1109/HICSS.2003.1174188.

[3] F. Dignum & H. Weigand (1995): *Modelling Communication between Cooperative Systems*. *Proceedings of Advanced Information Systems Engineering (CAISE'95)*, pp. 140–153, doi:10.1007/3-540-59498-1_243.

[4] G. Governatori, Z. Milosevic & S. Sadiq (2006): *Compliance checking between business processes and business contracts*. *Proceedings of the 10th IEEE Conference on Enterprise Distributed Object Computing*, pp. 221–232, doi:10.1109/EDOC.2006.22.

[5] J. Hatcliff, G.T. Leavens, k.R.M. Leino, P. Muller & M. Parkinson (2009): *Behavioral Interface Specification Languages*. Technical Report CS-TR-09-01, School of EECS, University of Central Florida, doi:10.1145/2187671.2187678.

[6] P.R. Krishna, K. Karlapalem & A.R. Dani (2005): *From Contract to E-Contracts: Modeling and Enactment*. *Information Technology and Management* 6(4), pp. 363–387, doi:10.1007/s10799-005-3901-z.

[7] A. Lomuscio, H. Qu & M. Solanki (2008): *Towards verifying contract regulated service composition*. *Proceedings of IEEE International Conference on Web Services (ICWS 2008)*, pp. 254–261, doi:10.1109/ICWS.2008.115.

[8] O. Marjanovic & Z. Milosevic (2001): *Towards formal modeling of e-Contracts*. *Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference*, pp. 59–68, doi:10.1109/EDOC.2001.950423.

[9] E. Martínez, G. Díaz & M. E. Cambronero (2011): *Contractually Compliant Service Compositions*. *ICSOC 2011 - The Ninth International Conference on Service Oriented Computing*, pp. 636–644, doi:10.1007/978-3-642-25535-9_50.

[10] E. Martínez, G. Díaz, M. E. Cambronero & G. Schneider (2010): *A Model for Visual Specification of e-Contracts*. In: *The 7th IEEE International Conference on Services Computing (IEEE SCC'10)*, pp. 1–8, doi:10.1109/SCC.2010.32.

[11] E. Martínez, G. Díaz, M. E. Cambronero & G. Schneider (2012): *Specification and Verification of Normative Specifications using C-O Diagrams*. https://www.dsi.uclm.es/descargas/thecnicalreports/DIAB-12-05-1/TSE11.pdf.

[12] P. McNamara (2006): *Deontic Logic*. In: *Gabbay, D.M., Woods, J., eds.: Handbook of the History of Logic*, 7, North-Holland Publishing, pp. 197–289, doi:10.1016/S1874-5857(06)80029-4.

[13] J. C. Okika & A. P. Ravn (2008): *Classification of SOA Contract Specification Languages*. In: *2008 IEEE International Conference on Web Services (ICWS'08)*, IEEE Computer Society, pp. 433–440, doi:10.1109/ICWS.2008.36.

[14] J. Tretmans (1999): *Testing Concurrent Systems: A Formal Approach*. In: *CONCUR'99, LNCS 1664*, Springer, pp. 46–65, doi:10.1007/3-540-48320-9_6.