Specification and Verification of Normative Specifications using C-O Diagrams

Enrique Martínez, M. Emilia Cambronero, Gregorio Díaz and Gerardo Schneider

Abstract—C-O Diagrams have been introduced as a means to have a more 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 fulfilment of their obligations and prohibitions. In such diagrams we are also able to represent absolute and relative timing constraints. In this paper we present a formal semantics for C-O Diagrams based on timed automata extended with an ordering of states and edges in order to represent different deontic modalities. As a proof of concept we apply our approach to two different case studies.

Index Terms—Normative documents, electronic contracts, deontic logic, formal verification, visual models, timed automata, C-O Diagrams.

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*) [1]. This notion of contracts basically relies on the Hoare's 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 [2]. Here it is possible to express contracts on the history of events, including causality properties. However, the approach is limited when it comes to contracts containing exceptional behaviour, since the focus is mainly on the interaction concerning expected (and prohibited) behaviour.

In the context of SOA, there are different service contract specification languages, like ebXML [3], WSLA [4], and WS-Agreement [5]. These standards and specification languages suffer from one or more of the following problems: They are restricted to bilateral contracts, lack 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 [6] where a taxonomy is presented.

More recently, some researchers have investigated how to adapt deontic logic [7] to define (consistent) contracts targeted to software systems where the focus is on the normative notions of obligation, permission and prohibition, including sometimes exceptional cases (e.g., [8]). Independently of the application domain, there still is need to better fill the gap between a contract understood by non-experts in formal methods (for its use), its logical representation (for reasoning), and its internal machine-representation (for runtime monitoring, and to be manipulated by programmers). We see two possible ways to bridge this gap: i) to develop suitable techniques to get a good translation from contracts written in natural language into formal languages, and ii) to provide a graphical representation (and tools) to manipulate contracts at a high level, with formal semantics supporting automatic translation into the formal language. We take in this paper the second approach.

In [9] we have introduced *C-O Diagrams*, 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 [7]), as well as *reparations* describing contractual

- *Enrique Martínez, M. Emilia Cambronero and Gregorio Díaz are with the Department of Computer Science, University of Castilla-La Mancha, Albacete, Spain.*
  E-mail: {emartinez, gregorio, emicp}@dsi.uclm.es
- *Gerardo Schneider is with the Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden.*
  E-mail: gersch@chalmers.se

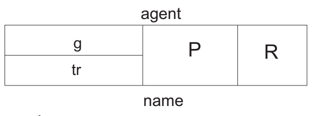

Fig. 1. Box structure

clauses in case of not fulfilment of obligations and prohibitions. Besides, *C-O Diagrams* permit to define real-time constraints. In [10] some of the satisfaction rules needed to check if a timed automaton satisfies a *C-O Diagram* specification were defined.

The goal of this paper is to further develop our previous work, in particular we present here a formal semantics for *C-O Diagrams* based on timed automata, extended with an ordering of states and edges. As a proof of concept, two case studies are presented, one of them in the field of Web service composition and another one in the field of requirements engineering.

The rest of the paper is structured as follows: Section 2 presents *C-O Diagrams* and their syntax, Section 3 develops the formal semantics of *C-O Diagrams*, and Section 4 explains the implementation of the resulting timed automata in UPPAAL [11]. Section 5 presents a case study about an *Online Auctioning Process* and Section 6 presents a case study about the requirements engineering of an *Adaptive Cruise Control* system. Finally, the related work is discussed in Section 7 and the conclusions and future work are commented in Section 8.

2 C-O Diagrams Description and Syntax

In Fig. 1 we show the basic element of *C-O Diagrams*. It is called a **box** and it is divided into four fields. On the left-hand side of the box we specify the conditions and restrictions. 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 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 (obligations, permissions and prohibitions) 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 and an agent. The *name* is useful both to describe the clause and to reference the box from other clauses, so it must be unique. The *agent* indicates who is the performer of the action.

Example 1: For example, if we consider a contract about the operation of a ***coffee machine***, some of the following clauses have to be considered:

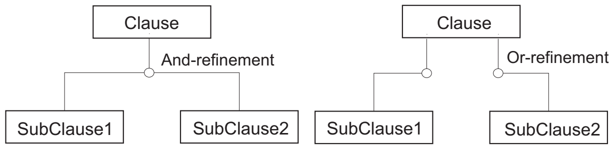

Fig. 2. AND/OR refinements in *C-O Diagrams*

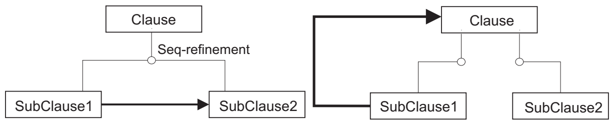

Fig. 3. SEQ refinement and repetition in *C-O Diagrams*

- "The coffee machine is *obliged* to deliver coffee after payment in less than *one minute*", that is an **obligation** including a **deadline**.
- "The client is *permitted* to choose the option coffee with milk", that is an example of **permission**.
- "The client is *forbidden* to pay with coins different from Euros or Dollars", that is an example of **prohibition**.
- "The coffee machine is *obliged* to deliver milk if *coffee with milk has been chosen*", that is a **obligation** applied only if a **condition** is fulfilled.
- "The coffee machine is *obliged* to refund money if *coffee is not deliver*", that is a **reparation** to the **obligation** of delivering coffee. □

The basic element of *C-O Diagrams* can be refined by using AND/OR/SEQ refinements. The aim of these refinements is to capture the hierarchical clause structure followed by most contracts. An **AND-refinement** (left-hand side of Fig. 2) means that all the subclauses must be satisfied in order to satisfied the parent clause. An **OR-refinement** (right-hand side of Fig. 2) means that it is only necessary to satisfy one of the subclauses in order to satisfy the parent clause, so as soon as one of its subclauses is fulfilled, we conclude that the parent clause is fulfilled as well. A **SEQ-refinement** (left-hand side of Fig. 3) means that the norm specified in the target box (*SubClause2* in Fig. 3) must be fulfilled after satisfying the norm specified in the source box (*SubClause1* in Fig. 3). By using these structures we can build a hierarchical tree with the clauses defined by a contract, where the leaf clauses correspond to the atomic clauses, that is, to the clauses that cannot be divided into subclauses. There is another structure that can be used to model **repetition**. This structure is represented as an arrow going from a subclause to one of its ancestor clauses (or to itself), meaning the repetitive application of all the subclauses of the target clause after satisfying the source subclause. For instance, in the right-hand side of Fig. 3, we have an **OR-refinement** with an arrow going from *SubClause1* to *Clause*. It means that after satisfying *SubClause1* we apply *Clause* again, but not after satisfying *SubClause2*.

It is only considered the specification of *atomic actions* in the **P** field of the leaf boxes of our diagrams. The composition of actions can be achieved by means of the different kinds of refinement. In this way, an AND-refinement can be used to model *concurrency*

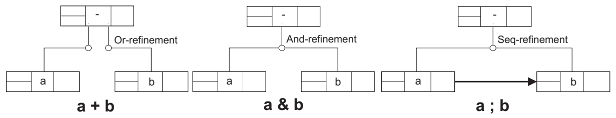

Fig. 4. Compound actions in *C-O Diagrams*

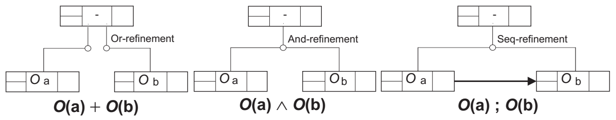

Fig. 5. Composition of norms in *C-O Diagrams*

"&" between actions, an OR-refinement can be used to model a *choice* "+" between actions, and a SEQ-refinement can be used to model *sequence* ";" of actions. In Fig. 4 we can see an example about how to model these compound actions through refinements, given two atomic actions *a* and *b*.

The *deontic norms* (obligations, permissions and prohibitions) that are applied over these actions can be specified in any box of our *C-O Diagrams*, affecting all the actions in the leaf boxes that are descendants of this box. If it is the case that the box where we specify the deontic norm is a leaf, the norm only affects the atomic action we have in this box. It is used an upper case "*O*" to denote an obligation, an upper case "*P*" to denote a permission, and an upper case "*F*" to denote a prohibition (forbidden). These letters are written in the top left corner of field **P**.

The composition of deontic norms is also achieved by means of the different refinements we have in *C-O Diagrams*. Thus, an AND-refinement corresponds to the *conjunction* operator "$\wedge$" between norms, an OR-refinement corresponds to the *choice* operator "+" between norms, and a SEQ-refinement corresponds to the *sequence* operator ";" between norms.

Example 2: For example, we can imagine having a leaf box specifying the obligation of performing an action *a*, written as *O*(a), and another leaf box specifying the obligation of performing an action *b*, written as *O*(b). These two norms can be combined in the three different ways above mentioned through the different kinds of refinement (Fig. 5). Considering the ***coffee machine*** example again, we can suppose that action *a* corresponds to "delivers coffee" and action *b* corresponds to "delivers milk", so if we consider the combination of both obligations with an AND-refinement, it is specified that both obligations have to be satisfied in any order. □

There are some syntactic constraints to be taken into account when defining *C-O Diagrams*. First, exactly one deontic norm must be specified in each one of the branches of our hierarchical tree, i.e., we cannot have an action without a deontic norm applied over it and we cannot have deontic norms applied over other deontic norms. Also, *agents* must only be specified in the boxes where a deontic norm is defined, being each agent associated to a concrete deontic norm. Finally, the *repetition* of both, actions and deontic norms, can be achieved by means of the repetition structure we have in *C-O Diagrams*.

Definition 1: (*C-O Diagrams* Syntax) We consider a finite set of real-valued variables $\mathcal{C}$ standing for clocks, a finite set of non-negative integer-valued variables $\mathcal{V}$, a finite alphabet Σ for atomic actions, a finite set of identifiers $\mathcal{A}$ for agents, and another finite set of identifiers $\mathcal{N}$ for names. The greek letter ϵ means that an expression is empty. We use C to denote the contract modelled by a *C-O Diagram*. The diagram is defined by the following EBNF grammar:

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

where $a \in \Sigma$, $agent \in \mathcal{A}$ and $name \in \mathcal{N}$. Guard g is ϵ or a conjunctive formula of atomic constraints of the form: $v \sim n$ or $v - w \sim n$, for $v, w \in \mathcal{V}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$, whereas time restriction tr is ϵ or a conjunctive formula of atomic constraints of the form: $x \sim n$ or $x - y \sim n$, for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. O, P and F are the deontic operators corresponding to obligation, permission and prohibition, respectively, where $O(C_2)$ states the obligation of performing C_2, $F(C_2)$ states prohibition of performing C_2, and $P(C_2)$ states the permission of performing C_2. And, Or and Seq are the operators corresponding to the refinements we have in *C-O Diagrams*, AND-refinement, OR-refinement and SEQ-refinement, respectively. □

The simplest contract we can have in *C-O Diagrams* is that consisting of only one box including the elements $agent$ and $name$. Optionally, we can specify a guard g and a time restriction tr. We also have a deontic operator (O, P or F) applied over an atomic action a, and in the case of obligations and prohibitions it is possible to specify another contract C as a reparation.

We use C_1 to define a more complex contract where we combine different deontic norms by means of any of the different refinements we have in *C-O Diagrams*. In the box where we have the refinement into C_1 we cannot specify an agent nor a reparation because these elements are always related to a single deontic norm, but we still can specify a guard g and a time restriction tr that affect all the deontic norms we combine.

Once we write a deontic operator in a box of our diagram, we have two possibilities as we can see in the specification of C_2: we can just write a simple action a in the box, being the deontic operator applied only over it, or we can refine this box in order to apply the deontic operator over a compound action. In this case we have that the subboxes (C_3) cannot define a new deontic operator as it has already been defined in the parent box (affecting all the subboxes).

Example 3: For example, we can consider $C := (Buyer, Example1, \epsilon, \epsilon, O(pay), C')$ as a very simple

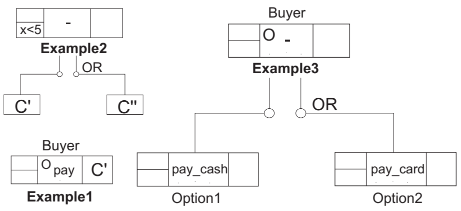

Fig. 6. Syntax examples

contract specifying for a buyer the obligation of paying, otherwise contract C' comes into effect (Fig. 6, on the bottom left). $C := (\epsilon, Example2, \epsilon, x < 5, C' \, Or \, C'', \epsilon)$ is a composed contract specifying that contract C' or contract C'' must be satisfied in order to satisfy C within 5 time units (Fig. 6, on the top left). Finally, $C := (Buyer, Example3, \epsilon, \epsilon, O(C' \, Or \, C''), \epsilon)$, where we have that $C' := (\epsilon, Option1, \epsilon, \epsilon, pay\_cash, \epsilon)$ and $C'' := (\epsilon, Option2, \epsilon, \epsilon, pay\_card, \epsilon)$, is a contract specifying for a buyer the obligation of paying by cash or by credit card (Figure 6, on the right). □

3 *C-O Diagrams* SEMANTICS

The *C-O Diagrams* semantics is defined by means of a transformation into a *Network of Timed Automata* (NTA), that is defined as a set of timed automata [12], [13] that run simultaneously, using the same set of clocks and variables, and synchronizing on the common actions. In this section we first present an extension of timed automata suitable as a semantic domain for *C-O Diagrams*, and we finish by showing the transformation rules to translate our diagrams into such timed automata.

3.1 Timed automata extended with ordering

In what follows we consider a finite set of real-valued variables $\mathcal{C}$ ranged over by $x, y, \ldots$ standing for clocks, a finite set of non-negative integer-valued variables $\mathcal{V}$, ranged over by $v, w, \ldots$ and a finite alphabet Σ ranged over by $a, b, \ldots$ standing for actions. We will use letters $r, r', \ldots$ to denote sets of clocks. We will denote by $Assigns$ the set of possible assignments, $Assigns = \{v := expr \mid v \in \mathcal{V}\}$, where *expr* are arithmetic expressions using naturals and variables. Letters $s, s' \ldots$ will be used to represent a set of assignments.

A *guard or invariant condition* is a conjunctive formula of atomic constraints of the form: $x \sim n$, $x - y \sim n$, $v \sim n$ or $v - w \sim n$, for $x, y \in \mathcal{C}$, $v, w \in \mathcal{V}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. The set of guard or invariant conditions will be denoted by $\mathcal{G}$, ranged over by $g, g', \ldots$.

Definition 2: (Timed Automaton)
A *timed automaton* is a tuple (N, n_0, E, I), where N is a finite set of locations (nodes), $n_0 \in N$ is the initial location, $E \subseteq N \times \mathcal{G} \times \Sigma \times \mathcal{P}(Assigns) \times 2^{\mathcal{C}} \times N$ is the set of edges, where the subset of *urgent edges* is called $E_u \subseteq E$, and they will graphically be distinguished as they will have their arrowhead painted in white. $I : N \rightarrow \mathcal{G}$ is a function that assigns invariant conditions (which could be empty) to locations. □

From now on, we will write $n \xrightarrow[s]{g,a,r} n'$ to denote $(n, g, a, s, r, n') \in E$, and $n \xrightarrow[s]{g,a,r}_u n'$ when $(n, g, a, s, r, n') \in E_u$.

In an NTA we distinguish two types of actions: internal and synchronization actions. Internal actions can be executed by the corresponding automata independently, and they will be ranged over the letters $a, b \ldots$. Synchronization actions, however, must be executed simultaneously by two automata, and they will be ranged over letters $m, m', \ldots$ and come from the synchronization of two actions $m!$ and $m?$, executed from two different automata. Due to lack of space, we refer the reader to [14] for the definition of the semantics of timed automaton and NTA.

To specify the *C-O Diagrams* semantics, we add the definition of two orderings, $\prec_N$ and $\prec_E$, where:

- $\prec_N$ is a (strict, partial) ordering on N where $n \prec_N n'$ means that node n is *better* than node n'.
- $\prec_E$ is a (strict, partial) ordering on E where $e \prec_N e'$ means that edge e is *better* than edge e'.

We also add a **violation set** $V(n)$ associated to each node n in N, that is the set of contractual obligations and prohibitions that are violated in n.

Definition 3: (Violation Set) Let us consider the set of contractual obligations and prohibitions CN ranged over cn, cn',... standing for identifiers of obligations and prohibitions. We write $n \not\models cn$ to express that obligation or prohibition cn is violated in node n. Therefore, the *violation set* is defined as $V(n) = \{cn \mid cn \in CN$ and $n \not\models cn\}$. □

Another set called **satisfaction set** $S(n)$ is also associated to each node n in N. This set is composed by the contractual obligations and prohibitions that have already been satisfied in n.

Definition 4: (Satisfaction Set) Let us consider the set of contractual obligations and prohibitions COF ranged over cof, cof',... standing for identifiers of obligations and prohibitions. We write $n \models cof$ to express that obligation or prohibition cof has been satisfied in node n (we consider a prohibition satisfied in node n if it has not been violated and cannot be violated anymore because the time frame specified for the prohibition has expired). Hence, the *satisfaction set* is defined as $S(n) = \{cof \mid cof \in COF$ and $n \models cof\}$. □

Once these two sets have been defined, we can formally define the **ordering on nodes** $\prec_N$, by comparing the violation sets and the satisfaction sets of the nodes, and the **ordering on edges** $\prec_E$, by comparing the violation sets and the satisfaction sets of the target nodes of the edges.

Definition 5: (Ordering on Nodes) A *node* n_1 *is better than another node* n_2 if the violation set of n_1 is a proper subset of the violation set of n_2 or, if the violation sets are the same, a node n_1 is better than another node n_2

if the satisfaction set of n_1 is a proper superset of the satisfaction set of n_2, that is, $n_1 \prec_N n_2$ iff $(V(n_1) \subset V(n_2))$ or $(V(n_1) = V(n_2)$ and $S(n_1) \supset S(n_2))$. □

Definition 6: (Ordering on Edges) An *edge* e_1 *is better than another edge* e_2 if the source node is the same in both cases but the violation set of the target node of e_1 is a proper subset of the violation set of the target node of e_2 or, if the violation sets are the same, an edge e_1 is better than another edge e_2 if the satisfaction set of the target node of e_1 is a proper superset of the satisfaction set of the target node of e_2. Considering $e_1 = (n_1, g_1, a_1, s_1, r_1, n_1')$ and $e_2 = (n_2, g_2, a_2, s_2, r_2, n_2')$, $e_1 \prec_E e_2$ iff $(n_1 = n_2)$ and $(V(n_1') \subset V(n_2')$ or $(V(n_1') = V(n_2')$ and $S(n_1') \supset S(n_2')))$. □

Finally, another set called **permission set** $P(n)$ is associated to each node n in N. This set influences neither the ordering on nodes nor the ordering on edges, it is used just to record the permissions in the contract that have been made effective.

Definition 7: (Permission Set) Let us consider the set of contractual permissions CP ranged over *cp*, *cp*′,... standing for identifiers of permissions. We write $n \models cp$ to express that permission *cp* has already been made effective in node n. Then, the *permission set* is defined as $P(n) = \{cp \mid cp \in CP$ and $n \models cp\}$. □

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. In the initial node of the automata we build corresponding to *C-O Diagrams* these three sets are empty. By default, a node keeps in these sets the same content of the previous node when we compose the automata. Only in a few cases the content of these sets is modified (when an obligation or a prohibition is violated, an obligation or a prohibition is satisfied or a permission is made effective).

Example 4: For example, considering a simple contract having only the clause "The coffee machine is *obliged* to deliver coffee after payment in less than *one minute*" of *Example 1*, that we call *Del_Coffee*, in the corresponding automaton we have a node n_0 with empty violation, satisfaction and permission sets before the violation or satisfaction of the obligation ($V(n_0) = \{\}$, $S(n_0) = \{\}$ and $P(n_0) = \{\}$), another node n_1 where the obligation is satisfied and it is added to the satisfaction set ($V(n_1) = \{\}$, $S(n_1) = \{Del\_Coffee\}$ and $P(n_1) = \{\}$) and finally another node n_2 where the obligation is violated and it is added to the violation set ($V(n_2) = \{Del\_Coffee\}$, $S(n_2) = \{\}$ and $P(n_2) = \{\}$). Therefore, according to the definition of the ordering on nodes, we have that the order between these nodes is $n_1 \prec_N n_0 \prec_N n_2$. □

Concerning the **real-time restrictions** tr specified in the contract, the two types of time restrictions we can have in *C-O Diagrams* must be translated in a different way for their inclusion into a timed automaton construction:

- A time restriction specified using **absolute time** must be specified in timed automata by rewriting the terms in which absolute time references occur. For that purpose we define a global clock $T \in \mathcal{C}$ that is never reset during the execution of the automata and, taking into account the moment at which the contract is enacted, we rewrite the absolute time references as deadlines involving clock T and considering the smallest time unit needed in the contract.
- A time restriction specified using **relative time** must be specified in timed automata by introducing an additional clock to register the amount of time that has elapsed since another clause has been satisfied, resetting the additional clock value when this happens and specifying the deadline using it. We call this clock t_{name}, where $name$ is the clause used as reference for the specification of the time restriction. Therefore, we define a set of additional clocks $C_{add} = \{t_{name} \mid t_{name} \in \mathcal{C}\}$ including a clock for every clause that is used as reference in the time restriction of at least another clause.

As a result, the set of clocks of the timed automata would be $\mathcal{C} = \{T\} \cup C_{add}$. When we construct the timed automata corresponding to *C-O Diagrams*, we always consider $(x \geq t1)\, and\, (x \leq t2)$ as the interval corresponding to the time restriction tr of the clause, where $x \in \mathcal{C}$ is the clock used for its specification ($x = T$ in the case of absolute time and $x = t_{name}$ in the case of relative time, being $name$ the clause used as reference), $t1 \in \mathbb{N}$ is the beginning of the interval and $t2 \in \mathbb{N}$ is the end of the interval ($t1 \leq t2$). If tr does not define the lower bound of the interval we take $t1 = 0$, if tr does not define the upper bound of the interval we take $t2 = \infty$, and if $tr = \epsilon$ we take $t1 = 0$, $t2 = \infty$ and $x = T$.

Example 5: For example, in the case of **absolute time** let us consider a clause that must be satisfied between the *5th of November* and the *10th of November*, and that the contract containing this clause is enacted the *31st of October*. If we suppose that *days* is the smallest time unit used in the contract for the specification of real-time restrictions, the time restriction of this clause is written as $(T \geq 5)\, and\, (T \leq 10)$. In the case of **relative time** let us consider a contract with a clause that must be satisfied between *5* and *10* days after another clause $name1$ has been satisfied. In this case we define an additional clock t_{name1} that is reset to zero when clause $name1$ is satisfied ($t_{name1} := 0$) and the time restriction of the other clause is written as $(t_{name1} \geq 5)\, and\, (t_{name1} \leq 10)$. □

3.2 Transformation rules

Once we have given these extensions of the definition of timed automata and we have explained how the

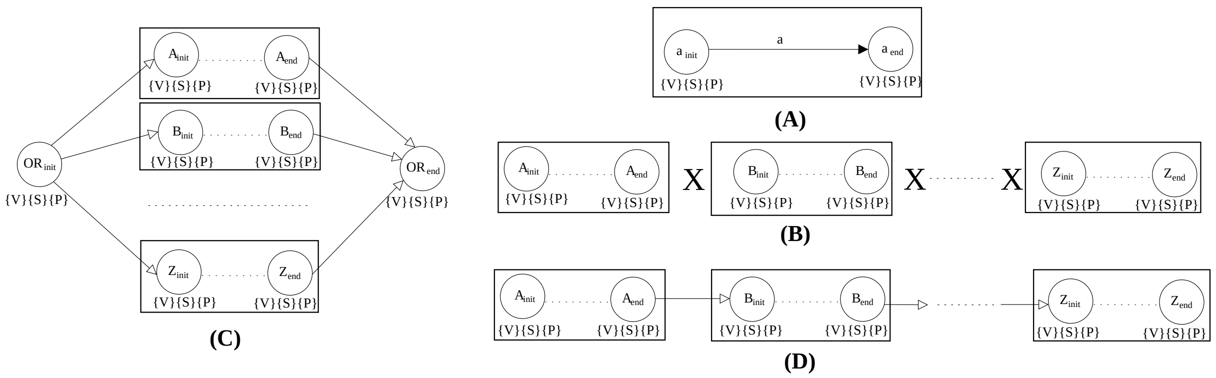

Fig. 7. Automata corresponding to a **simple action** a and to **compound actions**

different kinds of time restriction can be expressed, considering all the different elements we can specify in a *C-O Diagram*, we can define the transformation of the diagrams into timed automata by induction using several transformation rules. We are going to give here an informal description of the transformation rules, a formal definition of each one of these rules can be seen in Appendix A:

(1) An **atomic action** in a *C-O Diagram*, that is, $(\epsilon, name, \epsilon, \epsilon, a, \epsilon)$ corresponds to a timed automaton with an initial node (a_{init}), a final node (a_{end}) and an edge from the initial node to the final node performing action a The violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so $V(a_{init}) = V(a_{end})$, $S(a_{init}) = S(a_{end})$ and $P(a_{init}) = P(a_{end})$. This timed automaton can be seen in Fig. 7-(A).
(2) A **compound action** in a *C-O Diagram* where an **AND-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, And\, C_2\, And\, \ldots\, And\, C_n, \epsilon)$ corresponds to the cartesian product of the automata corresponding to each one of the subcontracts. The violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 7-(B).
(3) A **compound action** in a *C-O Diagram* where an **OR-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, Or\, C_2\, Or\, \ldots\, Or\, C_n, \epsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts is considered as an alternative. The violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 7-(C).
(4) A **compound action** in a *C-O Diagram* where a **SEQ-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, Seq\, C_2\, Seq \ldots Seq\, C_n, \epsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts are connected in sequence. Again, the violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 7-(D).

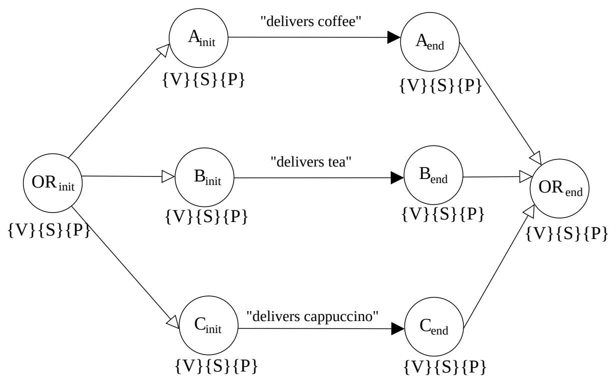

Fig. 8. Example of an **OR-refinement** of actions

Example 6: For example, considering the operation of the ***coffee machine***, we can have the actions “delivers coffee”, “delivers tea” and “delivers cappuccino” composed by an **OR-refinement** (only one of them is performed). The automaton that we obtain, according to rule **(3)**, is shown in Fig. 8. □

Until now, we have seen how the automata corresponding to the different actions (atomic or compound) specified in a *C-O Diagram* are constructed and we have seen that these translations do not modify the content of any of the sets (violation, satisfaction or permission). Next, we define the transformation rules specifying how these “action” automata are modified when we apply a deontic norm (obligation, permission or prohibition) over the actions in the *C-O Diagram*:

(5) The application of an **obligation**, a **permission** or a **prohibition** over an action in a *C-O Diagram*, i.e., $(agent, name, g, tr, O/P/F(C), R)$ corresponds to an automaton where the obligation/prohibition of performing the action specified in the subcontract C can be skipped, fulfilled or violated, whereas the permission of performing the action can be skipped, made effective or not made effective. The resulting timed automata are shown graphically in Fig. 9, where (A) corresponds to obligation, (B) corresponds to permission and (C) corresponds to prohibition. We consider a one of the atomic actions included in the subcontract C.

Example 7: For example, if we consider the clause

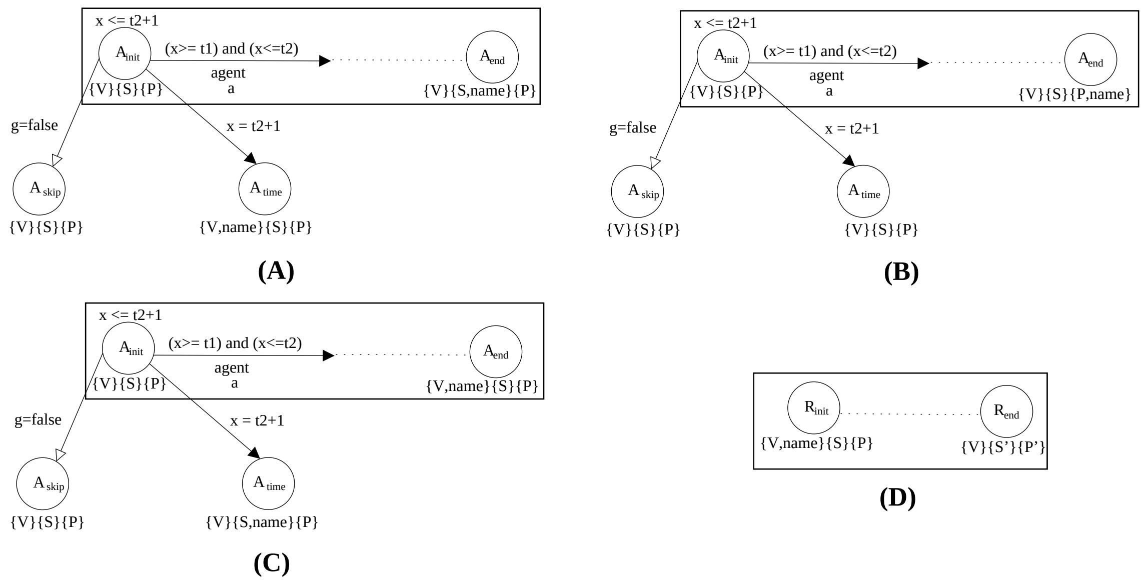

Fig. 9. Automata corresponding to **deontic norms** and automaton corresponding to a **reparation**

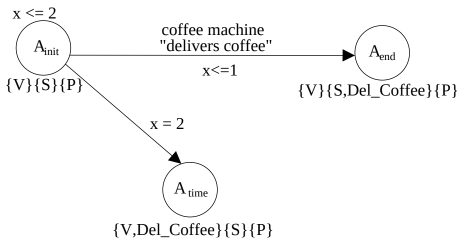

Fig. 10. Example of an **Obligation** over an action

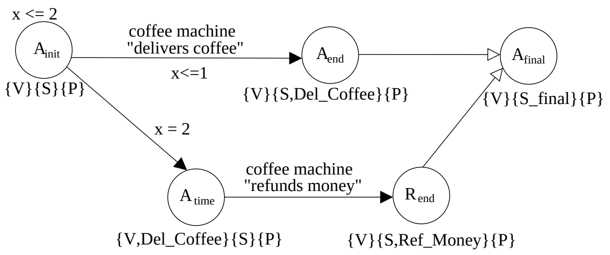

Fig. 11. Example of a **Reparation** to a violation

"The coffee machine is *obliged* to deliver coffee after payment in less than *one minute*", that we call *Del_Coffee*, about the operation of the ***coffee machine***, the automaton that we obtain, according to rule **(5)**, is shown in Fig. 10, where x is the clock used to control the deadline. □

We can see that the above constructions can include a reparation contract R in the cases of obligation and prohibition. If this reparation is defined, we have to construct the automaton corresponding to the reparation contract and integrate this automaton as part of the automaton we have generated for the obligation or prohibition:

(6) An **obligation** or **prohibition** in a *C-O Diagram* specifying a contract **reparation** $R \neq \epsilon$ corresponds to the obligation automaton $O(A)$ or the prohibition automaton $F(A)$ together with the reparation automaton R, considering the node with $name$ in its violation set (A_{vio}) as the initial node of the reparation automaton (R_{init}). In the ending node of the reparation automaton (R_{end}) $name$ is removed from the violation set, as the violation has been repaired. In this node we also have that the satisfaction set and the permission set are different from the ones we have in the initial node of the reparation because we have to include in the satisfaction set all the obligations and prohibitions satisfied in the reparation contract, and in the permission set all the permissions that have been made effective in the reparation contract. This structure is shown graphically in Fig. 9-(D).

Finally, we define how the rules corresponding to different deontic norms are composed when we have a composition of deontic norms in our *C-O Diagram*. To make this composition possible, first we need to have only one ending node in the automata corresponding to the different deontic norms:

(7) If we have an automaton corresponding to an **obligation**, a **prohibition** or a **permission** in a *C-O Diagram*, the corresponding automaton with **only one ending node** is obtained by adding a new ending node and urgent edges from the old ending nodes to this new node, preserving in the new node the violation, satisfaction and permission sets of the previous node. Notice that in the case of obligation and prohibition, if there is no reparation defined, the node violating the norm is a final node of the whole automaton construction where the contract is breached.

Example 8: For example, if together with the obligation of delivering coffee we consider the reparation clause "The coffee machine is *obliged* to refund money if *coffee is not deliver*", that we call *Ref_Money*, about the operation of the ***coffee machine***, the automaton that we obtain, according to rules **(5)**, **(6)** and **(7)**, is shown in Fig. 11, where we have that in node R_{end} the violation of clause *Del_Coffee* has been repaired, so it is removed from the violation set, and in the

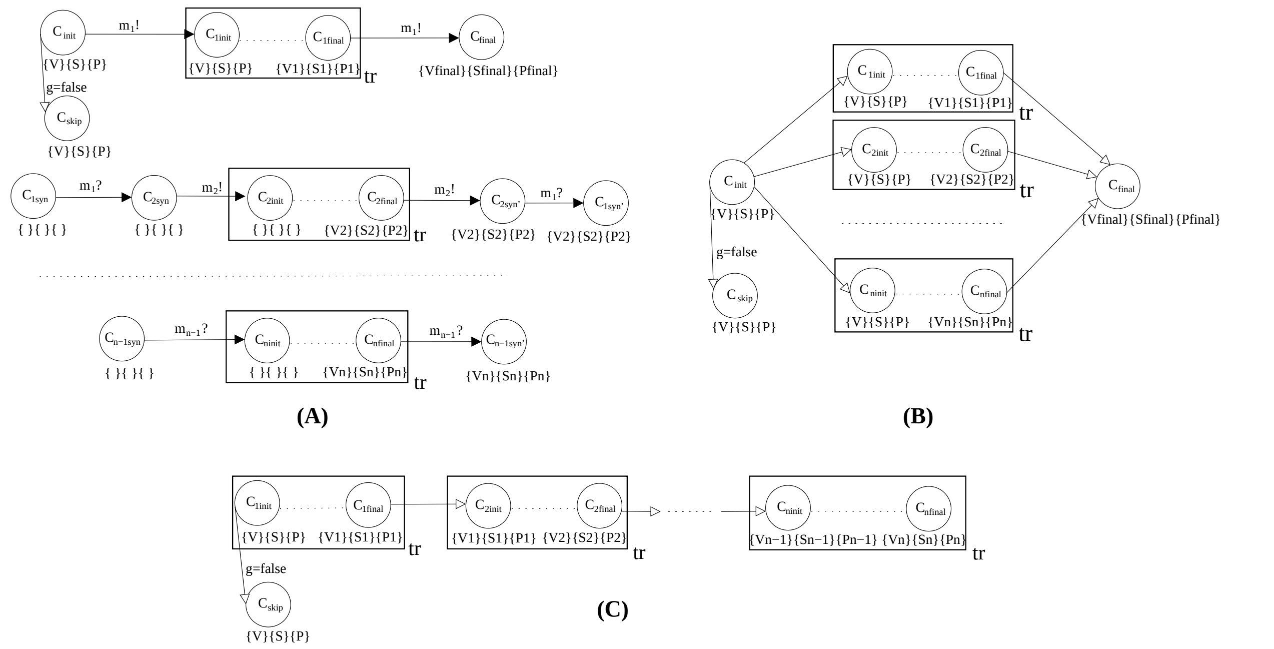

Fig. 12. Automata corresponding to the **compositions of deontic norms**

satisfaction set (S_{final}) of node A_{final} it is kept the satisfaction set of the previous node. □

The composition of the automata corresponding to different deontic norms is now defined by three additional transformation rules:

(8) If several norms are composed by an **AND-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, And\, C_2\, And \ldots And\, C_n, \epsilon)$, their composition corresponds to a network of automata in which we consider all the norms we are composing in parallel. This composition of timed automata is shown graphically in Fig. 12-(A).

(9) If several norms are composed by an **OR-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, Or\, C_2\, Or \ldots Or\, C_n, \epsilon)$, their composition corresponds to an automaton in which the automata corresponding to each one of the norms is considered as an alternative. This composition of timed automata is shown graphically in Fig. 12-(B).

(10) If several norms are composed by a **SEQ-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, Seq\, C_2\, Seq \ldots Seq\, C_n, \epsilon)$, their composition corresponds to an automaton in which the automata corresponding to each one of the norms are connected in sequence. This composition of timed automata is shown graphically in Fig. 12-(C).

Example 9: For example, if we consider a composition with an **AND-refinement** of the clauses "The coffee machine is *obliged* to deliver coffee after payment in less than *one minute*", that we call *Del_Coffee*, and "The coffee machine is *obliged* to deliver milk after payment in less than *one minute*", that we call *Del_Milk*, about the operation of the ***coffee machine***, the network of automata that we obtain, according to

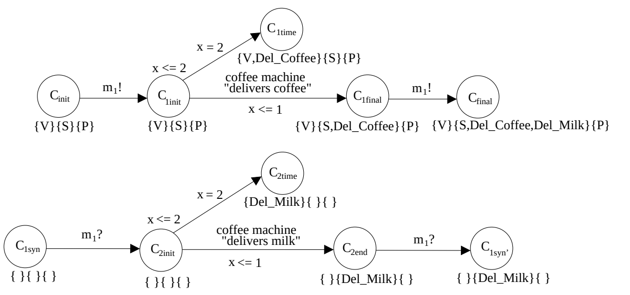

Fig. 13. Example of an **AND-refinement** of obligations

rule **(8)**, is shown in Fig. 13, where *x* is the clock used to control the deadline and *m1* is the urgent channel used to synchronize both automata. □

4 UPPAAL IMPLEMENTATION OF C-O DIAGRAMS

The implementation of the NTAs we have obtained in UPPAAL is quite straightforward as both, the NTA formalism considered by the tool and the NTA formalism that we have considered, are very similar. There are only a few implementation points that need a more detailed explanation:

1) As there is no way in UPPAAL of directly expressing that an edge without synchronisation should be taken without delay, that is, there are no urgent edges, we have to find an alternative way of encoding this behaviour. For this purpose we consider the modelling pattern proposed in [14]. The encoding of urgent edges introduces an extra automaton, that we call $Urgent$, with a single location and a self loop. The self loop synchronises on an urgent channel that we call urg_edge. An edge can now be made urgent by performing the complimentary action.
2) The performance of actions by agents is implemented by means of boolean variables in

UPPAAL. We define a boolean variable called $agent\_action$ for each one of the actions considered in the contract. These variables are initialized to **false** and, when one of the actions is performed by an agent in one of the edges, we update the value of the corresponding variable to **true**.

3) Finally, the violation, satisfaction and permission sets are implemented in UPPAAL by means of boolean arrays and constant integers with the names of the clauses of the contract containing obligations, prohibitions or permissions. We define an array V for violation, an array S for satisfaction, and an array P for permission, all of them initialized to **false**. The size of the arrays V and S is equal to the number of obligations and prohibitions in the contract, whereas the size of the array P is equal to the number of permissions. We also define constant integers with the name of the clauses containing obligations and prohibitions, initializing each one of them to a different value (from 0 to the size of the arrays V and S minus 1), and constant integers with the name of the clauses containing permissions, initializing each one of them to a different value (from 0 to the size of the array P minus 1). These constants are used as indexes in the arrays. When taking a transition where the target node contains at least one modified set (an obligation/prohibition is violated, an obligation/prohibition is satisfied or a permission is made effective), we update to **true** in the proper array the value of the index corresponding to the clause. In the case of repairing an obligation/prohibition violation, the index corresponding to the proper clause in V is set to **false**.

Example 10: For example, let us consider a contract with only one obligation (*Clause_1*), one prohbition (*Clause_2*) and one permission (*Clause_3*). For the implementation of the corresponing NTA in UPPAAL we have to define a boolean array V and a boolean array S of size two (one obligation plus one prohibition), and a boolean array P of size one (only one permission). The constant integers $Clause\_1 = 0$, $Clause\_2 = 1$ and $Clause\_3 = 0$ are also defined as indexes for the arrys. In this way, we can properly update these arrays. For instance, if it is taken a transition where the obligation (*Clause_1*) is satisfied, we update the array S with the operation $S[Clause\_1] = true$, and if if it is taken a transition where the permission (*Clause_3*) is made effective, we update the array P with the operation $P[Clause\_3] = true$. □

5 Case Study: Online Auctioning Process (OAP)

The case study presented in this section is inspired by the motivating example described in [15]. It consists of an *Online Auctioning Process* involving the interaction between three different agents: the *buyer*, the *seller*, and the *auction service*.

The description of the process we are considering here is the following: the online auctioning starts when a *seller* wants to auction an item. Therefore, the *seller* has **one day** to upload valid information about the item he wants to sell, being forbidden the sale of inadequate items such as replicas of designers items or wild animals. Once it has been checked that the item can be auctioned, the *auction service* also has **one day** to publish the auction of the item. After that, the *buyer* can place bids during **seven days**. When this period of time is over, if the bid placed by the *buyer* is the highest one, 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 (for example not allowing the *seller* to auction new items for a period of time). However, if the reception of the item by the *buyer* is acknowledge on time, the auction process is considered finished successfully.

In Table 1 we show a list of the obligations, permissions and prohibitions that can be inferred from the description of the process, where the obligation specified by **Clause 9** and the permission specified by **Clause 10** are a possible reparation for the violation of **Clause 8**. In this table we can see that there are nine clauses specifying real-time constraints. In **Clause 2** and **Clause 3** it is specified that after selecting an item to auction (**Clause 1**), the *buyer* has one day to upload valid information about the item, being forbidden to auction an inadequate item. In **Clause 4** we have that the *auction service* has one day to publish the auction of the item after it has been checked (**Clause 2** and **Clause 3**). In **Clause 5** we have that the *buyer* is allowed to place bids for the item during seven days after publication (**Clause 4**). In **Clause 6** and **Clause 7** it is specified that the *buyer* has three days to pay the item (respectively by credit card or PayPal) after winning the auction. **Clause 8** specifies that the *seller* has fourteen days to send the item to the *buyer* after payment (**Clause 6** or **Clause 7**). Finally, **Clause 9** and **Clause 10** specify that the *auction service* has seven days to refund the payment to the *buyer* if the item is not received on time and can penalize the *seller* in this period of time. We can also see in Table 1 that **Clause 6**, **Clause 7** and **Clause 8** are conditioned to the result of the auction, that is, they are only applied if the *buyer* has won the auction.

The main problem we have with this textual specification is that it is no clear the relationship existing between the different clauses, which makes difficult

Clause	Agent	Modality	Action	Condition	Time
1	*Seller*	*Permission*	Can select an item to auction	∅	∅
2	*Seller*	*Prohibition*	Auctions a fraudulent item	∅	One day after selection (t_1)
3	*Seller*	*Obligation*	Uploads valid information	∅	One day after selection (t_1)
4	*Auction S.*	*Obligation*	Publishes the auction	∅	One day after checked (t_2)
5	*Buyer*	*Permission*	Can place bids for the item	∅	Seven days after publication (t_3)
6	*Buyer*	*Obligation*	Pays the item by credit card	Highest bid (g_1)	Three days after auction (t_4)
7	*Buyer*	*Obligation*	Pays the item by PayPal	Highest bid (g_1)	Three days after auction (t_4)
8	*Seller*	*Obligation*	Sends the item to the *buyer*	Highest bid (g_1)	Fourteen days after auction (t_5)
9	*Auction S.*	*Obligation*	Refunds the payment	∅	Seven days after violation of *C.8* (t_6)
10	*Auction S.*	*Permission*	Can penalize the *seller*	∅	Seven days after violation of *C.8* (t_6)

TABLE 1
Norms of the *Online Auctioning Process* contract

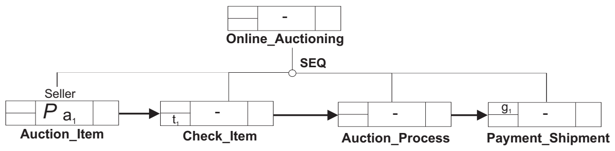

Fig. 14. Top-level of the *Online Auctioning Process*

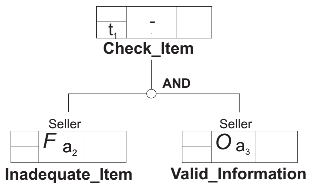

Fig. 15. Decomposition of clause *Check_Item*

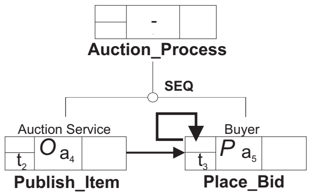

Fig. 16. Decomposition of clause *Auction_Process*

any kind of analysis. Therefore, we aim at a specification language (*C-O Diagrams*) that clearly defines the relationship between the different clauses, but not so formal that an expert is needed.

5.1 OAP *C-O Diagrams*

In what follows we model this contract with *C-O Diagrams* taking into account the information provided in Table 1. In these diagrams we use a_n to denote the action corresponding to clause number n and the reparation for **Clause 8** is called R_1. We also use t_1 to denote the real-time constraint of **Clause 2** and **Clause 3**, t_2 to denote the real-time constraint of **Clause 4**, t_3 to denote the real-time constraint of **Clause 5**, t_4 to denote the real-time constraint of **Clause 6** and **Clause 7**, t_5 to denote the real-time constraint of **Clause 8**, and t_6 to denote the real-time constraint of **Clause 9**. The condition of placing the highest bid is denoted as g_1.

In Fig. 14 we show the top-level of the *C-O Diagram* we specify for the process, called *Online_Auctioning*, starting the sequence from the permission specified in **Clause 1**, that has been called *Auction_Item*. We have grouped the rest of the clauses in Table 1 (except the ones corresponding to the reparation) into three more general clauses with a sequence relationship between them: *Check_Item* (**Clause 2** and **Clause 3**), *Auction_Process* (**Clause 4** and **Clause 5**), and *Payment_Shipment* (**Clause 6**, **Clause 7** and **Clause 8**). These general clauses cover the different phases that we have identified in the *Online Auctioning Process*.

The decomposition of clause *Check_Item* into subclauses can be seen in Fig. 15, where an *AND-refinement* is used in the decomposition and the real-time constraint t_1 is affecting the whole composition. We have on the left-hand side the specification of the prohibition specified in **Clause 2**, that has been called *Inadequate_Item*, and on the right-hand side the obligation specified in **Clause 3**, that has been called *Valid_Information*.

The decomposition of clause *Auction_Process* into subclauses can be seen in Fig. 16, where a *SEQ-refinement* is used in the decomposition. We have on the left-hand side the specification of the obligation specified in **Clause 4**, that has been called *Publish_Item*, including the real-time constraint t_2, and on the right-hand side the permission specified in **Clause 5**, that has been called *Place_Bid*, including the real-time constraint t_3. We can see in this clause that the repetition structure of *C-O Diagrams* is used to model that the *buyer* is allowed to place multiple bids.

The decomposition of clause *Payment_Shipment* into subclauses can be seen in Fig. 17, where a *SEQ-refinement* is used again in the decomposition and the evaluation of the condition g_1 is taken into account for the application of the clauses. We have on the left-hand side the obligation specified in **Clause 6** and **Clause 7** about the payment, that has been called *Payment_Item*, including the real-time constraint t_4 and 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 specified in **Clause 8**, that has been called *Send_Item*, including the real-time constraint t_5 and a reference to reparation R_1.

Finally, in Fig. 18 we can see the diagram corresponding to reparation R_1. It has been called *Refund_Penalty*, including the real-time constraint t_6, and it is decomposed into two subclauses by means of an *AND-refinement*. The subclause on the left corresponds to the obligation specified in **Clause 9**, that has been

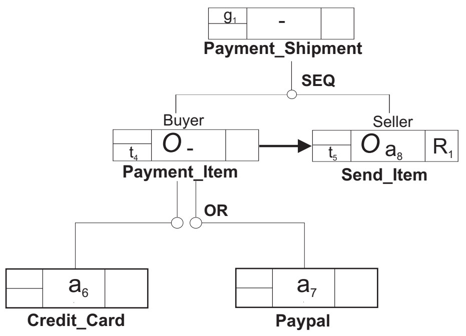

Fig. 17. Decomposition of clause *Payment_Shipment*

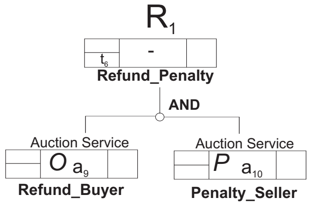

Fig. 18. Reparation of clause *Send_Item*

called *Refund_Buyer*, and the subclause on the right corresponds to the permission specified in **Clause 10**, that has been called *Penalty_Seller*.

5.2 OAP Timed Automata

Next, we are going to show how this contract is translated into a network of timed automata according to the *C-O Diagrams* semantics, where the basic transformation rule **(1)** is applied to obtain an edge executing each one of the actions considered in the contract.

In Fig. 19 we can see the translation into automata corresponding to the first part of the contract. In the initial state n_0 of main automaton A_0 we have that the violation, satisfaction and permission sets are empty ($V0 = \{\}$, $S0 = \{\}$ and $P0 = \{\}$).

As the first norm of the contract is a permission over an action without time restriction nor guard, by applying the transformation rule **(5)** we obtain the state n_1 of A_0, adding the clause *Auction_Item* to its permission set ($V1 = \{\}$, $S1 = \{\}$ and $P1 = \{Auction\_Item\}$), and the edge from n_0 to n_1 performing agent **seller** action a_1. In this edge the clock t_1, used to model the time restriction with the same name, is reset.

Considering that there is a **SEQ-refinement** connecting this first norm with the rest of the contract, by applying rule **(10)** we connect state n_1 of A_0 with state n_2 of A_0 by means of an urgent edge, keeping in n_2 the same sets that in n_1.

As the next part of the contract consists of an **AND-refinement** composing two different norms, the transformation rule **(8)** is applied, defining a new automaton A_1 to execute in parallel with the main automaton A_0. The edge from n_2 to n_3 in A_0 and the edge from n_0 to n_1 in A_1 are used to synchronize by means of the channel m_1. Now we have on the one hand the prohibition over an action considered in automaton A_0, and on the other hand the obligation

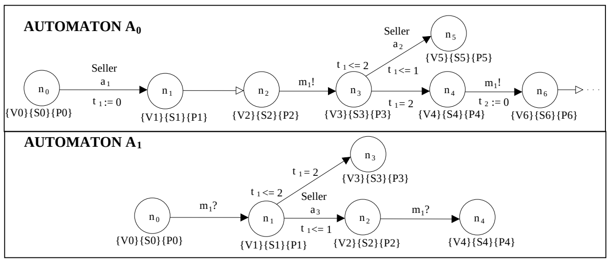

Fig. 19. First part of automaton A_0 and automaton A_1

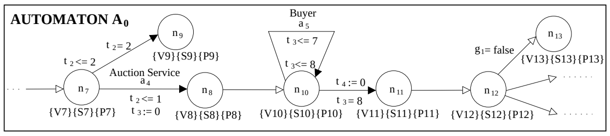

Fig. 20. Second part of automaton A_0

over an action considered in automaton A_1, so we apply in both cases rule **(5)**.

Concerning the prohibition in A_0, in the node n_3 we keep the same sets that in n_2 and, as time restriction t_1 is specified for the norm, the invariant $t_1 \leq 2$ is defined in the node. Next, we have an edge from n_3 to n_5 considering the performance by agent **seller** of the forbidden action a_2 with the guard $t_1 \leq 1$, so in n_5 the clause *Inadequate_Item* is added to the violation set ($V5 = \{Inadequate\_Item\}$, $S5 = \{\}$ and $P5 = \{Auction\_Item\}$), having that n_5 is a final node of the whole automaton as no reparation is defined for the prohibition and the contract is breached. However, we also have an edge from n_3 to n_4 considering that the forbidden action is not executed with the guard $t_1 = 2$, so in n_4 the clause *Inadequate_Item* is added to the satisfaction set ($V4 = \{\}$, $S4 = \{Inadequate\_Item\}$ and $P4 = \{Auction\_Item\}$).

In the automaton A_1, where we consider the obligation, we have that nodes n_0 and n_1 have empty violation, satisfaction and permission sets and, as time restriction t_1 is specified for the norm, the invariant $t_1 \leq 2$ is included in n_1. Next, we have an edge from n_1 to n_2 considering the performance by agent **seller** of the obliged action a_3 with the guard $t_1 \leq 1$, so in n_2 the clause *Valid_Information* is added to the satisfaction set ($V2 = \{\}$, $S2 = \{Valid\_Information\}$ and $P2 = \{\}$). We also have an edge from n_1 to n_3 considering that the obliged action is not executed with the guard $t_1 = 2$, so in n_3 the clause *Valid_Information* is added to the violation set ($V3 = \{Valid\_Information\}$, $S3 = \{\}$ and $P3 = \{\}$), having that n_3 is also a final node where the contract is breached.

Finally, to synchronize both automata again by means of channel m_1, we have the edge from n_4 to n_6 in A_0 (resetting the clock t_2 used to model the time restriction with the same name) and the edge from n_2 to n_4 in A_1. In node n_4 in A_1 we keep the sets of n_2, but in node n_6 in A_0 we add the satisfaction of clause *Valid_Information* to its satisfaction set ($V6 = \{\}$, $S6 = \{Inadequate\_Item, Valid\_Information\}$ and $P6 = \{Auction\_Item\}$).

In Fig. 20 we can see the automaton corresponding

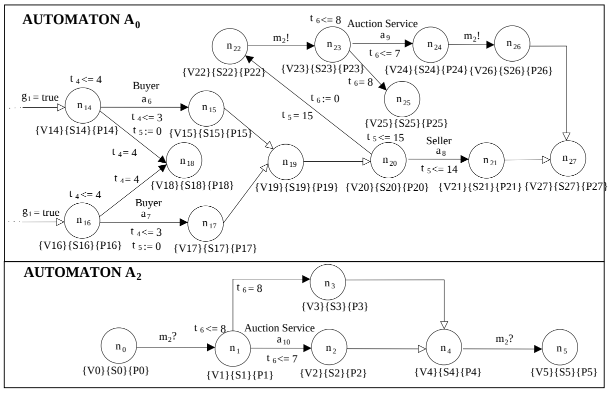

Fig. 21. Third part of automaton A_0 and automaton A_2

to the second part of the contract, that is, starting from clause *Auction_Process*. We first have an obligation over an action, so we apply rule **(5)** again, preserving in node n_7 the same sets that in n_6 and adding also the node n_8 ($V8 = \{\}$, $S8 = \{Inadequate\_Item, Valid\_Information, Publish\_Item\}$ and $P8 = \{Auction\_Item\}$) and the violation node n_9 ($V9 = \{Publish\_Item\}$, $S9 = \{Inadequate\_Item, Valid\_Information\}$ and $P9 = \{Auction\_Item\}$), together with the necessary edges, guards and invariants.

After that, as the next norm in the contract is a permission over an action, we apply rule **(5)**, but taking into account that this time the permission includes a time restriction t_3 and it is applied repetitively. Therefore, after performing the permitted action for the first time, it is added to the permission set of node n_{10} ($V10 = \{\}$, $S10 = \{Inadequate\_Item, Valid\_Information, Publish\_Item\}$ and $P10 = \{Auction\_Item, Place\_Bid\}$).

Finally, we have in the contract an obligation over a compound action in clause *Payment_Item* with time restriction t_4, where the actions are composed by means of an **OR-refinement**, so we now apply the translation rules **(3)** and **(5)**. First, as the guard condition g_1 is considered for the rest of the contract, an urgent edge from n_{12} to n_{13} is added with the guard $g_1 = false$. In n_{13} we keep the violation, satisfaction and permission sets that we have in n_{12} and the automaton ends (but in this case the contract is not breached). The other two urgent edges going out of n_{12} correspond to the choice of performing one of the obliged actions or the other when the guard $g_1 = true$ is satisfied.

In Fig. 21 we can see that when it is chosen the performance of the obliged action a_6, the automaton moves to state n_{14} where we keep the same sets that in n_{12} and the invariant $t_4 \leq 4$ is included. Next, we have an edge from n_{14} to n_{15} considering the performance of a_6 by agent **buyer** with the guard $t_4 \leq 3$ (and resetting the clock t_5 used to model the time restriction with the same name), so in n_{15} the clause *Payment_Item* is added to the satisfaction set ($V15 = \{\}$, $S15 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item\}$ and $P15 = \{Auction\_Item, Place\_Bid\}$). We also have an edge from n_{14} to n_{18} where the action a_6 is not executed with the guard $t_4 = 4$, so in n_{18} the clause *Payment_Item* is added to the violation set and the contract is breached:

- $V18 = \{Payment\_Item\}$
- $S18 = \{Inadequate\_Item, Valid\_Information, Publish\_Item\}$
- $P18 = \{Auction\_Item, Place\_Bid\}$

If it is chosen the performance of the obliged action a_7 instead of a_6, the automaton considered is analogous, but considering this time nodes n_{16} and n_{17} instead of nodes n_{14} and n_{15}. After performing action a_7 we have an urgent edge from n_{15} to n_{19} and after performing a_6 we have another urgent edge from n_{17} to n_{19}. In state n_{19} we keep the same sets that we have in n_{15} and n_{17}.

At this point we have in the contract an obligation over an action with time restriction t_5, so we apply rule **(5)** again. Therefore, we add node n_{21} with the following sets:

- $V21 = \{\}$
- $S21 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item, Send\_Item\}$
- $P21 = \{Auction\_Item, Place\_Bid\}$

We also add the violation node n_{22} with the following sets:

- $V22 = \{Send\_Item\}$
- $S22 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item\}$
- $P22 = \{Auction\_Item, Place\_Bid\}$

However, in this case n_{22} is not a final node of the automaton as reparation R_1 has been defined for the clause *Send_Item* and we apply rule **(6)**. Therefore, considering n_{22} the starting node of the reparation contract, as this contract consists of an **AND-refinement** composing two different norms, the transformation rule **(8)** is applied as in the case of *Check_Item*, defining a new automaton A_2 to execute in parallel with the main automaton A_0. Now we have on the one hand the obligation over an action considered in automaton A_0, and on the other hand the permission over an action considered in automaton A_2. In both cases we take into account the time restriction t_6.

Concerning the obligation in A_0, in the node n_{23} we keep the same sets that in n_{22}, and we add node n_{24} with the following sets:

- $V24 = \{Send\_Item\}$
- $S24 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item, Refund\_Buyer\}$
- $P24 = \{Auction\_Item, Place\_Bid\}$

We also add the violation node n_{25} with the following sets::

- $V25 = \{Send\_Item, Refund\_Buyer\}$
- $S25 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item\}$
- $P25 = \{Auction\_Item, Place\_Bid\}$

In the automaton A_2, where we consider the permission, we have that nodes n_0 and n_1 have empty violation, satisfaction and permission sets. In n_2 the clause *Penalty_Seller* is added to the permission set ($V2 = \{\}$, $S2 = \{\}$ and $P2 = \{Penalty\_Seller\}$), but not in n_3. We also have an edge from n_1 to n_3 considering that the permitted action is not executed with the guard $t_6 = 8$, so in n_3 the clause *Penalty_Seller* is not added and the violation, satisfaction and permission sets remains empty. In node n_4 we keep the sets of the previous node.

After that, we synchronize both automata. In node n_5 in A_2 we keep the sets of n_4, but in node n_{26} in A_0 we modify the sets of n_{24} by removing clause *Send_Item* from its violation set and adding to its permission set the clause *Penalty_Seller* if it has been made effective:

- $V26 = \{\}$
- $S26 = \{Inadequate\_Item, Valid\_Information, Publish\_Item, Payment\_Item, Refund\_Buyer, Penalty\_Seller\}$
- $P26 = \{Auction\_Item, Place\_Bid, Penalty\_Seller?\}$

Finally, according to rule **(8)**, we add an urgent edge from n_{21} to n_{27} and another urgent edge from n_{26} to n_{27}, keeping in the node n_{27} the sets of the previous node, that is, the sets of n_{21} or the sets of n_{26}. This is the main final node of the structure we have built, where we have that the contract has been fulfilled and the process ends.

5.3 OAP Validation and Verification

The automata we have obtained modelling the contractually correct behaviours of the *Online Auctioning Process* are implemented in the UPPAAL tool. At this moment the validation of the contract can be done by means of simulation, where we check that it behaves as expected. For this purpose we use the simulator included in the UPPAAL tool, which can be used in three different ways:

- The system can be run manually, selecting the transitions to be executed at each step.
- The system can run on its own and the transitions to be performed are therefore selected randomly.
- The user can run a trace extracted from the verifier of the UPPAAL tool. This is usually done in the event of a failure when testing a property, in order to analyze the trace that the verifier provides us, which leads to the state at which the property does not hold.

For the verification of the contract we check if the process satisfies some properties of interest by using the verifier of the UPPAAL tool. The query language used for the specification of properties in this verifier is a simplified version of CTL logic described in [14].

Next, we describe the results that have been obtained in the verification of some of these properties:

- First, we want to check if it is possible to reach a state in the process in which the item has been sent to the *buyer*, that is, the obligation $Send\_Item$ has been satisfied. This property is written as follows in the UPPAAL verifier:
$$E <> \quad S[Send\_Item] == true$$
We obtain that this property is **satisfied**.
- Second, we want to check that the item is sent to the *buyer* ($Send\_Item$) only if the payment has been performed before ($Payment\_Item$). This property is written as follows in the UPPAAL verifier:
$$A[] \quad S[Send\_Item] == true$$
$$imply \quad S[Payment\_Item] == true$$
We obtain that this property is **satisfied**.
- Another property we want to check is that after the item has been checked (node n_7 of automaton A_0), if the *auction service* takes more than two days to publish the auction of the item after ($t2 > 2$), the clause $Publish\_Item$ is violated. This property is written as follows in the UPPAAL verifier:
$$A0.n7 \quad and \quad t2 > 2 --> $$
$$V[Publish\_Item] == true$$
We obtain that this property is **satisfied**.
- Finally, we want to check that there exists a maximal path in which none of the main obligations and prohibitions of the contract is violated, that is, the process ends without any violation. This property is written as follows in the UPPAAL verifier:
$$E[] \quad V[Inadequate\_Item] == false$$
$$and \quad V[Valid\_Information] == false$$
$$and \quad V[Publish\_Item] == false$$
$$and \quad V[Payment\_Item] == false$$
$$and \quad V[Send\_Item] == false$$
We obtain that this property is **satisfied**.

An alternative way of verification that can be performed with the UPPAAL verifier is checking that some undesirable behaviours never happen. This is done by means of the verification of properties corresponding to these behaviour, so we expect to obtain that the properties are not satisfied.

Some of these properties that we have verified are the following:

- We do not want the obligation of sending the item to the *buyer* ($Send\_Item$) to be satisfied if the payment has not been performed ($Payment\_Item$). This property is written as follows in the UPPAAL verifier:
$$S[Send\_Item] == true -->$$
$$S[Payment\_Item] == false$$
We obtain that this property is **not satisfied** as expected.
- We do not want the obligation of sending the item to the *buyer* ($Send\_Item$) and the obligation of refunding the payment to the *buyer* ($Refund\_Buyer$) to be satisfied at the same time, as the refund has to be done only if the item is not sent. This property is written as follows in the UPPAAL verifier:

$$E <> \quad S[Send\_Item] == true$$
$$and \quad S[Refund\_Buyer] == true$$

We obtain that this property is **not satisfied** as expected.

- We do not want to have that for all the states the obligation of paying the item ($Payment\_Item$) is not satisfied or the obligation of refunding the payment ($Refund\_Buyer$) is not satisfied, as we expect to have a state where both obligations are satisfied in the case of paying the item but not getting it. This property is written as follows in the UPPAAL verifier:

$$A[] \quad S[Payment\_Item] == false$$
$$or \quad S[Refund\_Buyer] == false$$

We obtain that this property is **not satisfied** as expected.

- Another interesting property that at first glance might appear to be satisfiable is that when the obligation of paying the item is satisfied ($Payment\_Item$), either the obligation of sending the item ($Send\_Item$) or the obligation of refunding the payment ($Refund\_Buyer$) must be eventually satisfied. This property is written as follows in the UPPAAL verifier:

$$S[Payment\_Item] == true -- >$$
$$S[Send\_Item] == true$$
$$or \quad S[Refund\_Buyer] == true$$

We obtain that this property is **not satisfied**. To see what is happening, we run the trace provided by the verifier leading to the state at which the property does not hold. In this trace we can see that automaton A_0 ends in node n_{25}. In this node both norms, the obligation of sending the item and the obligation of refunding the payment, have been violated. Therefore, it is correct the result we have obtained for this property as it is possible to reach a state where both norms are violated and the contract is breached.

6 Case Study: Adaptive Cruise Control (ACC)

This case study is about the requirements engineering process for the correct operation of an *Adaptive Cruise Control* system. The aim of this system is the automatic control of the speed of a car, in some cases taking into account the distance to the vehicle ahead. Sensors are used to measure this distance.

The description of the system operation that we consider is the following: the system starts working when it is inactive and the *driver* presses a button to activate the system. It can be activated with the current speed or with a previously stored speed. After that, on the one hand the *driver* is forbidden to deactivate the system for **10 milliseconds**, just in order to allow the system to fully perform all the security checks before allowing deactivation. On the other hand the *system* has to validate within **5 milliseconds** if the speed is in the permissible range. If the validation fails the *system* has **10 milliseconds** to notify that the speed is invalid, ending the activation process, but if the validation does not fail the *system* has to continue validating within **5 milliseconds** that no vehicle is within the minimum safety distance. Again, if the validation fails the *system* has **10 milliseconds** to notify that the distance is invalid, ending the activation process, but if the validation does not fail the *system* has to finish the activation process.

At this moment there are two possibilities. If no vehicle ahead is detected, the *system* just displays within **5 milliseconds** the activation speed that is going to be followed. However, if a vehicle within the minimum safety distance is detected, the *system* has **15 milliseconds** to reduce the engine torque and actuate the brakes in order to adapt the cruise speed to the vehicle ahead speed, and after that the *system* displays the speed adopted. Finally, after displaying the speed, the *driver* is allowed to deactivate the system at any moment.

In Table 2 we show a list of the obligations, permissions and prohibitions that can be extracted from the description of the system operation. In this table we can see that there are seven clauses specifying real-time constraints. In **Clause 4** it is specified that after activation (**Clause 1** or **Clause 2**), the *system* has 5 milliseconds to validate the speed. In **Clause 5** we have that the *system* has 10 milliseconds to notify invalid speed after speed validation (**Clause 4**). In **Clause 6** we have that the *system* has 5 milliseconds to validate the distance after speed validation (**Clause 4**). In **Clause 7** we have that the *system* has 10 milliseconds to notify invalid distance after distance validation (**Clause 6**). **Clause 8** and **Clause 9** specify that the system has 15 milliseconds to actuate brakes and reduce engine torque after distance validation (**Clause 6**). Finally, **Clause 9** specifies that the system has 5 milliseconds to display the adopted speed.

6.1 ACC *C-O Diagrams*

What follows we model this contract with *C-O Diagrams* taking into account the information provided in Table 2. In these diagrams we use a_n to denote the action corresponding to clause number n. We also use t_1 to denote the real-time constraint of **Clause 3**, t_2 to denote the real-time constraint of **Clause 5**, t_3 to denote the real-time constraint of **Clause 6**, t_4 to denote the real-time constraint of **Clause 7**, t_5 to denote the real-time constraint of **Clause 8**, t_6 to denote the real-time constraint of **Clause 9**, t_7 to denote the real-time constraint of **Clause 10**, and t_8 to denote the real-time constraint of **Clause 11**. The condition **valid speed** is denoted as g_1, the condition **valid distance** is denoted as g_2, and the condition **vehicle ahead** is denoted as g_3,.

In Fig. 22 we show the top-level of the *C-O*

Clause	Agent	Modality	Action	Condition	Time
1	*Driver*	*Permission*	Can active the system with the current speed	$\emptyset$	$\emptyset$
2	*Driver*	*Permission*	Can active the system with a previously stored speed	$\emptyset$	$\emptyset$
3	*Driver*	*Prohibition*	Deactivates the system	$\emptyset$	Ten milliseconds after activation (t_1)
4	*Driver*	*Permission*	Deactivates the system	$\emptyset$	$\emptyset$
5	*System*	*Obligation*	Validates the car speed	$\emptyset$	Five milisecons after activation (t_2)
6	*System*	*Obligation*	Notifies invalid speed	Invalid speed ($\neg g1$)	Ten milliseconds after speed validation (t_3)
7	*System*	*Obligation*	Validates distance	Valid speed ($g1$)	Five milliseconds after speed validation (t_4)
8	*System*	*Obligation*	Notifies invalid distance	$g1 \land$ Invalid distance ($\neg g2$)	Ten milliseconds after distance validation (t_5)
9	*System*	*Obligation*	Reduces engine torque	$g1 \land g2 \land$ Vehicle ahead ($g3$)	Fifteen milliseconds after distance validation (t_6)
10	*System*	*Obligation*	Actuates brakes	$g1 \land g2 \land$ Vehicle ahead ($g3$)	Fifteen milliseconds after distance validation (t_7)
11	*System*	*Obligation*	Displays speed	$g1 \land$ Valid distance ($g2$)	Five milliseconds after setting speed (t_8)

TABLE 2
Norms of the *Adaptive Cruise Control* contract

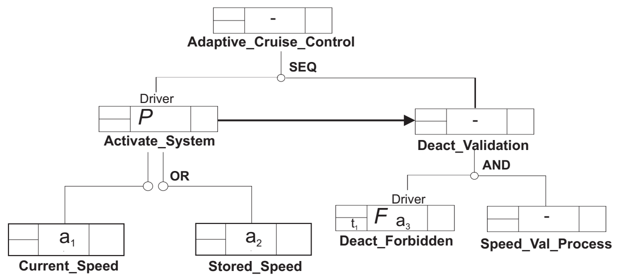

Fig. 22. Top-level of the *Adaptive Cruise Control*

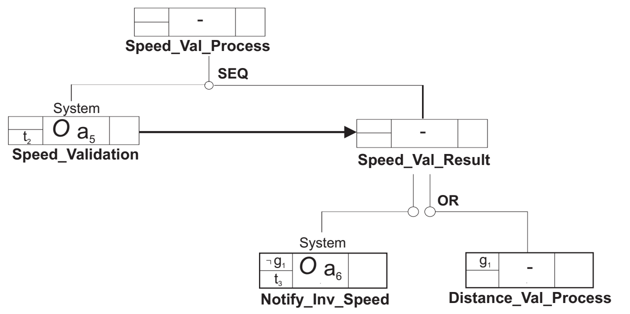

Fig. 23. Decomposition of clause *Speed_Val_Process*

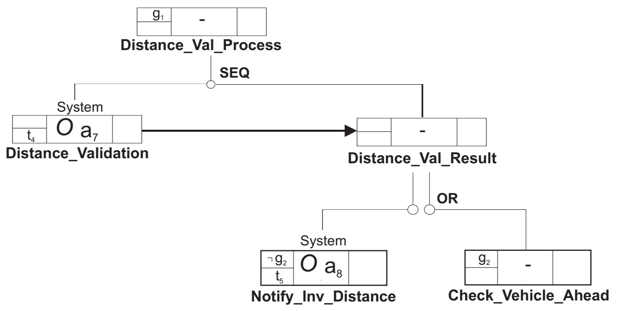

Fig. 24. Decomposition of clause *Distance_Val_Process*

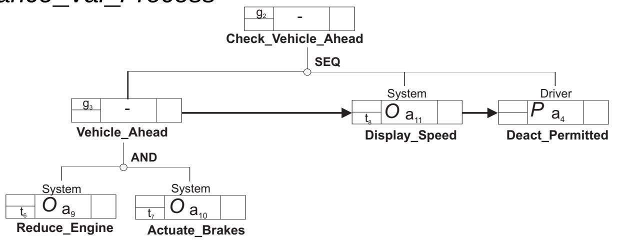

Fig. 25. Decomposition of clause *Check_Vehicle_Ahead*

Diagram we specify for the process, called *Adaptive_Cruise_Control*, starting the sequence from the permissions specified in **Clause 1** and **Clause 2**, that have been called *Activate_System*, and composing the actions of activating the system with current speed or with a previously stored speed by means of an *OR-refinement*. After that, we have an *AND-refinement* considering the prohibition of deactivating the system specified in **Clause 3**, called *Deact_Forbidden* and a general clause called *Speed_Val_Process* which decomposition is shown next.

The decomposition of clause *Speed_Val_Process* into subclauses can be seen in Fig. 23, where a *SEQ-refinement* is used in the decomposition. We have on the left-hand side the specification of the obligation specified in **Clause 5**, that has been called *Speed_Validation*, and on the right-hand side we have an *OR-refinement* where we have that the obligation specified in **Clause 6**, called *Notify_Inv_Speed*, is applied if the speed is not valid, otherwise a general clause called *Distance_Val_Process* is applied.

The decomposition of clause *Distance_Val_Process* into subclauses is shown in Fig. 24, where a *SEQ-refinement* is used in the decomposition. We have on the left-hand side the specification of the obligation specified in **Clause 7**, that has been called *Distance_Validation*, and on the right-hand side we have an *OR-refinement* where we have that the obligation specified in **Clause 8**, called *Notify_Inv_Distance*, is applied if the distance is not valid, otherwise a general clause called *Check_Vehicle_Ahead* is applied.

Finally, in Fig. 25 it is shown the decomposition of clause *Check_Vehicle_Ahead*, where a *SEQ-refinement* is used in the decomposition. In this case the sequence starts with an *AND-refinement* considering the obligation specified in **Clause 9**, that has been called

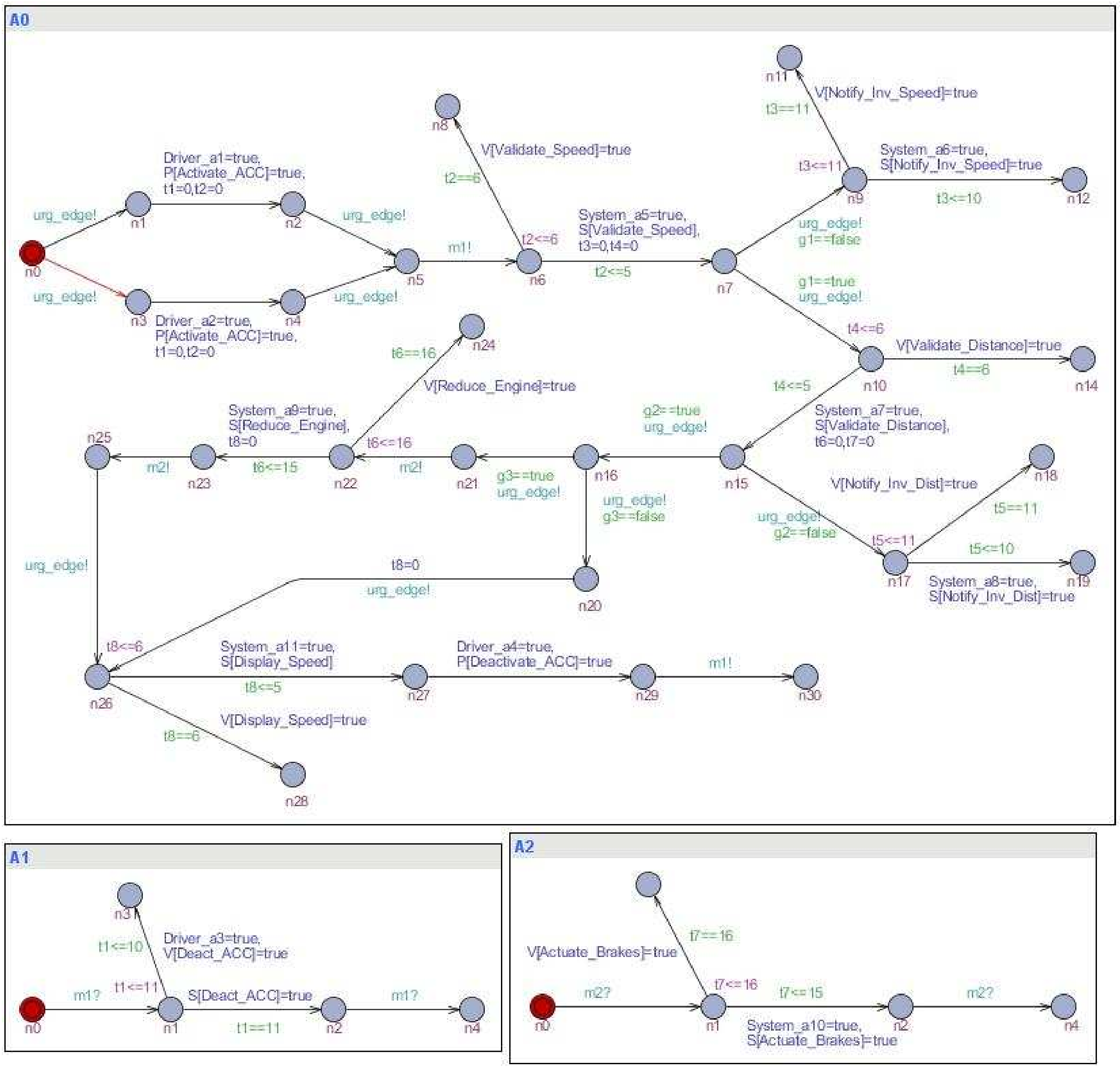

Fig. 26. Implementation of the ACC network of timed automata in UPPAAL

Reduce_Engine, and the obligation specified in **Clause 10**, that has been called *Actuate_Brakes*. These two obligations are applied only if a vehicle ahead is detected. After that, we have the specification of the obligation specified in **Clause 11**, that has been called *Display_Speed*, and at the end of the sequence we have the specification of the permission specified in **Clause 4**, that has been called *Deact_Permitted*.

6.2 ACC Validation and Verification

Once again, we obtain the network of timed automata corresponding to this contract by applying the transformation rules of the *C-O Diagrams* semantics and we implement these automata in the UPPAAL tool for the validation and verification of the contract, as in the OAP case study. The automata implemented in this case in the UPPAAL tool can be seen in Fig. 26.

As in the OAP case study, the validation of the contract can be done by means of simulation in the UPPAAL Simulator and the verification of the contract consists of checking if the process satisfies some properties of interest in the UPPAAL verifier.

We describe now the results that have been obtained in the verification of some of these properties:

- First, we want to check if it is possible to reach a state in the process in which the system has been eventually activated and the speed adopted is displayed, that is, the obligation $Display\_Speed$ has been satisfied. This property is written as follows in the UPPAAL verifier:
  $$E <> \quad S[Display\_Speed] == true$$
  We obtain that this property is **satisfied**.
- Second, we want to check that if after the system has validated the speed (node n_10 of automaton A_0) it takes more than five milliseconds to validate the distance ($t4 > 5$), the clause $Validate\_Distance$ is violated. This property is written as follows in the UPPAAL verifier:

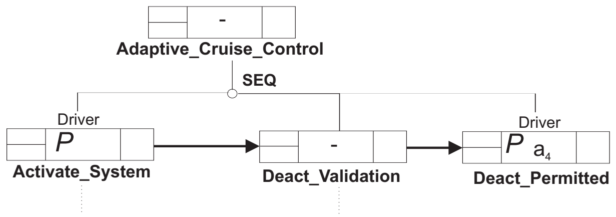

Fig. 27. Modification of the *Adaptive Cruise Control*

$$A0.n10 \quad and \quad t4 > 5 --> V[Validate\_Distance] == true$$

We obtain that this property is **satisfied**.

- Another property we want to check is if there exists a maximal path in which none of the main obligations and prohibitions of the contract is violated, that is, the process ends without any violation. This property is written as follows in the UPPAAL verifier:

$$\begin{gathered} E[] \quad V[Deact\_ACC] == false \\ and \quad V[Validate\_Speed] == false \\ and \quad V[Notify\_Inv\_Speed] == false \\ and \quad V[Validate\_Distance] == false \\ and \quad V[Notify\_Inv\_Dist] == false \\ and \quad V[Reduce\_Engine] == false \\ and \quad V[Actuate\_Brakes] == false \\ and \quad V[Display\_Speed] == false \end{gathered}$$

  We obtain that this property is **satisfied**.
- Finally, we want to check that it is not allowed by the contract specification reaching a state where both, the permission and the prohibition of deactivating the system are considered at the same time. This property is written as follows in the UPPAAL verifier:

$$A[] \quad not(A0.n27 \quad and \quad A1.n1)$$

  We obtain that this property is **not satisfied**.

At this point we have found out that one of the desired properties is not satisfied by the contract specification. Therefore, we have to go back to the *C-O Diagrams* modelling the contract in order to solve this problem. In this case the problem is that the permission of deactivating the system can be enacted after all the activation process before the prohibition of deactivation has expired.

The problem can be solved by modifying the contract in such a way that the permission of deactivating the system is only enacted after the prohibition of doing that has expired. The modification proposed is shown in Fig. 27, where we have that the $Deact\_Permitted$ clause is now applied only after the $Deact\_Validation$ process has finished (therefore the $Deact\_Permitted$ clause is not a subclause of this process anymore). After this modification of the contract specification, the transformation into a network of timed automata and the implementation into UPPAAL have been done again, and it is obtained now that all the properties are satisfied.

7 RELATED WORK

The use of deontic logic for reasoning about contracts is widely spread in the literature since it was proposed in [16] for modelling communication processes. In [17] 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 [18] 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, being based again on deontic logic. The work by Lomuscio et al. provides another methodology to check whether service compositions are compliant with e-contracts. In [19] they present an approach 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, whereas in [20] they consider a service composition in OWL-S and check with MCMAS if the composition fulfils some contract properties written in a formal language based on epistemic and deontic logic. In both works we have that the specification of real-time constraints is not allowed because they are not supported by MCMAS and the deontic norms are restricted to obligations.

The approach followed by *C-O Diagrams* is inspired by the formal language $\mathcal{CL}$ [8]. In this language a contract is also expressed as a composition of obligations, permissions and prohibitions over actions, and the way of specifying reparations is the same that in our model. However, $\mathcal{CL}$ does not support neither the specification of agents nor timing constraints natively, so they have to be encoded in the definition of the actions. In [21] Solaiman et al. show how relevant parts of contracts can be described by means of Finite State Machines (FSMs), using these FSMs to check the correctness of the contract specification, detecting any undesirable ambiguity, whereas our approach seeks to check if the contract specified satisfies some properties of interest by using the verifier of the UPPAAL tool. In [22] Desai et al. also automate reasoning about the correctness the contract specification, but in this case representing contracts formally as a set of commitments.

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 [23] 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 [24] Krishna et al. proposed another meta-model based on entity-relationship diagrams that they use 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. Another approach can be found in [25], where Rouached et al. propose a contract layered model for modelling and monitoring e-contracts. It consists of a business entities layer, a business actions layer, and a business rules layer. These three layers specify the parties, the actions and the clauses of the contract respectively, including the conditions under which these clauses are executed. However, real-time restrictions are not included and the specification of the clauses follows an operational approach, not a deontic approach. Finally, in [26] Heckel and Lohmann propose to visualize contracts by graph transformation rules over UML data models, but this approach is closer to implementation than ours and it is focused on testing.

8 Conclusions

In this work we have developed a formal semantics for *C-O Diagrams* based on timed automata extended with an ordering of states and edges in order to represent the different deontic modalities. We have also seen how these automata can be implemented in UPPAAL in order to model-check the contract specification, and we have applied it to two different case studies.

We plan to work on several other case studies from different application domains to further explore the applicability of our approach. This includes legal contracts, regulatory texts, and electronic contracts (e.g., contracts in the context of SOA). Finally, an interesting research direction is to use GF (the *Grammatical Framework*) [27] to relate *C-O Diagrams* with natural language. This might be done by means of using controlled natural languages as an intermediate language and by encoding *C-O Diagrams* into GF as has been recently done for the language CL [28].

Acknowledgments

Partially supported by the Spanish government (co financed by FEDER founds) with the project TIN2009-14312-C02-02 and the JCCLM regional project PEII09-0232-7745. The first author is supported by the European Social Fund and the JCCLM.

References

[1] B. Meyer, "Design by Contract," Interactive Software Engineering Inc., Tech. Rep. TR-EI-12/CO, 1986.

[2] J. Hatcliff, G. Leavens, k. Leino, P. Mller, and M. Parkinson, "Behavioral Interface Specification Languages," School of EECS, University of Central Florida, Tech. Rep. CS-TR-09-01, 2009.

[3] "ebXML: Electronic Business using eXtensible Markup Language," www.ebxml.org.

[4] "WSLA: Web Service Level Agreements," www.research.ibm.com/wsla/.

[5] "Web Services Agreement Specification (WS-Agreement)," https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/7.

[6] J. C. Okika and A. P. Ravn, "Classification of soa contract specification languages," in *2008 IEEE International Conference on Web Services (ICWS'08)*. IEEE Computer Society, 2008, pp. 433–440.

[7] P. McNamara, "Deontic Logic," in *Gabbay, D.M., Woods, J., eds.: Handbook of the History of Logic*. North-Holland Publishing, 2006, vol. 7, pp. 197–289.

[8] C. Prisacariu and G. Schneider, "CL: An Action-based Logic for Reasoning about Contracts," in *16th Workshop on Logic, Language, Information and Computation (WOLLIC'09)*, ser. LNCS, vol. 5514. Springer, June 2009, pp. 335–349.

[9] E. Martínez, Díaz, G., M. E. Cambronero, and G. Schneider, "A Model for Visual Specification of e-Contracts," in *The 7th IEEE International Conference on Services Computing (IEEE SCC'10)*, 2010, pp. 1–8.

[10] E. Martínez, Díaz, G., and M. E. Cambronero, "Contractually Compliant Service Compositions," *ICSOC 2011 - The Ninth International Conference on Service Oriented Computing*, pp. 636–644, 2011.

[11] K. G. Larsen, Z. Pettersson, and Y. Wang, "UPPAAL in a Nutshell," *STTT: International Journal on Software Tools for Technlogy Transfer*, vol. 1, no. 1–2, pp. 134–152, 1997.

[12] R. Alur and D. Dill, "Automata For Modeling Real-Time Systems." in *ICALP*, 1990, pp. 322–335.

[13] ——, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126(2), pp. 183–235, 1994.

[14] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," *Formal Methods for the Design of Real-Time Systems*, no. 3185, pp. 200–236, 2004.

[15] G. Decker, "Design and Analysis of Process Choreographies," Ph.D. dissertation, Hasso Plattner Institute, Uiversity of Postdam, 2009.

[16] F. Dignum and H. Weigand, "Modelling Communication between Cooperative Systems," *Proceedings of Advanced Information Systems Engineering (CAISE'95)*, pp. 140–153, 1995.

[17] O. Marjanovic and Z. Milosevic, "Towards formal modeling of e-Contracts," *Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference*, pp. 59–68, 2001.

[18] G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," *Proceedings of the 10th IEEE Conference on Enterprise Distributed Object Computing*, pp. 221–232, 2006.

[19] A. Lomuscio, H. Qu, and M. Solanki, "Towards verifying contract regulated service composition," *Proceedings of IEEE International Conference on Web Services (ICWS 2008)*, pp. 254–261, 2008.

[20] ——, "Towards verifying compliance in agent-based web service compositions," *Proceedings of 7th on Autonomous Agents and Multiagent Systems*, pp. 265–272, 2008.

[21] E. Solaiman, C. Molina-Jimenez, and S. Shrivastava, "Model Checking Correctness Properties of Electronic Contracts," *Proceedings of International Conference on Service Oriented Computing (ICSOC03)*, pp. 303–318, 2003.

[22] N. Desai, N. C. Narendra, and M. P. Singh, "Checking correctness of business contracts via commitments," *Proceedings of 7th on Autonomous Agents and Multiagent Systems*, pp. 787–794, 2008.

[23] D. Chiu, S. Cheung, and S. Till, "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, 2003.

[24] P. Krishna, K. Karlapalem, and A. Dani, "From Contract to E-Contracts: Modeling and Enactment," *Information Technology and Management*, vol. 6, no. 4, pp. 363–387, 2005.

[25] M. Rouached, O. Perrin, and C. Godart, "A Contract Layered Architecture for Regulating Cross-Organisational Business Processes," *Proceedings of Third International Conference on Business Process Management*, pp. 410–415, 2005.

[26] R. Heckel and R. Lohmann, "Towards verifying compliance in agent-based web service compositions," *Proceedings of International Workshop on Test and Analysis of Component Based Systems*, pp. 145–156, 2004.

[27] A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars*. Stanford: CSLI Publications, 2011, iSBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[28] S. M. Montazeri, N. Roy, and G. Schneider, "From Contracts in Structured English to CL Specifications," in *FLACOS'11*, ser. EPTCS, vol. 68, 2011, pp. 55–69. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.68.6

APPENDIX A
C-O Diagrams TRANSFORMATION RULES

In this Appendix we provide the formal definition of the *C-O Diagrams* transformation rules that have been informally described in Section 3.

Definition 8: (*C-O Diagrams* Transformation Rules)

(1) An **atomic action** in a *C-O Diagram*, that is, $(\epsilon, name, \epsilon, \epsilon, a, \epsilon)$ corresponds to the timed automaton $A = (N_A, n_{0_A}, E_A, I_A)$, where:
- $N_A = \{a_{init}, a_{end}\}$.
- $n_{0_A} = a_{init}$.
- $E_A = \{a_{init} \xrightarrow{a} a_{end}\}$.
- $I_A = \emptyset$.

(2) A **compound action** in a *C-O Diagram* where an **AND-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, And\, C_2\, And\, \ldots\, And\, C_n, \epsilon)$ corresponds to the cartesian product of the automata corresponding to each one of the subcontracts. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton AND corresponds to the cartesian product of these automata, that is, $AND = A \times B \times \ldots \times Z$.

(3) A **compound action** in a *C-O Diagram* where an **OR-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, Or\, C_2\, Or\, \ldots\, Or\, C_n, \epsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts is considered as an alternative. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton OR preserves the structure of the automata we are composing but adding a new initial node OR_{init} and connecting this node by means of urgent edges performing no action to the initial nodes of $A, B, \ldots, Z$ $(A_{init}, B_{init}, \ldots, Z_{init})$. It is also added a new ending node OR_{end} and urgent edges performing no action from the ending nodes of $A, B, \ldots, Z$ $(A_{end}, B_{end}, \ldots, Z_{end})$ to this new ending node. Let $A = (N_A, n_{0_A}, E_A, I_A), B = (N_\mathcal{B}, n_{0_B}, E_B, I_B), \ldots, Z = (N_Z, n_{0_Z}, E_Z, I_Z)$. The resulting automaton is therefore $OR = (N_{OR}, n_{0_{OR}}, E_{OR}, I_{OR})$, where:
- $N_{OR} = N_A \cup N_B \cup \ldots \cup N_Z \cup \{OR_{init}, OR_{end}\}$.
- $n_{0_{OR}} = OR_{init}$.
- $E_{OR} = E_A \cup E_B \cup \ldots \cup E_Z \cup \{OR_{init} \longrightarrow_u A_{init}, OR_{init} \longrightarrow_u B_{init}, \ldots, OR_{init} \longrightarrow_u Z_{init}\} \cup \{A_{end} \longrightarrow_u OR_{end}, B_{end} \longrightarrow_u OR_{end}, \ldots, Z_{end} \longrightarrow_u OR_{end}\}$.
- $I_{OR} = I_A \cup I_B \cup \ldots \cup I_Z$.

(4) A **compound action** in a *C-O Diagram* where a **SEQ-refinement** is used to compose actions, that is, $(\epsilon, name, \epsilon, \epsilon, C_1\, Seq\, C_2\, Seq \ldots Seq\, C_n, \epsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts are connected in sequence. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton SEQ preserves the structure of the automata we are composing, adding no extra nodes. We only connect with an urgent edge performing no action the ending node of each automaton in the sequence $(A_{end}, B_{end}, \ldots, Y_{end})$ with the initial node of the next automaton in the sequence $(B_{init}, C_{init}, \ldots, Z_{init})$. This rule is not applied in the cases of A_{init} (as there is not previous ending node to connect) and Z_{end} (as there is not following initial node to connect). Let $A = (N_A, n_{0_A}, E_A, I_A), \mathcal{B} = (N_B, n_{0_B}, E_B, I_B), \ldots, Z = (N_Z, n_{0_Z}, E_Z, I_Z)$. The resulting automaton is therefore $SEQ = (N_{SEQ}, n_{0_{SEQ}}, E_{SEQ}, I_{SEQ})$, where:
- $N_{SEQ} = N_A \cup N_B \cup \ldots \cup N_Z$.
- $n_{0_{SEQ}} = A_{init}$.
- $E_{SEQ} = E_A \cup E_B \cup \ldots \cup E_Z \cup \{A_{end} \longrightarrow_u B_{init}, B_{end} \longrightarrow_u C_{init}, \ldots, Y_{end} \longrightarrow_u Z_{init}\}$.
- $I_{SEQ} = I_A \cup I_B \cup \ldots \cup I_Z$.

(5) The application of an **obligation**, a **permission** or a **prohibition** over an action in a *C-O Diagram*, i.e., $(agent, name, g, tr, O/P/F(C), R)$ corresponds to an automaton where the obligation/prohibition of performing the action specified in the subcontract C can be skipped, fulfilled or violated, whereas the permission of performing the action can be skipped, made effective or not made effective. Let us consider $A = (N_A, n_{0_A}, E_A, I_A)$ the automaton corresponding to C, being A_{init} the initial node and A_{end} the ending node. The resulting automaton $D(A)$, where $D \in \{O, P, F\}$, preserves the structure of the automaton A but adding a new ending node A_{time} including the obligation over the action in its violation set, the prohibition over the action in its satisfaction set or nothing if a permission over the action is considered. If guard condition $g \neq \epsilon$, we add another ending node A_{skip} where the violation, satisfaction and permission sets are not modified. We also include the obligation over the action in the satisfaction set of A_{end}, the prohibition over the action in the violation set of A_{end}, or the permission over the action in the permission set of A_{end}. An invariant $x \leq t2 + 1$ is added to each node of A except A_{end} and each edge

- $N_{D(A)} = N_A \cup \{A_{time}, A_{skip}\}$.
- $n_{0_{D(A)}} = A_{init}$.
- $E_{D(A)} = \{A_{init} \xrightarrow{\neg g}_u A_{skip}\} \cup \begin{cases} n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' \neq A_{end}, & \\ n \xrightarrow{g_1, agent(a), t_{name}} n' \,|\, n \xrightarrow{a} n' \in E_A & \\ \text{and } n' = A_{end}, n \xrightarrow{g_2} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = O \\ n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' \neq A_{end}, & \\ n \xrightarrow{g_1, agent(a), t_{name}} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' = A_{end}, & \\ n \xrightarrow{g_2} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = P \\ n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A, & \\ n \xrightarrow{g_2, t_{name}} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = F \end{cases}$
- $I_{D(A)} = I_A \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_A - \{A_{end}\}\}$.

Result of transformation rule (5)

performing one of the actions in this automaton is guarded with $(x \geq t1)\, and\, (x \leq t2)$ and action performed by $agent$. New edges guarded with $x = t2 + 1$ and no action performed are added from each node of A except A_{end} to the new node A_{time} and, if guard condition $g \neq \epsilon$, an urgent edge from A_{init} to A_{skip} is also added guarded with the guard condition of the clause negated ($\neg g$). Finally, if $t_{name} \in \mathcal{C}$, all the edges reaching A_{end} reset t_{name} in the cases of obligation and permission, whereas all the edges reaching A_{time} reset t_{name} in the case of prohibition. Considering the more complex case, where $g \neq \epsilon$ and $t_{name} \in \mathcal{C}$, and having that $g_1 \equiv (x \geq t1)\, and\, (x \leq t2)$ and $g_2 \equiv x = t2 + 1$, the resulting automaton is therefore $D(A) = (N_{D(A)}, n_{0_{D(A)}}, E_{D(A)}, I_{D(A)})$, which elements are described in next page.

(6) An **obligation** or **prohibition** in a *C-O Diagram* specifying a contract **reparation** $R \neq \epsilon$ corresponds to the obligation automaton $O(A)$ or the prohibition automaton $F(A)$ together with the reparation automaton R, considering the node with $name$ in its violation (A_{vio}) set as the initial node of the reparation automaton (R_{init}). In the ending node of the reparation automaton (R_{end}) $name$ is removed from the violation set, as the violation has been repaired. In this node we also have that the satisfaction set and the permission set are different from the ones we have in the initial node of the reparation because we have to include in the satisfaction set all the obligations and prohibitions satisfied in the reparation contract, and in the permission set all the permissions that have been made effective in the reparation contract. Let us consider $D(A) = (N_{D(A)}, n_{0_{D(A)}}, E_{D(A)}, I_{D(A)})$, where $D \in \{O, F\}$, and $R = (N_R, n_{0_R}, E_R, I_R)$. The resulting automaton is therefore $D(A)_R = (N_{D(A)_R}, n_{0_{D(A)_R}}, E_{D(A)_R}, I_{D(A)_R})$, where:

- $N_{D(A)_R} = N_{D(A)} \cup N_R - \{R_{init}\}$.
- $n_{0_{D(A)_R}} = A_{init}$.
- $E_{D(A)_R} = E_{D(A)} \cup \{n \xrightarrow[s]{g,a,r} n' \,|\, n \xrightarrow[s]{g,a,r} n' \in E_R$ and $n \neq R_{init}\} \cup$ $\{A_{vio} \xrightarrow[s]{g,a,r} n' \,|\, n \xrightarrow[s]{g,a,r} n' \in E_R$ and $n = R_{init}\}$.
- $I_{D(A)_R} = I_{D(A)} - \{I(A_{vio})\} \cup \{I(n) \,|\, n \in N_R - \{R_{init}\}\} \cup \{I(A_{vio}) \equiv I(R_{init})\}$.

(7) Let $D(A)_R = (N_{D(A)_R}, n_{0_{D(A)_R}}, E_{D(A)_R}, I_{D(A)_R})$, where $D \in \{O, P, F\}$, be the automaton corresponding to an **obligation**, a **prohibition** or a **permission** in a *C-O Diagram*, specifying a **reparation** $R \neq \epsilon$ in the two first cases. The corresponding automaton with only one ending node, that we call A_{final} and preserves the violation, satisfaction and permission sets of the previous node, is $D(A)'_R = (N_{D(A)'_R}, n_{0_{D(A)'_R}}, E_{D(A)'_R}, I_{D(A)'_R})$, where:

- $N_{D(A)'_R} = N_{D(A)_R} \cup \{A_{final}\}$.
- $n_{0_{D(A)'_R}} = n_{0_{D(A)_R}}$.
- $E_{D(A)'_R} = E_{D(A)_R} \cup \{A_{skip} \longrightarrow_u A_{final}\} \cup \begin{cases} A_{end} \longrightarrow_u A_{final}, R_{end} \longrightarrow_u A_{final} & \text{if } D = O \\ A_{end} \longrightarrow_u A_{final}, A_{time} \longrightarrow_u A_{final} & \text{if } D = P \\ A_{time} \longrightarrow_u A_{final}, R_{end} \longrightarrow_u A_{final} & \text{if } D = F \end{cases}$
- $I_{D(A)'_R} = I_{D(A)_R}$.

(8) If several norms are composed by an **AND-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, And\, C_2\, And \ldots And\, C_n, \epsilon)$, their composition corresponds to a network of automata in which we consider all the norms we are composing in parallel. Let us consider $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ the automata corresponding to the norms we are composing. The resulting network of automata preserves the structure of the automata we are composing, adding to each one of them the additional nodes and edges necessary for synchronization (these nodes are called C_{init} and C_{final} in the first automaton, C_{isyn} and $C_{isyn'}, i = 1, \ldots, n-1$ in the other automata). Before its initial node, each automaton synchronizes with the other automata and it synchronizes again after its final node by means of urgent channels $(m_1, m_2, \ldots, m_{n-1})$. In the first automaton we add another node C_{skip} if guard condition of the parent clause $g \neq \epsilon$ and an urgent edge from C_{init} to this new node guarded with the guard condition negated ($\neg g$). In the final node of the first automaton the violation, satisfaction and permission sets are the union of the sets resulting in each one of the automata

- $N_{\mathcal{C}*_i} = N_{\mathcal{C}_i} \cup \begin{cases} C_{init}, C_{final}, C_{skip} & \text{if } i = 1 \\ C_{isyn}, C_{isyn'}, C_{i-1syn}, C_{i-1syn'} & \text{if } i = 2, \ldots, n-1 \\ C_{i-1syn}, C_{i-1syn'} & \text{if } i = n \end{cases}$
- $n_{0_{\mathcal{C}*_i}} = \begin{cases} C_{init} & \text{if } i = 1 \\ C_{i-1syn}, C_{i-1syn'} & \text{if } i = 2, \ldots, n \end{cases}$
- $E_{\mathcal{C}*_i} = E_{\mathcal{C}_i} \cup \begin{cases} C_{init} \xrightarrow{\neg g}_u C_{skip}, C_{init} \xrightarrow{m_i!} C_{iinit}, & \\ C_{ifinal} \xrightarrow{m_i!} C_{final} & \text{if } i = 1 \\ C_{i-1syn} \xrightarrow{m_{i-1}?} C_{isyn}, C_{isyn'} \xrightarrow{m_{i-1}?} C_{i-1syn'}, & \\ C_{isyn} \xrightarrow{m_i!} C_{iinit}, C_{ifinal} \xrightarrow{m_i!} C_{isyn'} & \text{if } i = 2, \ldots, n-1 \\ C_{i-1syn} \xrightarrow{m_{i-1}?} C_{iinit}, C_{ifinal} \xrightarrow{m_{i-1}?} C_{final} & \text{if } i = n \end{cases}$
- $I_{\mathcal{C}*_i} = I_{\mathcal{C}_i} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_i} - \{C_{ifinal}\}\}$.

Result of transformation rule (8)

running in parallel, so we have that Vfinal = $V1 \cup V2 \cup \ldots \cup Vn$, Sfinal = $S1 \cup S2 \cup \ldots \cup Sn$ and Pfinal = $P1 \cup P2 \cup \ldots \cup Pn$. If time restriction of the parent clause $tr \neq \epsilon$, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let $\mathcal{C}_1 = (N_{\mathcal{C}_1}, n_{0_{\mathcal{C}_1}}, E_{\mathcal{C}_1}, I_{\mathcal{C}_1}), \mathcal{C}_2 = (N_{\mathcal{C}_2}, n_{0_{\mathcal{C}_2}}, E_{\mathcal{C}_2}, I_{\mathcal{C}_2}), \ldots, \mathcal{C}_n = (N_{\mathcal{C}_n}, n_{0_{\mathcal{C}_n}}, E_{\mathcal{C}_n}, I_{\mathcal{C}_n})$. Considering the case where $g \neq \epsilon$ and $tr \neq \epsilon$, and having that $E_{\mathcal{C}_1}*, E_{\mathcal{C}_2}*, \ldots, E_{\mathcal{C}_n}*$ are the sets of edges considering time restriction tr together with their own time restriction, the resulting network of automata is therefore $\mathcal{C}*_i = (N_{\mathcal{C}*_i}, n_{0_{\mathcal{C}*_i}}, E_{\mathcal{C}*_i}, I_{\mathcal{C}*_i})$, $i = 1, \ldots, n$, which elements are described in next page.

(9) If several norms are composed by an **OR-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, Or\, C_2\, Or \ldots Or\, C_n, \epsilon)$, their composition corresponds to an automaton in which the automata corresponding to each one of the norms is considered as an alternative. Let us consider $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ the automata corresponding to the norms we are composing. The resulting automaton $OR*$ preserves the structure of the automata we are composing, adding two nodes called C_{init} and C_{final}. We define an urgent edge performing no action for each one of the norms we are composing connecting C_{init} with the initial node of the automaton corresponding to the norm and we also define an urgent edge performing no action for each one of the norm we are composing connecting the final node of its automaton with C_{final}. We add another node C_{skip} if guard condition of the parent clause $g \neq \epsilon$ and an urgent edge from C_{init} to this new node guarded with the guard condition negated $(\neg g)$. In the final node of this new structure we keep the violation, satisfaction and permission sets of the previous final node, so we have that Vfinal = $V1|V2|\ldots|Vn$, Sfinal = $S1|S2|\ldots|Sn$ and Pfinal = $P1|P2|\ldots|Pn$. If time restriction of the parent clause $tr \neq \epsilon$, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let $\mathcal{C}_1 = (N_{\mathcal{C}_1}, n_{0_{\mathcal{C}_1}}, E_{\mathcal{C}_1}, I_{\mathcal{C}_1}), \mathcal{C}_2 = (N_{\mathcal{C}_2}, n_{0_{\mathcal{C}_2}}, E_{\mathcal{C}_2}, I_{\mathcal{C}_2}), \ldots, \mathcal{C}_n = (N_{\mathcal{C}_n}, n_{0_{\mathcal{C}_n}}, E_{\mathcal{C}_n}, I_{\mathcal{C}_n})$. Considering the case where $g \neq \epsilon$ and $tr \neq \epsilon$, and having that $E_{\mathcal{C}_1}*, E_{\mathcal{C}_2}*, \ldots, E_{\mathcal{C}_n}*$ are the sets of edges considering time restriction tr together with their own time restriction, the resulting automaton is therefore $OR* = (N_{OR*}, n_{0_{OR*}}, E_{OR*}, I_{OR*})$, where:

- $N_{OR*} = N_{\mathcal{C}_1} \cup N_{\mathcal{C}_2} \cup \ldots \cup N_{\mathcal{C}_n} \cup \{C_{init}, C_{final}, C_{skip}\}$.
- $n_{0_{OR*}} = C_{1init}$.
- $E_{OR*} = E_{\mathcal{C}_1} * \cup E_{\mathcal{C}_2} * \cup \ldots \cup E_{\mathcal{C}_n} * \cup \{C_{init} \longrightarrow_u C_{1init}, C_{init} \longrightarrow_u C_{2init}, \ldots, C_{init} \longrightarrow_u C_{ninit}\} \cup \{C_{1final} \longrightarrow_u C_{final}, C_{2final} \longrightarrow_u C_{final}, \ldots, C_{nfinal} \longrightarrow_u C_{final}\} \cup \{C_{init} \xrightarrow{\neg g}_u C_{skip}\}$.
- $I_{OR*} = I_{\mathcal{C}_1} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_1} - \{C_{1final}\}\} \cup I_{\mathcal{C}_2} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_2} - \{C_{2final}\}\} \cup \ldots \cup I_{\mathcal{C}_n} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_n} - \{C_{nfinal}\}\}$.

(10) If several norms are composed by a **SEQ-refinement**, that is, we have specified the diagram $(\epsilon, name, g, tr, C_1\, Seq\, C_2\, Seq \ldots Seq\, C_n, \epsilon)$, their composition corresponds to an automaton in which the automata corresponding to each one of the norms are connected in sequence. Let us consider $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ the automata corresponding to the norms we are composing. The resulting automaton $SEQ*$ preserves the structure of the automata we are composing, adding just one extra node C_{skip} if guard condition of the parent clause $g \neq \epsilon$ and an urgent edge from C_{1init} to this new node guarded with the guard condition negated $(\neg g)$. We connect with an urgent edge performing no action the ending node of each automaton in the sequence $(C_{1final}, C_{2final}, \ldots, C_{n-1final})$ with the initial node of the next automaton $(C_{2init}, C_{3init} \ldots, C_{ninit})$. This rule is not applied in the cases of C_{1init} (as there is not previous ending node to connect) and C_{nfinal} (as there is not following initial node to connect). In the initial node of each one of the composed automata we preserve the violation, satisfaction and permission sets of the previous final node.

If time restriction of the parent clause $tr \neq \epsilon$, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let $\mathcal{C}_1 = (N_{\mathcal{C}_1}, n_{0_{\mathcal{C}_1}}, E_{\mathcal{C}_1}, I_{\mathcal{C}_1}), \mathcal{C}_2 = (N_{\mathcal{C}_2}, n_{0_{\mathcal{C}_2}}, E_{\mathcal{C}_2}, I_{\mathcal{C}_2}), \ldots, \mathcal{C}_n = (N_{\mathcal{C}_n}, n_{0_{\mathcal{C}_n}}, E_{\mathcal{C}_n}, I_{\mathcal{C}_n})$. Considering the case where $g \neq \epsilon$ and $tr \neq \epsilon$, and having that $E_{\mathcal{C}_1}*, E_{\mathcal{C}_2}*, \ldots, E_{\mathcal{C}_n}*$ are the sets of edges considering time restriction tr together with their own time restriction, the resulting automaton is $SEQ* = (N_{SEQ*}, n_{0_{SEQ*}}, E_{SEQ*}, I_{SEQ*})$, where:

- $N_{SEQ*} = N_{\mathcal{C}_1} \cup N_{\mathcal{C}_2} \cup \ldots \cup N_{\mathcal{C}_n} \cup \{C_{skip}\}$.
- $n_{0_{SEQ*}} = C_{1init}$.
- $E_{SEQ*} = E_{\mathcal{C}_1} * \cup E_{\mathcal{C}_2} * \cup \ldots \cup E_{\mathcal{C}_n} * \cup \{C_{1init} \xrightarrow{\neg g}_u C_{skip}, C_{1final} \longrightarrow_u C_{2init}, C_{2final} \longrightarrow_u C_{3init}, \ldots, C_{n-1final} \longrightarrow_u C_{ninit}\}$.
- $I_{SEQ*} = I_{\mathcal{C}_1} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_1} - \{C_{1final}\}\} \cup I_{\mathcal{C}_2} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_2} - \{C_{2final}\}\} \cup \ldots \cup I_{\mathcal{C}_n} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathcal{C}_n} - \{C_{nfinal}\}\}$. $\square$