Computing and Informatics, Vol. 39, 2020, 1022–1060, doi: 10.31577/cai_2020_5_1022

FORMAL VERIFICATION OF UML MARTE SPECIFICATIONS BASED ON A TRUE CONCURRENCY REAL TIME MODEL

Nadia Chabbat

LAAS Laboratory, Department of Computer Science
University of Badji Mokhtar, Annaba, Algeria
e-mail: lydia.chab@hotmail.fr

Djamel Eddine Saidouni, Radja Boukharrou

MISC Laboratory, Department of Computer Science
University of Abdelhamid Mehri, Constantine 2, Algeria
e-mail: {djamel.saidouni, radja.boukharrou}@univ-constantine2.dz

Salim Ghanemi

LAAS Laboratory, Department of Computer Science
University of Badji Mokhtar
Annaba, Algeria
e-mail: ghanemisalim@univ-annaba.dz

Abstract. For critical embedded systems the formal validation and verification is required. However, the real-time model checking suffers from problems of state-space explosion and clock explosion. The aim of this paper is to ensure an improvement of the Modeling and Analysis of Real-Time Embedded systems (MARTE), which is de facto standard, with formal semantics for verification finality. Therefore, we propose an operational method for translating UML sequence diagrams with MARTE annotations to Time Petri nets with Action Duration specifications (DTPN). Based on true concurrency semantics, the semantics of these specifications are defined in terms of Duration Action Timed Automata (daTA).

Keywords: Real-time embedded system, UML MARTE, DTPN, duration action timed automata, parallel computing, sequence diagram, formal verification

1 INTRODUCTION

Real-time embedded systems solve behaviors constrained by time. These systems provide a specific function in a much larger system within time consideration [1, 2]. The design of such systems is associated with time constraints specification. Real-time embedded systems are often critical and require the modeling of real-time response at the functional level. The assessment of such systems may be achieved through verification and validation processes based on robust formal approaches to meet the required timed constrained functional system properties.

Several studies have proposed a model-based engineering approaches that offer advanced modeling mechanisms such as UML (Unified Modeling Language) [3]. MARTE is an OMG UML profile dedicated for modeling and analysis of real-time embedded systems [4, 5]. MARTE specifies concepts for characterizing UML elements in order to model software and hardware platforms, resources, and quantitative characteristics such as execution time. However, once the software is modeled, the difficulty lies in the expression of appropriate properties and formal checks. In order to meet these requirements, formal analytical approaches have been developed so to integrate or extend formal models in the designing process of real-time embedded systems developed with MARTE. Both works presented in [6, 7] use the Time Petri Net analyzer [8] (TINA) model-checking tool to verify temporal properties on structural and behavioral diagrams specified in MARTE.

Some work, as [9, 10], extended a formal language in order to take into consideration time aspects. In [9], the discrete-time of Promela language has been extended by a variable, named *"timer"*, that corresponds to the discrete-time *"countdown"*. However, the extended model is difficult to use for representing the coincidence clock tickings. Another work, presented in [10], proposes an extension of Promela language with discrete-time to allow the verification by SPIN model checker [11].

In [12], an interesting approach has been proposed. It interprets parallel activities, modeled in MARTE sequence diagrams, by parallel transitions in a Petri net like specification. More precisely, the specification is written in timed colored Petri nets with inhibitor arcs model (TCPNIA) [13]. In the later work, the duration of an activity is integrated as a constraint interval associated with the corresponding transition of the Petri net. To verify some properties, TCPNIA specification is translated to a Timed Automata [14, 15] in order to use some existing model checker tools as SMV [16]. However, this approach only expresses activity duration without considering *latency*, *delay* and *congestion* time specification. Since Timed Automata is based on interleaving semantics, there is no way to express the parallel execution of two activities. To overcome such limitation, it is possible to interpret each activity having a non-null duration by two sequential transitions modeling the

activity's start and end events. Though this solution is correct, it presents some serious inconvenience. In fact, larger number of transitions in the Petri net specification leads to a significant clock number increase in the associated timed automata. Indeed, the number of zones, respectively regions, in the zone graph, respectively region graph, is exponential to the number of clocks [15]. Thus, this leads to the state space combinatorial explosion problem [17, 18, 19].

In our proposed approach, Time Petri Nets with action Duration (DTPN) specification model is used as a semantics model of MARTE SD specification [20]. In fact, DTPN has a true concurrency semantics and considers both timing constraints and duration of actions. The true concurrency semantics allows the consideration of activities duration without using the split technique. The underlying semantics model is a Durational Action Timed Automaton (daTA) [21].

This paper defines an operational method for translating MARTE SD specifications to DTPNs ones. As a consequence, on one hand, the transformation leads to a small DTPN specification and on the other hand, the verification of a DTPN specification is based on its daTA corresponding model, which allows the reuse of clocks. So, this lead to reduce, in some case avoid the problem of the combinatorial explosion of the number of graph states, which is one of the main limitations of applying model-checking methods on industrial-sized models and, to reduce of the verification time, that is often exponential for large-size systems.

The rest of this paper is structured as follows: In Section 2, we present preliminary definitions of MARTE sequence diagrams, Time Petri Nets with action duration and durational action timed automata. Section 3 defines the formal semantics of MARTE sequence diagrams and formal translation rules of basic elements and some combined fragments. In Section 4, we present a case study to better illustrate the interest of the proposed approach, as well as its implementation in practice. Section 5 discusses the advantages of the proposed approach with related works. Finally, Section 6 gives some conclusions and perspectives of this work.

2 PRELIMINARY DEFINITIONS

2.1 MARTE Sequence Diagrams

The UML sequence diagram is a form of interaction diagram. It allows the description of a specific interaction in terms of participating objects and sequence of messages exchanged along progress to perform desired activity. Graphically, an interaction is composed by two lifelines and a message. A lifeline represents an instance corresponding to a particular object that participates in the interaction. The events along a lifeline are, in general, partially ordered, the order in which these events will occur. A message represents a communication that transmits information between objects or an object and its environment. A message specifies the kind of a communication between objects as synchronous or asynchronous and notes the occurrences of the sending event at the sender and the receiving event at the receiver level. In this work, we will focus on the order and type of messages,

synchronous, asynchronous and reply. Additionally, we will take into consideration the timing constraints imposed on transmitted messages between objects involved in the interaction.

A sequence diagram describes only a fragment of the system behavior. The complete behavior of the system can be expressed by a set of sequence diagrams to specify all possible interactions. An interaction is a behavior unit that focuses on the observable transmissions of information between connected objects over time. Each interaction may be caused by actions executed by communicated objects. As defined in [22]: *an action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty.* For examples, an action can be a call operation, send signal, receive signal, write variable or read variable. The execution of actions can result by an event as a call or a signal event.

To get more complex interactions, combined fragments technique may be used. A combined fragment consists of an operator and a number of operands. Depending on the used operator, the number of operands is defined. For example, break, opt, loop, assert, ref and neg operators have one operand. Most other operators like alt, seq and par, have more than one operand. Fragments and their operators can be inductively combined for describing complex interactions.

In order to enrich UML models with annotations related to time (values and timing constraints), the UML MARTE profile provides basic and advanced time modeling concepts, such as stereotypes, which allow the consideration of temporal behaviors. This profile is intended to replace the existing UML SPT Profile (Schedulability, Performance and Time) [23], that is incompatible with UML 2 and MDA standards [24].

Figure 1 shows the different concepts in a sequence diagram used for specifying time and timing constraints in UML MARTE profile. These elements are defined in the SimpleTime package of CommonBehaviors. TimeObservation is a reference to an instant of time, while DurationObservation is a reference to a duration of an execution. TimeConstraints can be in the form of duration, as well as some instants of time; sequence diagram supports both. DurationConstraint defines a Constraint that refers to a duration interval, which defines the range between two duration. A duration defines a value specification that specifies the temporal distance between two time instants. Timing constraints are expressed between a pair of braces. The annotations in color are not part of the model, they are used to specify model elements. More details about this system example can be found in [25].

2.2 Time Petri Nets with Action Duration

Time Petri Nets with Action Duration can be considered as a generalization of Merlin's TPNs [27], T-TdPNs [28] and P-TdPN [29]. The basic idea of DTPN is to associate two date's *min* and *max* with each transition that define its firing interval (temporal interval). Although the firing of a transition is instantaneous, the execution duration of the action associated to this transition may have non-null duration. For example, let t be a transition associated to the action which

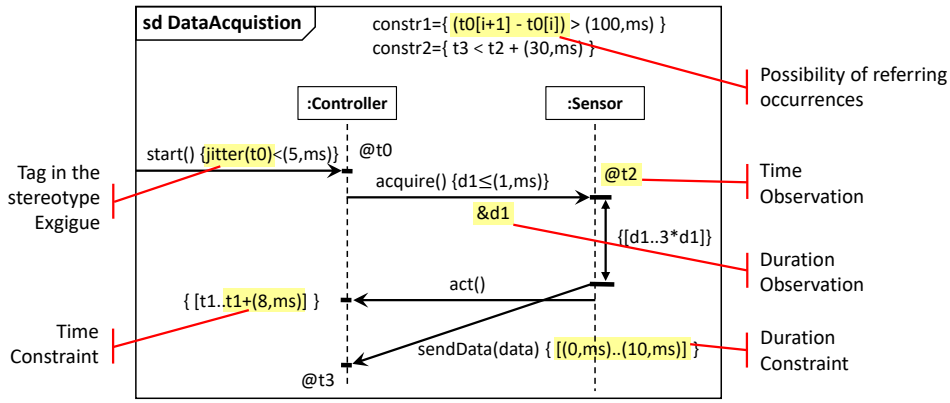
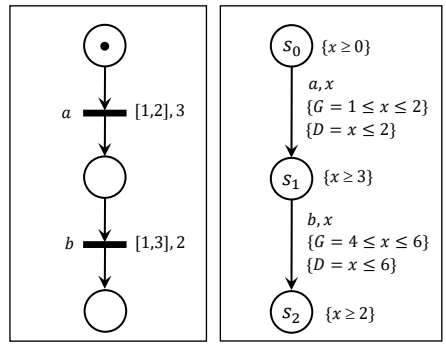


Figure 1. Time and timing constraints illustration [26]

has a duration d. If θ is the enabling date of t then the firing of t will be in the time interval $[\theta + min, \theta + max]$. The firing of t marks the start of execution of the associated action. Figure 2 a) shows an example of a time Petri nets with action duration.



a) DTPN b) daTA

Figure 2. DTPN and its corresponding daTA automata

A place of a DTPN corresponds to two sets: a set of *available tokens* or *free tokens* and a set of *unavailable tokens* or *bound tokens*. Unavailable tokens, put on the right side of a place, are bound to the firing of transitions associated to actions that are currently running. In a DTPN, an unavailable token becomes available if the end of execution of the action associated to the transition that produced this token is reached. A token in place p at the time ϑ becomes *available* (in the left side of p) at the time $\vartheta + d$. Thus, the token is bound to the firing of the transition during the interval $[\vartheta, \vartheta + d[$ and it becomes free at the time $\vartheta + d$.

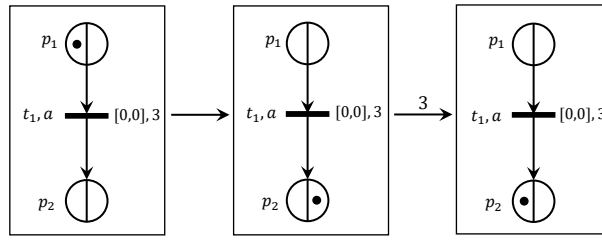


Figure 3. Marked DTPN

In Figure 3 a), the token in place P_1 is not bound to any transition. This token is called *free*. In the case when the transition would be fired, it could be argued that the action associated to the firing of t_1 has started its execution. This is marked by the presence of the token in place P_2 (Figure 3 b)). Thus, the token in place P_2 is bound to the firing of t_1, but after completion of the action a, i.e. after 3 units of time, this token will become free (Figure 3 c)). In a place, the set of free tokens will be denoted by FT, while bound tokens set will be denoted by BT.

Definition 1. Let $\mathbb{T}$ be a non-negative temporal domain, like $\mathbb{Q}^+$ or $\mathbb{R}^+$.

Definition 2. Let Act be a finite set of actions, i.e. an alphabet. A Time Perti Net with action Duration (DTPN) on $\mathbb{T}$ and of support Act is a tuple $\langle P, T, B, F, \lambda, SI, \Gamma \rangle$, such that:

- $Q = \langle P, T, B, F \rangle$ is Perti net, where P is a set of places, T is set of *transitions* such that $P \cap T = \emptyset$;
- $B : P \times T \to \mathbb{N}$ is a backward incidence function such that $B(p_i, t_j)$ represents arc weight from t_j to p_i;
- $F : P \times T \to \mathbb{N}$ is a forward incidence function such that $F(p_i, t_j)$ represents arc weight from p_i to t_j;
- $\lambda : T \to Act \cup \{\tau\}$ is a labelling function of a $DTPN$. If $\lambda(t) \in Act$ then t is called observable or external;
- $SI : T \to \mathbb{T} \times \mathbb{T} \cup \{\infty\}$ is a function that associates to each transition a static firing interval;
- $\Gamma : Act \to D$ is a function that associates to each action its static duration.

I is the set of all intervals of a DTPN such that $I(t) = [min, max]$ is the interval associated to the transition t. We denote by $\downarrow I(t) = min$ and $\uparrow I(t) = max$, two functions which give respectively the lower and upper bound of an interval.

As commonly in use in the literature, we write $^\circ t$ (resp. t°) to denote the set of places such that $^\circ t = (p \in P / B(p,t) > 0$ (resp. $t^\circ = \{p \in P / F(p,t) > 0\}$), and $^\circ p$ (resp. p°) to represent the set of transitions such that $^\circ p = \{t \in T / F(p,t) > 0\}$

(resp. $p^\circ = \{t \in T/B(p,t) > 0\}$). Noting that marked DTPN is a tuple $\langle PN, M_0 \rangle$, such that $PN = \langle P, T, B, F, \lambda, SI, \Gamma \rangle$ is a DTPN, and M_0 is its initial marking, where $\forall p \in P : M(p) \in \mathbb{N}$.

2.3 Durational Action Timed Automata

Durational action Timed Automata (daTA) is structurally a subclass of timed automata [14, 15, 30]. However, the difference to be underlined is the one concerning the semantics associated with the model. The daTA model [21, 31] is a timed model defined by a timed transition system based on a true-concurrency semantics, expressing parallel behaviors and supporting at the same time timing constraints, explicit actions duration, structural and temporal non-atomicity of actions i.e., actions may be divisible and of non-null duration.

The daTA model supports the notions of urgency and deadlines as timing constraints of the system. An action duration is expressed by a duration condition associated to states of the model. On the other hand, timing constraints, due to restrictions on the enabling domain of an action, are expressed by the *enabling constraint G* (for Guard) and by *urgency constraint D* (for Deadline) at the level of daTA transitions. In addition, a transition represents only the start of an action, end of execution is captured by the corresponding duration expressed at the level of target state. On the target state, a timed expression manifests that the action is potentially in execution.

From operational point of view, with each action a *clock* is associated which is *reset* at the start of the action. This clock will be used in the construction of the timing constraints as guards of the transitions. Figure 2 b) shows the structure of the corresponding daTA to DTPN of Figure 2 a). This daTA is composed of three states and two transitions labelled with two actions a and b of durations 3 and 2 units of time, respectively. From the initial state S_0 of the illustrative daTA, the execution of action a leads to a reset of clock x associated with it. The expression $x \geq 3$ in state S_2 represents a duration condition on action a and means that a is potentially in execution until the clock x reaches the value 3. The action a does not wait for the end of any other action, so the clock designated by x is used in the enabling domain of this action.This enabling domain will be expressed by the guard and the deadline on the clock x as $(1 \leq x \leq 2)$ and $(x \leq 2)$. Below, we define the timed domain modeling clocks of daTA.

Definition 3. Let $\mathcal{H}$ be a set of clocks with non-negative values (within a time domain $\mathcal{H}$, like $\mathbb{Q}^+$ or $\mathbb{R}^+$). The set $\Phi_t(\mathcal{H})$ of temporal constraints γ over $\mathcal{H}$ is $\gamma x \sim t$, where x is a clock in $\mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ and $t \in \mathbb{T}$. F_x is used to indicate a constraint of the form $x \sim t$. A valuation v for $\mathcal{H}$ is a function which associates to each $x \in \mathcal{H}$ a value in $\mathbb{T}$. The valuation v for $\mathcal{H}$ satisfies a temporal constraint γ over $\mathcal{H}$ iff γ is true by using clock values given by v. For $I \subseteq \mathcal{H}$, $[I \rightarrow 0]$ indicates the valuation for $\mathcal{H}$ which assigns 0 value to each $x \in I$, and agrees with v over the other clocks of $\mathcal{H}$. The set of all valuations for $\mathcal{H}$ is noted $\Xi(\mathcal{H})$. The

satisfaction relation $\models$ for temporal constraints is defined over the set of valuations for $\mathcal{H}$ by $(v \models x \sim t) \Leftrightarrow (v(x) \sim t)$ such that $v \in \Xi(\mathcal{H})$. $2^{\mathbb{T}}_{fn}$ is used to denote the set of finite subsets of a set $\mathbb{T}$.

Definition 4. A daTA is a tuple $\langle S, L_s, s_0, \mathcal{H}, T_D \rangle$ of the support Act, where:

- S is a finite set of states;
- $L_s : S \to 2^{\Phi_{t(\mathcal{H})}}_{fn}$ is a function which assigns to each state s the set F of ending condition (duration conditions) of actions possibly in execution in s;
- $s_0 \in S$ is the initial state, such that $L_s(s_0) = \emptyset$;
- $\mathcal{H}$ is a finite set of clocks;
- $T_D \subseteq S \times 2^{\Phi_{t(\mathcal{H})}}_{fn} \times 2^{\Phi_{t(\mathcal{H})}}_{fn} \times Act \times \mathcal{H} \times S$ is the set of transitions.

A transition (s, G, D, a, x, s') represents a switch from state s to state s' by starting execution of action a and resetting clock x. G is the corresponding guard, which must be satisfied to fire the transition. D is the corresponding deadline which requires, at the moment of its satisfaction, that action a must occur.

(s, G, D, a, x, s') can be written $s \xrightarrow{G,D,a,x} s'$. Figure 2 b) shows the structure of the corresponding daTA to DTPN of Figure 2 a).

Definition 5. The semantics of a daTA $\mathcal{A} = \langle S, L_s, s_0, \mathcal{H}, T_D \rangle$ is defined by associating with it an infinite transition system $S_{\mathcal{A}}$ over $Act \cup \mathbb{T}$. A state of $S_{\mathcal{A}}$, viewed as a configuration, is a pair $\langle s, v \rangle$ such that s is a state of $\mathcal{A}$ and v is a valuation for $\mathcal{H}$. A configuration $\langle s_0, v_0 \rangle$ is initial if s_0 is the initial state of $\mathcal{A}$ and $\forall x \in \mathcal{H}$, $v_0(x) = 0$. Two types of transitions between $S_{\mathcal{A}}$ configurations are possible, and which correspond respectively to time passing (Rules (1) and (2)) and the launching of a transition from $\mathcal{A}$ (Rule (3)):

$$\frac{d \in \mathbb{T} \quad \forall d' \leq d, v + d' \not\models \mathfrak{D}}{\langle s, v \rangle \xrightarrow{d} \langle s, v + d \rangle}, \tag{1}$$

$$\frac{\varepsilon \in \mathbb{T} \quad v + \varepsilon \models \mathfrak{D} \wedge \varepsilon \in \eta}{\langle s, v \rangle \xrightarrow{\varepsilon} \langle s, v + \varepsilon \rangle}, \tag{2}$$

$$\frac{(s, G, D, a, x, s') \in T_D \quad v \models G}{\langle s, v \rangle \xrightarrow{a} \langle s', [\{x\} \mapsto 0]v \rangle}. \tag{3}$$

In Rule (2), η corresponds to the smallest real quantity of time in which no action occurs [31]. In Rule (3), $\mathfrak{D} = \vee_{i \in I} D_i$, where $\{(s, G_i, D_i, a_i, x, s_i)\}_{i \in I}$ is the set of all transitions stemming from state s. Indeed, whenever a D_i holds, time cannot progress regardless of the other D_i.

In order to guarantee that at least a transition could be drawn starting from a state if time cannot progress any more within this state, the formula $D_i \Rightarrow G_i$ must be satisfied.

Remark 1. For urgency domains, we require that deadline can be only of the form $x \leq t$ or $x < t$.

3 TRANSLATION OF MARTE SEQUENCE DIAGRAMS TO DTPN

This paper proposes a translation of a high-level model written in MARTE SD towards a specific formal time model that is DTPN. Figure 4 gives a general overview of the approach. The approach will follow two main steps. In the first step, MARTE sequence diagrams are formalized. The translation method is developed in the second step. It is defined inductively starting from the basic elements.

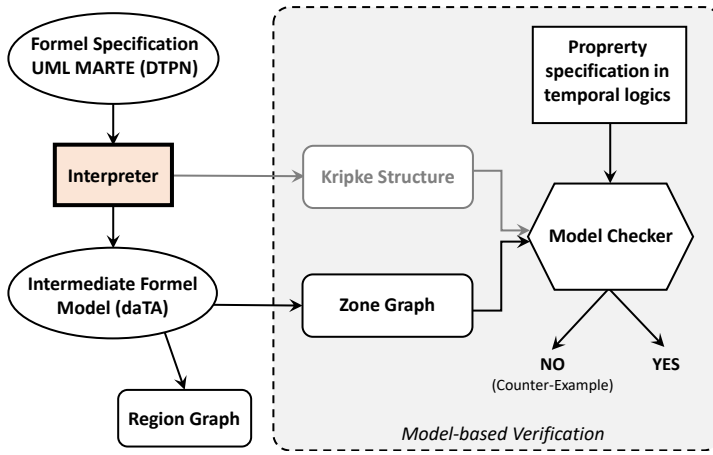


Figure 4. Verification process

3.1 Formalisation

Sequence diagrams are semi-formal notation, in other words the syntax and semantics notations are open to different interpretations. In the literature, there are several research papers that address the formal formalization of sequence diagrams [32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Among these works, the approach of [41] is an interesting one as a formal translation. In this work, the authors have defined formal rules for the translation of an UML sequence diagram to a corresponding coloured Petri nets. Nevertheless, the transmission processing between two communicating objects in the sequence diagram is specified as a single transition with an abstraction to the different primitives of communication, i.e. sending process, transmitting and receiving process. In addition, this approach models the transmission without further consideration of any details in relation with its execution like *latency* and *time of execution*.

To overcome these limitations and in order to consider timing constraints and duration of execution, we propose a formal translation of MARTE sequence diagrams to Time Petri nets with action duration model. Besides the formal translation and the timing consideration, we investigate the parallel execution specification. For this purpose, the true-concurrency semantics based DTPN model is used to obtain durational action Timed Automata (daTA) (semantics representation) [42, 43]. In daTA, both true concurrency and action duration are considered. This structure allows the verification of properties related to parallel evolution of actions within timing constraints. We note that some properties related to reachability may be checked by means of KRONOS and UPPAAL like tools [43, 44]. However, for properties dealing with true concurrency behaviors, FOCOVE model checker may be used [45].

So, in the proposed approach, we follow the same formalization principle proposed in [41] with some differences. In fact, on one hand the transmission primitives specification between communicating parties is defined and in another hand the formalization is made inductive by associating a DTPN specification to each event and defining the translation of composite interaction in terms of its DTPN corresponding sub-elements. Also, we specify timing constraints, like latency, and execution duration of each action separately in order to model transmission process under timing constraints.

3.2 Formal Definitions

In formal terms, we define the MARTE sequence diagrams as follows:

Definition 6. MARTE $SD = (N, O, E, <_g, M, Act, D, Tc, \lambda, S, T, Pre, Post, Cf)$ where:

- N is a set of diagram names;
- O is a finite set of objects;
- $E = \bigcup_{i \in O} E_i$ is a set of events such that $E_i \bigcap E_j = \emptyset$ for any $i \neq j \in O$;
- $<_g = < \cup \{\bigcup_{i,j \in O, i \neq j} <_{i,j}\}$
  - $<= \bigcup_{i \in O}$ is a set of partial orders on events of E_i such that $\forall i \in O, <_i \subseteq E_i \times E_i$;
  - $<_{i,j}$ defines an order between events e, e' such that $\forall (e, e') \in <_{i,j}$, $e \in O_i$, and $e' \in O_j$ or $e \in O_j$ and $e' \in O_i$ and e precedes e'.
- M is a finite set of message labels;
- *Act* is a set of actions. Each action $a \in Act$ can be launched by event. According to the specification context, we can distinguish different kinds of actions. Examples for actions are *synchronous sending (Sysen) or asynchronous sending a message (Asysen), synchronous receiving (Syrec) or asynchronous receiving*

a message (Asyrec), sending reply (Sreply), Receiving reply (Rreply), transmissions of message, activities or any behavior to be executed in the system.

$$Act = \{Sysen, Asysen, Syrec, Asyrec, Sreply, Rreply, Activity1, \ldots, Activityn, Trans1, \ldots, Transn\};$$

- $Dc : Act \to \mathbb{T}$ is a function that associates to each action a duration which may be null;
- $Tc : E \to \mathbb{T} \times \mathbb{T}$ is a function that associates a time interval to each event such that $\forall e \in E_i, Tc(e) = [d, d + t]$ which means that event e should be executed in the time interval $[d, d + t]$;
- $\lambda : E \to Act$ is a labelling function which associates an action name to each event. For each event $e \in E_i$, e may be defined by 3-uple $\langle e, D(\lambda(e)), Tc(e)\rangle$.
- S is the set of all the possible states with $S = \bigcup_{O_i \in O} S_i$ where S_i is the set of states of an object O_i. $S_i = \bigcup_{e \in E_i} S_e$, such that:
  - $\forall e \in E$, $S_e = \{s_e^0, s_e^1\}$;
  - If $e \in E$ is associated to the sending action then $S_e = \{s_e^0, s_e^1, s_e^{out}\}$;
  - If $e \in E$ is associated to the receiving action then $S_e = \{s_e^0, s_e^1, s_e^{in}\}$.

  Similarly to events, different objects cannot share the states, $S_i \cap S_j = \emptyset$ for $O_i \neq O_j \in O$;
- T is a set of transitions, such that for each transition $t \in T$ is associated to an event $e \in E$ and is of the form $\langle e, D(\lambda(e)), Tc(e)\rangle$. The arcs linking states to transitions are labeled by the Pre function, whereas the arcs linking transitions to states are labeled by the $Post$ function;
- $Pre : N \to 2^{S \times T}$ is an input function associating for each diagram sd, a set of arcs, where $(s, t) \in Pre(sd)$ defines the arc from s to t;
- $Post : N \to 2^{S \times T}$ is an output function associating for each diagram sd, a set of arcs, where $(s, t) \in Post(sd)$ defines the arc from t to s.

  For a given diagram sd:
  - The set of input arcs of transition $t \in T$ is denoted ${}^\circ t = \{(s, t) \in Pre(sd) \mid s \in S\}$;
  - The set of output arcs of transition $t \in T$ is denoted $t^\circ = \{(s, t) \in Post(sd) \mid s \in S\}$;
  - The set of input arcs of state $s \in S$ is denoted ${}^\circ s = \{(s, t) \in Pre(sd) \mid t \in T\}$;
  - The set of output arcs of state $s \in S$ is denoted $s^\circ = \{(s, t) \in Post(sd) \mid t \in T\}$;

- CF is a combined fragment defined by a type of operator and one or more operands.

- *Type* $= \{seq, alt, par, opt, break, loop, ref, strict, critical, neg, assert, ignore, consider\}$;
- *Op* is a finite set of operands;
- *Guard* is a boolean or temporal expression which associates a guard to an operator or to a combined fragment;
- *Frag* is a set of nested combined fragments inside the n^{th} operands of the m^{th} interaction fragments.

3.3 MARTE SD to DTPN Transformation

In the next lines, we present the translation rules and we explain through examples the translation between the input elements of the MARTE sequence diagram to the output elements of DTPN. Considering the different types of transmission between communicating objects, duration of the desired activities to be executed and the timing constraints imposed on the sending and receiving transmitted messages. By using DTPN, the action durations are fixed once and for all at the enabling moment of the system. The case where actions have durations chosen from an interval $[m, M]$ is not considered. Also, we note that the TimeConstraint stereotype presented in Section 2 is expressed by time interval $[min, max]$.

3.3.1 Basic Elements

Definition 7. For a given diagram sd, let $\langle e_1, d_1, tc_1\rangle$ and $\langle e_2, d_2, tc_2\rangle$ be two different transitions corresponding to a given transmission, such that:

- If $O_i, O_j \in O$ and $i \neq j$ such that $e_1 \in E_i$ and $e_2 \in E_j$, then the transitions represent an external evolution between both objects O_i and O_j. It is said an inter-objects transmission;
- If $\exists O_i \in O$ such that $e_1, e_2 \in E_i$ and $e_1 < e_2$, and $\nexists e_3 \in E_i$ such that $e_1 < e_3 < e_2$, then the transitions represent a local evolution in the object O_i. It is said an intra-objects transmission. In such case, $d_1 = d_2 = 0$ and $tc_1 = tc_2 = [0, 0]$.

Inter-objects transmission.

- **Synchronous transmission:** Synchronous transmissions are used when the sender waits for a response from the receiver to continue its operations. A synchronous transmission blocks the progression of the operations of its sender until the receiver gets its message (weak synchronization) or until the sender receives the response (strict synchronization).

  Let us consider the example of Figure 5 which shows a synchronous transmission and its reply. In the following paragraph, we detail how various elements of this sequence diagram (source model) are translated to the elements of DTPN (destination model).

In Figure 5, the Synchronous transmission represents an interaction between two objects O_1 and O_2 associated to both events e_1 and e_2 necessarily different. The translation of this transmission to an equivalent DTPN is based on the interpretation of each event and its associated action in the diagram with taking into account the two states (*before* and *after*) of this event. To construct the DTPN, we split the synchronous transmission in two principal steps:

1. sending the synchronous transmission and
2. receiving the synchronous transmission reply.

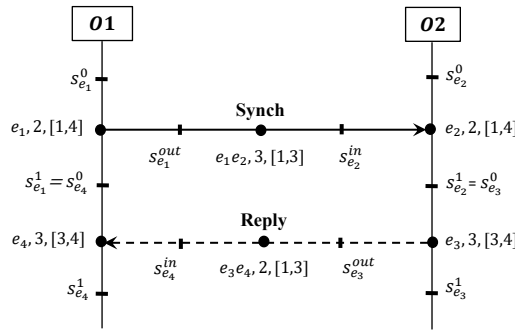


Figure 5. Synchronous transmission

Step 1:

1. Object O_1 launches a transmission sending operation at event e_1 to object O_2, which is constrained by the time interval $[1, 4]$ and duration equal to two units of time. The event e_1 is translated to a Petri net transition t_{e_1} constrained by the time interval $[1, 4]$ and an action of duration 2 units of time. The two states (before and after) of event e_1 are translated to the two places $s^0_{e_1}$, $s^1_{e_1}$. Also, we add the arcs that relate place to transition or transition to place. This translation is illustrated by Figure 6 a).
2. Object O_2 receives the transmission action at event e_2, with duration equal to 2 which may be delayed by 4 units of time. Since, the constraint interval $[1, 4]$ is considered. The corresponding DTPN is represented by Figure 6 b).
3. For representing the intermediate transmission action between object O_1 and object O_2, we create an event named e_1e_2 and two states s^{out}_{e1}, $s^{in}_{e_2}$. The event e_1e_2 models the action, which takes in our case a duration between 4 and 6 units of time. For considering this characteristic, we consider the constraint interval $[1, 3]$ and a duration of the transmission action equal to 3 units of time. Figure 6 c) shows the corresponding DTPN.

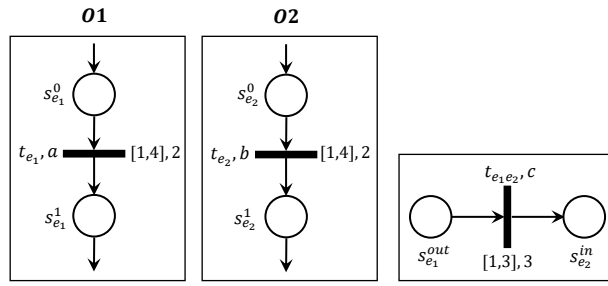


Figure 6. Elementary synchronous transmission corresponding DTPNs

4. After adding the arcs relating the three subnets DTPN models two subnets are related to two subnets, the complete resulting construction of this step is given by Figure 7.

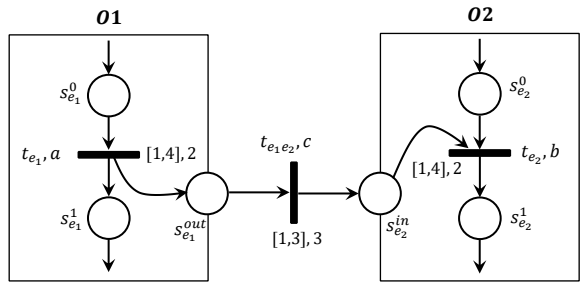


Figure 7. Sending transmission corresponding DTPN

Formally, S and T are the least sets verifying the following construction conditions:

- Let $\{t_{e_1}, t_{e_1e_2}, t_{e_2}\} = \{\langle e_1, 2, [1,4]\rangle, \langle e_1e_2, 3, [1,3]\rangle, \langle e_2, 2, [1,4]\rangle\}$ be a set of transitions modeling a synchronous transmission between objects O_1 and O_2.
- Let $e_1 \in E_1$ and $e_2 \in E_2$ be two events, such that $\lambda(e_1) =$ Sysen and $\lambda(e_2) =$ Syrec, with $(e_1, e_2) \in <_{i,j}$, $\{s^0_{e_1}, s^1_{e_1}, s^{out}_{e_1}, s^0_{e_2}, s^1_{e_2}, s^{in}_{e_2}\} \subseteq S$ and $\{t_{e_1}, t_{e_2}, t_{e_1e_2}\} \subseteq T$, then:

$$
\begin{aligned}
{}^\circ t_{e_1} &= \{s^0_{e_1}\};\\
t^\circ_{e_1} &= \{s^1_{e_1}, s^{out}_{e_1}\};\\
{}^\circ t_{e_2} &= \{s^{in}_{e_2}, s^0_{e_2}\};\\
t^\circ_{e_2} &= \{s^1_{e_2}\};
\end{aligned}
$$

$$^{\circ}t_{e_1e_2} = \{s^{out}_{e_1}\};$$
$$t^{\circ}_{e_1e_2} = \{s^{in}_{e_2}\}.$$

Step 2: Due to its similarity with the sending transmission in step1, it is possible to apply the same translation scheme in this step. As a consequence of the transmission reception by object O_2, this later responses by executing the behavior that is matched to that transmission action.

1. After the transmission processing, object O_2 sends the transmission reply at event e_3 back to object O_1. The event e_3 is characterized by a duration equal to 3 units of time and time interval $[3, 4]$. This is represented by the following DTPN (Figure 8 a)).

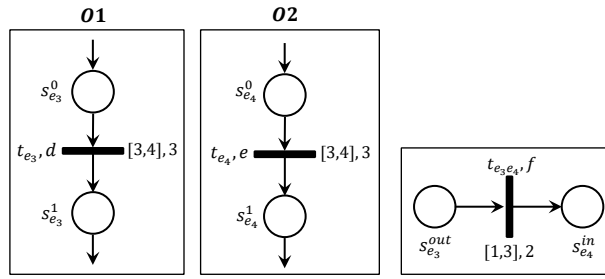


Figure 8. Elementary transmission reply corresponding DTPNs

2. Objects respond to messages that are generated by objects executing communication actions. The object O_1 receives transmission reply at event e_4, which has 3 units of time as duration and $[3, 4]$ as timing constraint. The corresponding DTPN is illustrated by Figure 8 b).
3. For modeling the reply transmission action between sending action at event e_3 and receiving action at event e_4, we create an event named e_3e_4 and two states $s^{out}_{e_3}$, $s^{in}_{e_4}$. This event takes in this case a duration between 3 and 5 units of time. Since then we consider the constraint interval $[1, 3]$ and a duration of the action equal to 2 units of time. The corresponding DTPN model is shown in Figure 8 c).
4. At this stage, the arcs to compose the three subnet DTPNs models are added. The complete resulting model is shown by Figure 9.

At last, the result of translation of the example in Figure 5 is depicted by Figure 10.

Formally, S and T are the least sets verifying the following construction conditions:

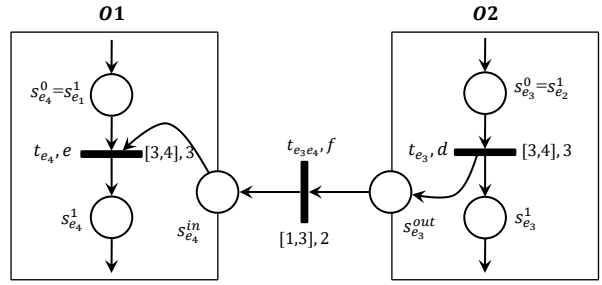


Figure 9. Transmission reply corresponding DTPN

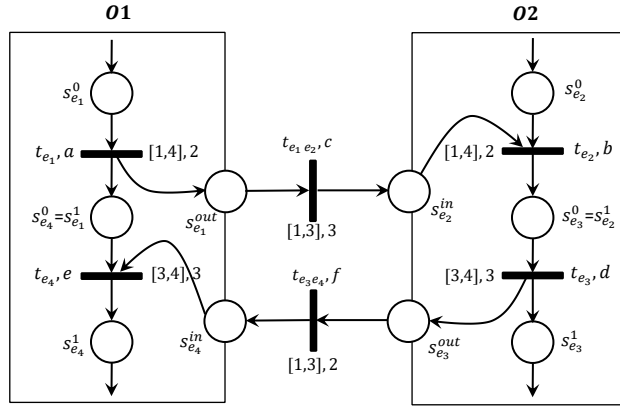


Figure 10. Synchronous transmission corresponding DTPN

– Let $\{t_{e_3}, t_{e_3e_4}, t_{e_4}\} = \{\langle e_3, 3, [3,4]\rangle, \langle e_3e_4, 2, [1,3]\rangle, \langle e_4, 3, [3,4]\rangle\}$ be a set of transitions which models a transmission reply between the two objects O_1 and O_2.
– Let $e_3 \in E_2$ and $e_4 \in E_1$ be two events, such that $\lambda(e_3) =$ Sreply and $\lambda(e_4) =$ Rreply with $(e_3, e_4) \in <_{i,j}, \{s^0_{e_3}, s^1_{e_3}, s^{out}_{e_3}, s^0_{e_4}, s^1_{e_4}, s^{in}_{e_4}\} \subseteq S$ and $\{t_{e_3}, t_{e_4}, t_{e_3e_4}\} \subseteq T$, then:

$$
\begin{aligned}
{}^\circ t_{e_3} &= \{s^1_{e_2}\};\\
t^\circ_{e_3} &= \{s^1_{e_3}, s^{out}_{e_3}\};\\
{}^\circ t_{e_4} &= \{s^{in}_{e_4}, s^1_{e_1}\};\\
t^\circ_{e_4} &= \{s^1_{e_4}\};\\
{}^\circ t_{e_3e_4} &= \{s^{out}_{e_3}\};\\
t^\circ_{e_3e_4} &= \{s^{in}_{e_4}\}.
\end{aligned}
$$

- **Asynchronous transmission:** Asynchronous transmission is used when the sender does not need to wait for response from receiver, it continues its execution after sending the message. The basic functionality is the same as the synchronous transmission. We consider the sequence diagram in Figure 11, which shows the transmission of an asynchronous message between objects O_1 and O_2 at events e and e'.

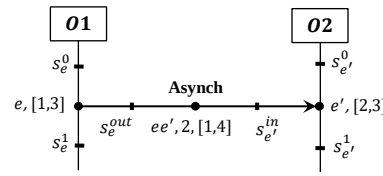


Figure 11. Asynchronous transmission

1. The event e in object O_1, represents the sending action, which is instantaneous and with timing constraint $[1, 3]$. The event e is interpreted as a Petri net transition t_e constrained by the time interval $[1, 3]$, its two states are translated to places s_e^0 and s_e^1. Moreover, we add the arcs linking places to transitions. This translation is represented by Figure 12 a).

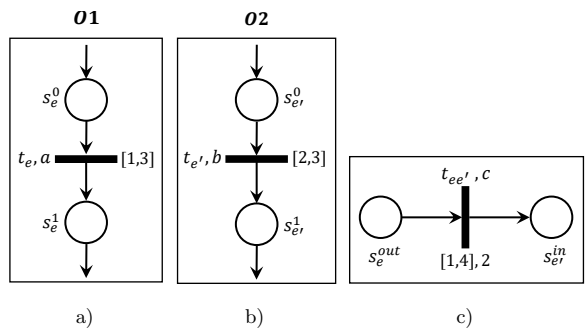


Figure 12. Elementary asynchronous transmission corresponding DTPNs

2. The receiving action in object O_2 is represented by the event e' and the reception message should be in time interval between 2 and 3 units of time. The interpretation of this part can be represented by the DTPN given in Figure 12 b).
3. For representing the asynchronous transmission operation between sending action and receiving action, we add an event named ee' between both events e and e' that takes in our case a duration between 3 and 6 units of time. Hence, we consider the constraint interval $[1, 4]$ and a duration of the action equal to 2 units of time. Figure 12 c) illustrates the corresponding resulting DTPN.

4. To ensure asynchronous transmission between objects O_1 and O_2 we add two arcs, the first arc from transition t_e to input state of transition $t_{ee'}$ and the second arc connects the output state of this last transition to transition $t_{e'}$.

As shown in Figure 13, we obtain the complete DTPN associated to the sequence diagram of Figure 11.

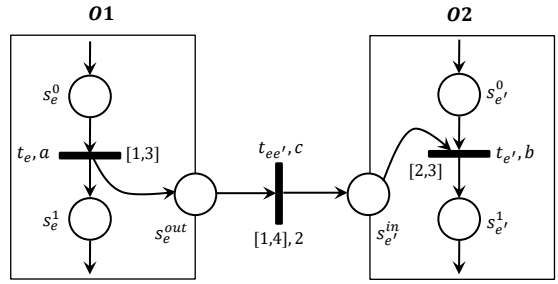


Figure 13. Asynchronous transmission corresponding DTPN

Formally, S and T are the least sets verifying the following construction conditions:

- Let $\{t_e, t_{ee'}, t_{e'}\} = \{\langle e, 0, [1,3]\rangle, \langle ee', 2, [1,4]\rangle, \langle e', 0, [2,3]\rangle\}$ be a set of transitions which models an asynchronous transmission between two objects O_1 and O_2.
- Let $e \in E_1$ and $e' \in E_2$ be two events, such that $\lambda(e) =$ Asysen, $\lambda(e') =$ Asyrec with $(e, e') \in <_{i,j}, \{s_e^0, s_e^1, s_e^{out}, s_{e'}^0, s_{e'}^1, s_{e'}^{in}\} \subseteq S$ and $(t_e, t_{e'}, t_{ee'}) \subseteq T$, then:

$$
\begin{aligned}
{}^\circ t_e &= \{s_e^0\};\\
t_e^\circ &= \{s_e^1, s_e^{out}\};\\
{}^\circ t_{e'} &= \{s_{e'}^{in}, s_{e'}^0\};\\
t_{e'}^\circ &= \{s_{e'}^1\};\\
{}^\circ t_{ee'} &= \{s_e^{out}\};\\
t_{ee'}^\circ &= \{s_{e'}^{in}\}.
\end{aligned}
$$

Intra-objects transmission.
An intra-objects transmission is a self-transmission in sequence diagram where the sender and receiver objects of a message are the same. In this transmission, if two events e and e' belong to the same object O_i, then the event e' immediately follows the event e with no other event in between, Figure 14 a) shows such an example.

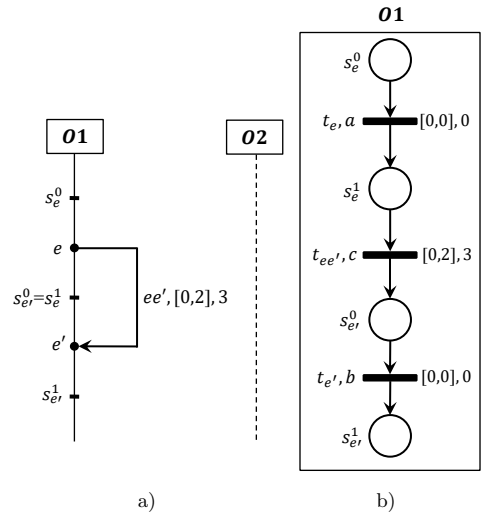


Figure 14. Intra-objects transmission and its corresponding DTPN

In the sequence diagram, sending and receiving actions at events e and e' are instantaneous and without timing constraints, $d1 = d2 = 0$, $ct1 = ct2 = [0, 0]$. The transmission action at event ee' has 3 units of time during the interval $[0, 2]$. In the following, we show how this is mapped to an equivalent DTPN.

1. In object O_1, the sending action at event e is translated to Petri net transition t_e without duration and timing constraint. Both states of this event e are translated to two Petri net places s_e^0 and s_e^1 with two arcs, each one relates a place to the transition t_e. Figure 15 a) shows this construction.

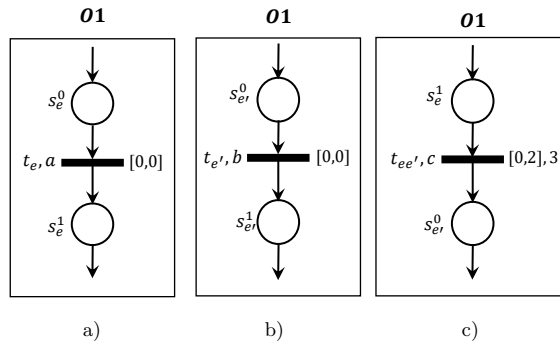


Figure 15. Elementary intra-objects transmission corresponding DTPNs

2. The receiving action at event e' in object O_1 has the same characteristics of the sending action at event e. A similar construction can be made by using two places $s^0_{e'}$ and $s^1_{e'}$, and a transition $t_{e'}$ with two arcs, each one relates a place to the transition $t_{e'}$ (see Figure 15 b)).
3. For modeling the transmission action operation between sending action and receiving action in a same object O_1, we add an event named ee' relating both events e and e' that takes in our case a duration between 3 and 5 units of time. For taking into account this characteristic we consider the constraint interval $[0, 2]$ and a duration of the action associated to the transition $t_{ee'}$ equal to 3. The corresponding DTPN in Figure 15 c). The transmission action at event ee' is translated a Petri net transition $t_{ee'}$ with a duration and a timing constraint. The two places of this transition are the output place s^1_e of transition t_e and the input place $s^0_{e'}$ of transition $t_{e'}$. In addition, two arcs are created, each one relates a place to the transition $t_{ee'}$.
4. For connecting transmission action operation with the two previous parts of intra-objects transmission, we will add two arcs, the first arc from the output place s^1_e of transition t_e to the transition $t_{ee'}$ and the second arc from the transition $t_{ee'}$ to the input place $s^0_{e'}$ of transition $t_{e'}$. The final corresponding DTPN is shown in Figure 14 b).

Formally, S and T are the least sets verifying the following construction conditions:

- Let $\{t_e, t_{ee'}, t_{e'}\} = \{\langle e, 0, [0, 0]\rangle, \langle ee', 3, [0, 2]\rangle, \langle e', 0, [0, 0]\rangle\}$ be a set of transitions modeling intra-objects transmission in O_1.
- Let $e, e' \in E_1$, be two events, such that $\lambda(e) =$ Activity and $\lambda(e') =$ Activity with $\forall(e, e') \in <$, $\{s^0_e, s^1_e, s^0_{e'}, s^1_{e'}\} \subseteq S$ and $\{t_e, t_{e'}, t_{ee'}\} \subseteq T$, then:

$$
\begin{aligned}
{}^\circ t_e &= \{s^0_e\};\\
t_e^\circ &= \{s^1_e\};\\
{}^\circ t_{e'} &= \{s^1_{e'}\};\\
t^\circ_{ee'} &= \{s^1_{e'}\};\\
{}^\circ t_{ee'} &= \{s^1_e\};\\
t^\circ_{ee'} &= \{s^0_{e'}\}.
\end{aligned}
$$

In the different cases of transmission, inter-objects and intra-objects each object in the sequence diagram has a single initial state and a single final state as defined as follows:

- A state s is said an initial iff ${}^\circ s = \emptyset$.
- A state s is said a final iff $s^\circ = \emptyset$.

3.3.2 Combined Fragments

Combined fragments are one important kind of Interactions, which are used to create interactions that are more complex. A combined fragment is defined by an interaction operator with its operands. The operands of a combined fragment can have guards on them as in alternative behavior (alt) and iterative behavior (loop). The guards of the operands are given by boolean expressions (boolean condition or temporal condition).

In UML MARTE profile, we can specify the time constraints on a combined fragment or on an operand using the tag *"execTime"* in the stereotype <<ResourceUsage>>. According to the MARTE SD specification context, the time constraint interval $[a, b]$ associated to this stereotype can take the following values: $\forall a, b \in R^*$

- if the execution of an operand or a combined fragment is urgent then the time interval $[a, b] = [0, 0]$ or $[a, b] = [t, t']$ with $t = t'$;
- if the execution of an operand or a combined fragment is not urgent then the time interval $[a, b] = [0, \infty[$;
- if execution of an operand or a combined fragment is specified with latency then the time interval $[a, b] = [v, v + t]$ with $a \geq 0$ and $a \leq b$.

In this subsection, we give the translation rules of the most used combined fragments such as weak sequencing behavior (seq), strict sequencing behavior (strict), parallel behavior (par), alternative behavior (alt), optional behavior (opt) and iterative behavior (loop). The combined fragment operands are transformed to DTPN subnets using previous translation rules and then integrated to the resulting DTPN.

Weak sequencing combined fragments.
Weak sequencing combined fragments defined by "seq" operator, contain two or more operands. It represent a weak sequencing between the behaviors of its operands. If no other operator is present on a diagram, then weak sequencing should be applied to the Interaction fragments. Figure 16 shows such an example.

1. The initial place $s^0_{f_i}$ with the initial marking is created $M(s^0_{f_i}) = 1$;
2. To synchronize all the operands that will be get involved in the execution of the weak interaction, the transition $t^0_{f_i}$ is created. An arc connecting the initial place $s^0_{f_i}$ to the transition $t^0_{f_i}$ is added;
3. Using the previous translation rules, the two DTPN subnets corresponding to the two operands in weak combined fragments are generated;
4. Since, the purpose of this operator is to allow the execution of only one operand, in other words, the operands are executed in mutual exclusion. In Petri net terms, the transitions related to the operands should be in conflict. So, the place named LP is created and connected to the two initial transitions of the two DTPN subnets;

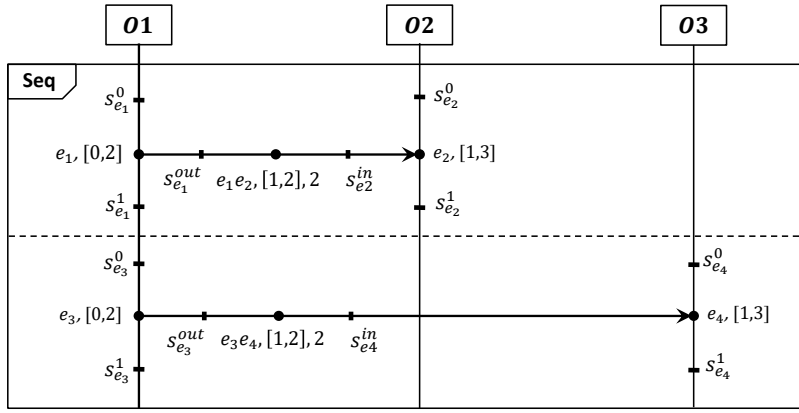


Figure 16. Weak sequencing combined fragments

5. The final transition $t^1_{f_i}$ is created and connected to the last place of every DTPN subnet;

6. The place $s^1_{f_i}$ corresponding to the final place of the DTPN is created and connected, as an output place, to the final transition $t^1_{f_i}$. The equivalent DTPN of weak sequencing combined fragments is given by Figure 17.

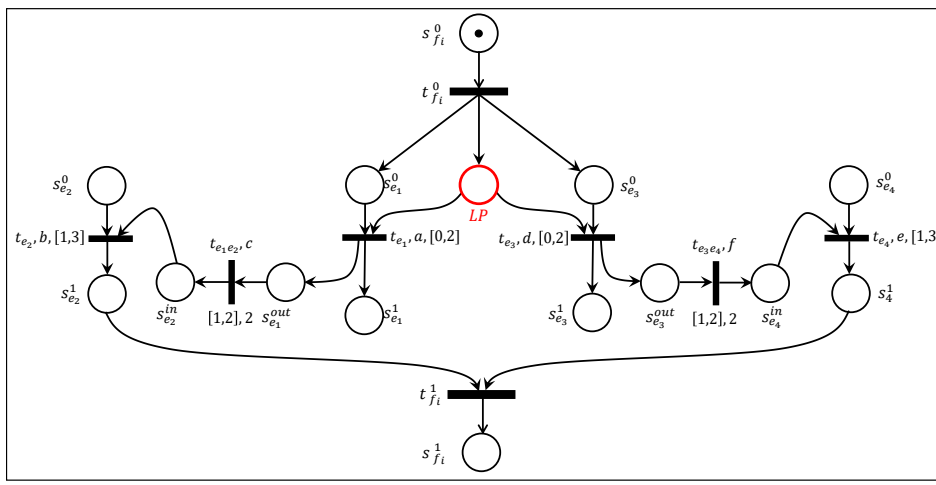


Figure 17. Weak sequencing combined fragments corresponding DTPN

Strict sequencing combined fragments.
Strict sequencing combined fragments are defined by "strict" operator, encloses two or more operands. The operands behaviors must occur in a given order. The order within each operand is preserved. This combined fragments is illustrated by Figure 18.

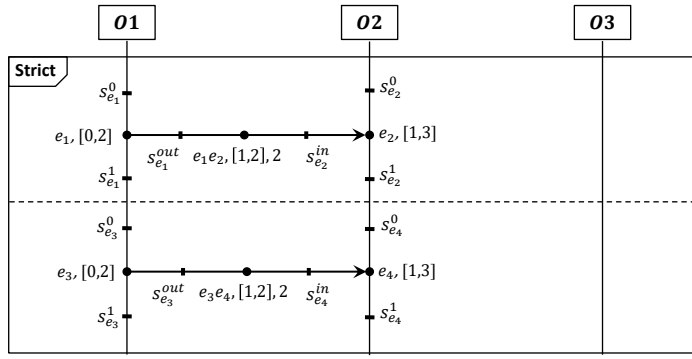


Figure 18. Strict sequencing combined fragments

1. The initial place $s^0_{f_i}$ with the initial marking is created $M(s^0_{f_i}) = 1$;
2. The initial transition $t^0_{f_i}$ is created and connected, as output transition to the initial place $s^0_{f_i}$;
3. Using the translation rules explained in the previous section, the two operands in strict combined fragments are translate to the two DTPN subnets. An arc connecting the last transition of the first DTPN subnet to the initial place of the second DTPN subnet is created;
4. The final transition $t^1_{f_i}$ of the strict combined fragments is generated and an arc connecting the final place of second subnet to the final transition $t^1_{f_i}$ of the strict fragment is added;
5. The final place $s^1_{f_i}$ of strict fragment is created and an arc connecting the transition $t^1_{f_i}$ to the place $s^1_{f_i}$ is added. Figure 19 illustrates the complete construction of the DTPN corresponding to the strict combined fragments.

Parallel combined fragments.
A combined fragment of "par" type is used to specify the concurrent behavior of real-time systems. It describes a parallel execution of the behaviors related to different operands. The behaviors of these operands can be interleaved in any way. However, each operand behavior preserves its predefined order. Figure 20 shows an example of parallel execution of elements *Operand-1* and *Operand-2* with a global time constraint associated to the execution parallel combined fragments. Object O_1

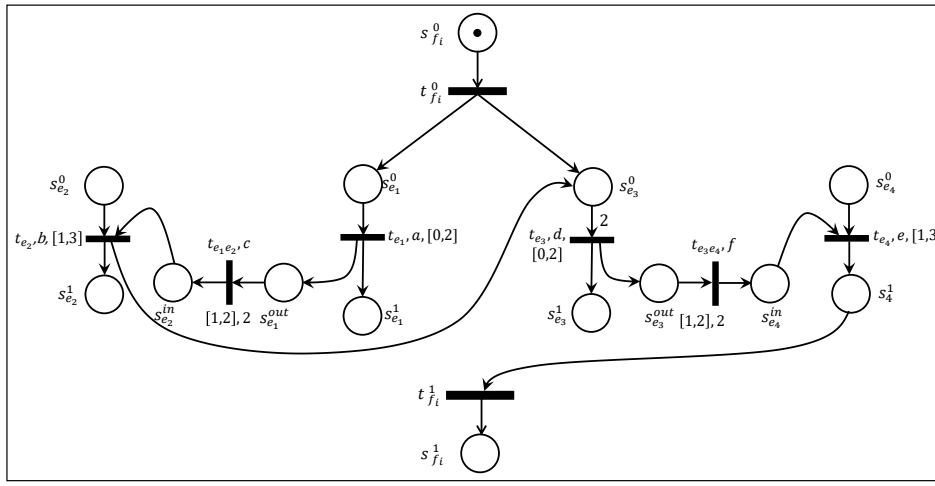


Figure 19. Strict sequencing combined fragments corresponding DTPN

launches concurrently two asynchronous transmission operations. Consequently, the corresponding receiving operations in object O_2 may happen concurrently too. The objects O_1 and O_2 can continue their execution according to the timing constraint of the parallel combined fragment.

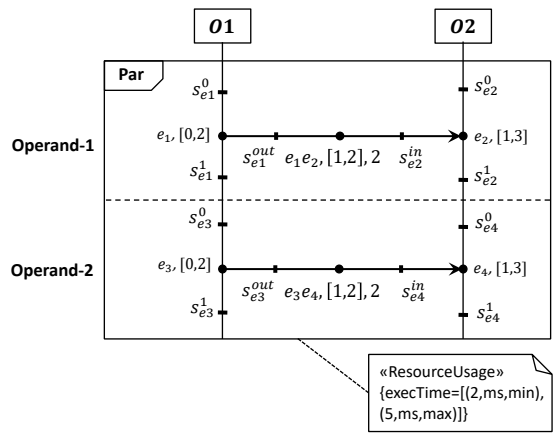


Figure 20. Parallel combined fragments

To detail the translation rules of this fragment type, let us consider the sequence diagram of Figure 20.

1. The initial place $s^0_{f_i}$ with the initial marking is created $M(s^0_{f_i}) = 1$. This marking place is used in the evaluation of the firing condition of the connected transitions;

2. For synchronizing all the operands involved in the parallel combined fragments execution, the transition $t^0_{f_i}$ is created. It is instantaneous and with zeroed time constraint ($d = 0, tc = 0$). An arc connecting the initial place $s^0_{f_i}$ to the transition $t^0_{f_i}$ is then added;
3. Using the translation rules explained in previous section two DTPN subnets $DTPN - 1$ and $DTPN - 2$ are generated, which correspond respectively to *Operand-1* and *Operand-2*. Note that places $s^0_{e_1}$ and $s^0_{e_3}$ have no marking since they become a non initial places ($M(s^0_{e_1}) = 0$, $M(s^0_{e_3}) = 0$). The starting transition $t^0_{f_i}$ is then connected to these places;
4. For synchronizing all operands involved in output of the parallel combined fragments, the final transition $t^1_{f_i}$ is created. It has the time interval $[2, 5]$ as a timing constraint. This transition is connected then, as output transition to each last place of every DTNP subnet ($s^1_{e_2}$, $s^1_{e_4}$). So, the final transition $t^1_{f_i}$ can only be fired when each operand in the combined fragments reaches its final state (one token in its final place);
5. The place $s^1_{f_i}$ corresponding to the final state of the parallel combined fragments is created. The final transition $t^1_{f_i}$ is then connected to this place. The resulting DTPN of this parallel combined fragments is given by Figure 21.

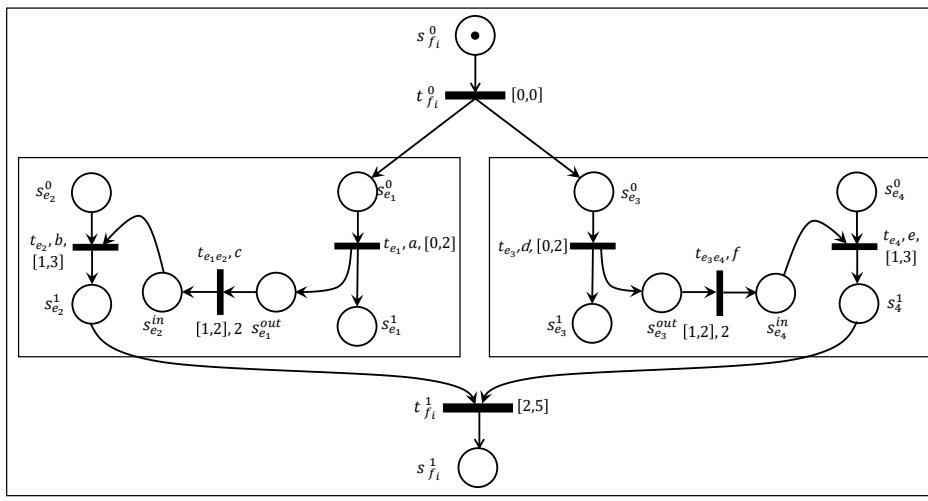


Figure 21. Parallel combined fragments corresponding DTPN

Alternative combined fragments.
Alternative combined fragments defined by "alt" operator, represents a choice of behavior in a fragment. This operator implements a non deterministic choice between

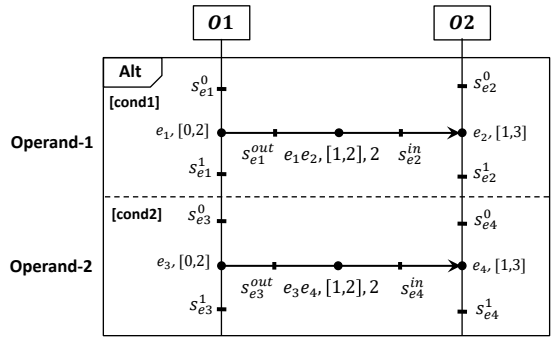


Figure 22. Alternative combined fragments

operands that have their constraints evaluated to true. Figure 22 shows an example of an alternative fragment with two operands.

1. The initial place $s^0_{f_i}$ with the initial marking is created $M(s^0_{f_i}) = 1$;
2. Two conflicting transitions $t^1_{f_i}$ and $t^2_{f_i}$ are generated. In this manner, that only one of the operands is selected, even the transitions guards are both evaluated as true. Two arcs connecting both transitions $t^1_{f_i}$ and $t^2_{f_i}$ to the initial place $s^0_{f_i}$ are added;
3. Using the translation rules explained in the previous section, two DTPN subnets $DTPN-1$ and $DTPN-2$ are generated, which correspond respectively to Operand-1 and Operand-2. The previous transitions $t^1_{f_i}$ and $t^2_{f_i}$ are connected by two arcs to the initials places $s^0_{e_1}$ and $s^0_{e_3}$, respectively, which launch the executions of internal operands (*Operand-1* and *Operand-2*);
4. For representing the output for each DTPN subnet, two final transitions $t^3_{f_i}$ and $t^4_{f_i}$ are generated. These transitions are then related, as an output transitions to the final places $s^1_{e_2}$ and $s^1_{e_4}$, which are associated to sub DTPN-1 and sub DTPN-2, respectively;
5. The final place $s^1_{f_i}$ of DTPN is created and connected to it as the output place to the two final transitions $t^3_{f_i}$ and $t^4_{f_i}$. Figure 23 shows the complete structure of the equivalent DTPN to the alternative combined fragments.

Optional combined fragments.
Optional combined fragments, defined by the operator "opt", contain only one operand which is executed according to a guard condition or a state value. Figure 24 a) gives an example illustrating this operator.

Optional combined fragments is semantically equivalent to the alternative combined fragments. So, it is translated as a simplification of alternative combined fragments with only one operand, considered where the condition evaluation is true. Figure 24 b) shows the corresponding DTPN specification.

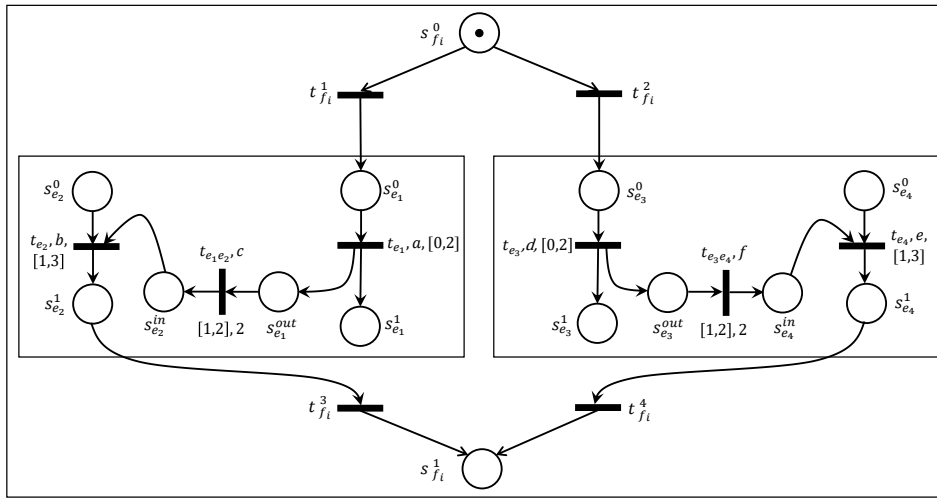


Figure 23. Alternative combined fragments corresponding DTPN

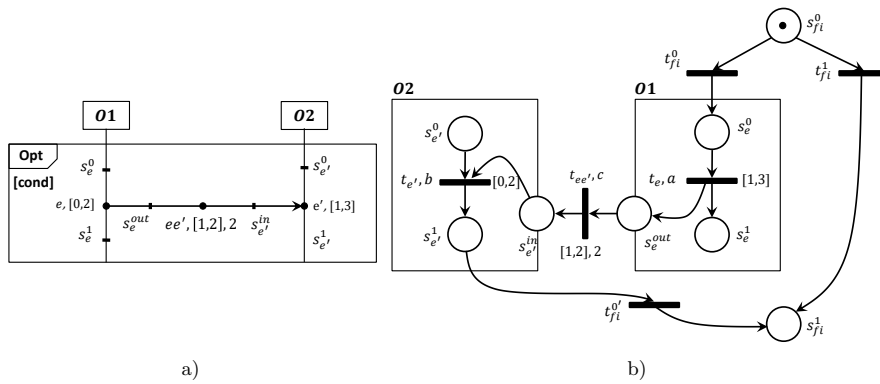


Figure 24. Optional combined fragments and its corresponding DTPN

We can see that in the resulting DTPN, the transition $t^0_{f_i}$ is possible only if a guard condition is true. Otherwise, the transition $t^1_{f_i}$ ends the behavior.

Loop combined fragments.
Iterative combined fragments, denoted by "loop" operator, have only one operand. It defines a recursive behaviour with a guard condition. The guard may either indicate a number of repetitions ($[min, max]$) that should be executed or a boolean expression. Figure 25 a) shows an example representing a loop combined fragments.

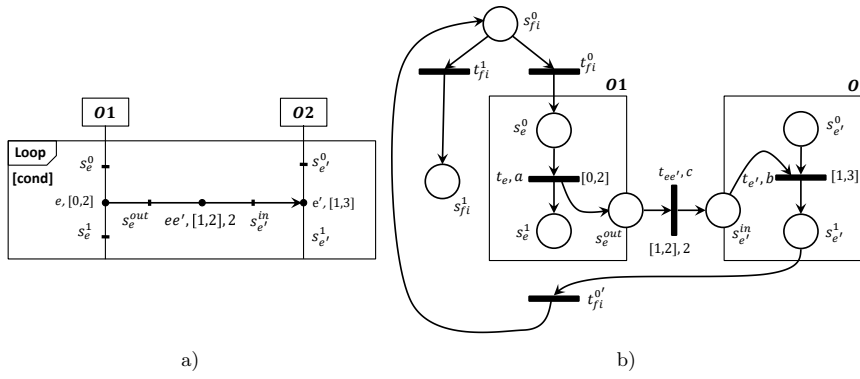


Figure 25. Loop combined fragments and its corresponding DTPN

The translation process follows the following steps:

1. The initial place $s^0_{f_i}$ with the initial marking is created $M(s^0_{f_i}) = 1$;
2. Two initial transitions $t^0_{f_i}$ and $t^1_{f_i}$ are generated. These transitions allow the guard evaluation that serves as a constraint for the combined fragments;
3. Two arcs connecting the initial place $s^0_{f_i}$ to both transitions $t^0_{f_i}$ and $t^1_{f_i}$ are added;
4. Using the translation rules detailed in the previous section, the DTPN subnet corresponding to the loop operand is created;
5. Since, the purpose of this operator is to repeat the execution of the loop operand until the loop guard is evaluated to false, the transition $t^{0'}_{f_i}$ is created and connected to the initial place $s^0_{f_i}$ of the loop fragment. The final place $s^1_{e'}$ of DTPN subnet is then connected to the transition $t^{0'}_{f_i}$;
6. For considering the case when the boolean expression associated to the transition $t^1_{f_i}$ is evaluated to false, the final place $s^1_{f_i}$ is created and connected to the transition $t^1_{f_i}$ as an output place. Figure 25 represents the resulting DTPN structure of the loop combined fragments in Figure 25 b).

4 CASE STUDY

In order to illustrate our approach, for modeling and analysis of real-time embedded systems, we propose to use an interaction fragment of a real case study which is an elevator controller system. We consider two elevators used in a building composed of many floors. The interest of this application is to consider the case of two parallel activities with a non-null duration. These activities correspond to the elevators moves between two different floors at the same time. The elevators are controlled by only one elevator controller system.

Specification.
Assume that two users are at two different floors. At a given instant, each one requests an elevator. For instance, *User-1* is at the first floor while *User-2* is at the seventh floor. It is also assumed that *User-1* wants to go to the seventh floor, and *User-2* wants to go to the first floor.

Since our study focused on parallel activities modeling, we assume that the elevators are parked at floor one and seven, respectively, where their doors are open and the users are inside them. Both users pushed simultaneously the floor button to be visited in the button panel. The elevator controller must move the two elevators in parallel to the requested floors.

To simplify the study, we abstract from elevator system components participating in the interaction which does not affect the study purpose, like *cabin*, *floor sensor* and *door*.

Once the elevator controller has received two requests at the same time, it creates and activates two processes in parallel *Elevator1* and *Elevator2* (two instances). *Elevator1* and *Elevator2* are responsible for the move of the first elevator and the second elevator, respectively.

Two time aspects are taken into consideration, the first one concerns the execution duration of actions and the second one concerns the timing constraints. In the example, the elevator movement action takes 3 units of time ($d = 3\,000$ ms) and the action execution may be delayed and executed in the time interval $[0, 10]$.

Figure 26 shows the sequence diagram representing the time aspects and the interactions between the principal objects involved in the elevator request functionality, with a global time constraint ($tc = [10, 30]$) associated.

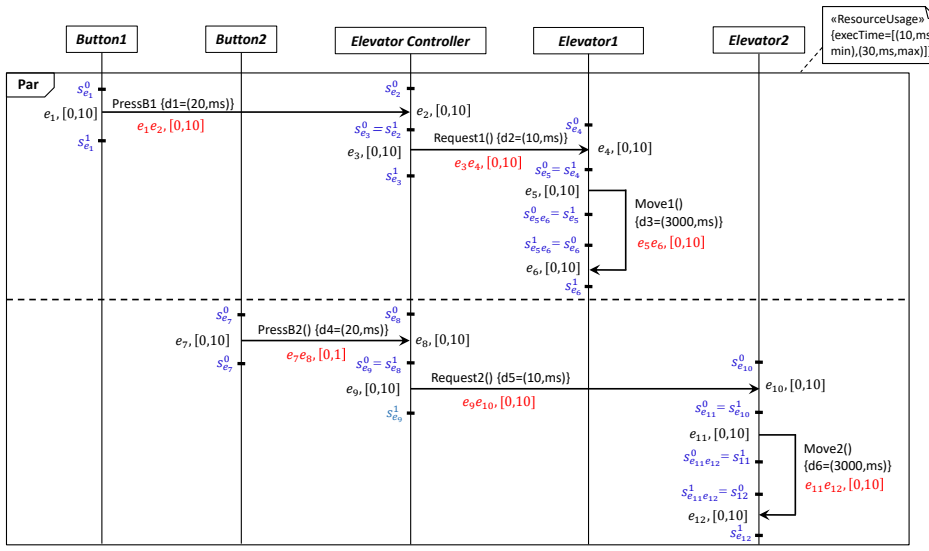


Figure 26. Interactions between objects for the request an elevator

To apply the translation method over the previous sequence diagram, first, we annotate the sending and receiving time events of the transmitted messages between objects. The set of Time events is $E^t = \{e_1, e_2, \ldots, e_{10}, e_{12}\}$. Events representing the intermediate transmissions (colored by red) with their associated duration and timing constraints are also added. Hence, $E^t = \{e_1, e_2, \ldots, e_{11}, e_{12}\} \cup \{e_1e_2, e_3e_4, e_5e_6, \ldots, e_{11}e_{12}\}$. For each object involved in the interaction, two state (colored by blue) to each event on its lifeline is associated, then $S = \{s^0_{e_1}, s^1_{e_1}, s^0_{e_2}, \ldots, s^1_{e_{12}}\}$. Similarly, states are associated to timed events. They correspond to intermediate transmissions in the sequence diagram. Hence, $S = \{s^0_{e_1}, s^1_{e_1}, \ldots, s^0_{e_{12}}, s^1_{e_{12}}\} \cup \{s^{out}_{e_1}, s^{in}_{e_2}, \ldots, s^{in}_{e_{12}}\}$.

The translations method detailed in Section 3 can now be applied on the sequence diagram. Figure 27 depicts the complete construction of the equivalent DTPN.

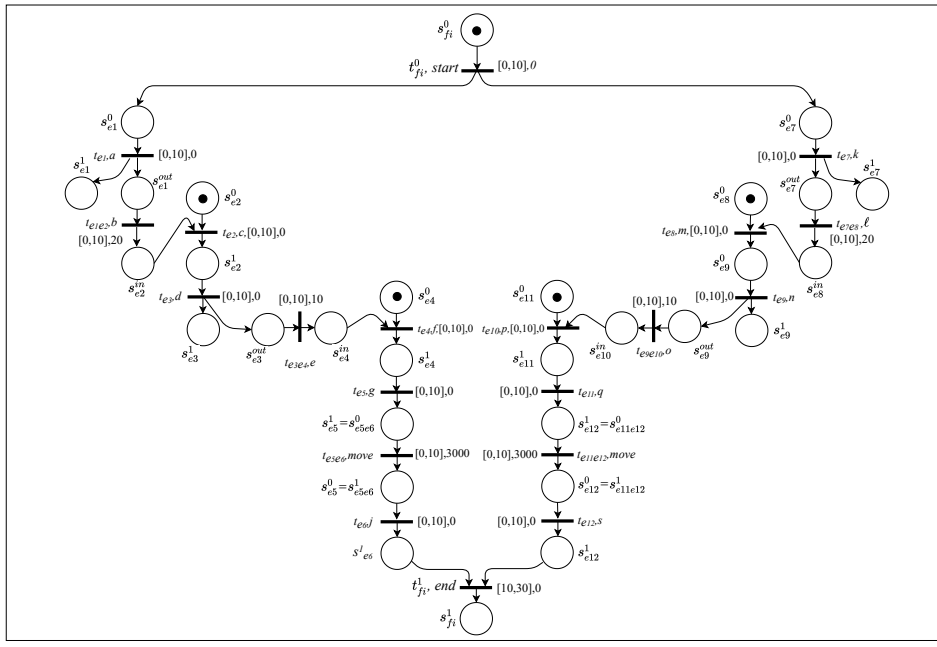


Figure 27. DTPN specification corresponding to sequence diagram for requesting an elevator

According to Figure 27, the initial place of the DTPN is $s^0_{f_i}$ with the initial marking $M(s^0_{f_i}) = 1$. The final place is $s^1_{f_i}$. The initial transition is $t^0_{f_i}$ with time constraint $[0, 0]$. The final transition is $t^1_{f_i}$ with time constraint $[2, 5]$. The places corresponding to the states are $P = \{s^0_{e_1}, s^1_{e_1}, s^0_{e_2}, \ldots, s^1_{e_{12}}, s^{out}_{e_1}, s^{in}_{e_2}, \ldots, s^{in}_{e_{12}}\}$ and the events corresponding to the Petri net transitions are $T = \{t_{e_1}, t_{e_2}, \ldots, t_{e_{11}}, t_{e_{12}}\}$.

In the resulting DTPN, the firing of transitions is bound to a time interval. The transition firing represents action launching which has an explicit duration.

Verification.
To investigate the verification and validation of real time embedded systems, we opt for the operational semantics developed in [20] to generate a semantics model to DTPN specification. The true-concurrency semantics based DTPN model is used to obtain the durational action Timed Automata (daTA) depicted in Figure 28. The resulting daTA has 42 states and 68 transitions; it is generated automatically using a tool integrated into FOCOVE environment [45]. For this reason, we represent only a fragment of the graph. This structure allows the verification of properties related to parallel evolution of actions as shown in the different states of resulting daTA. As an example, state $s36$ is labelled within duration conditions set $\{x \geq 3\,000\,\text{ms}, y \geq 3\,000\,\text{ms}\}$. In this state, actions $Move1$ and $Move2$ can comply in parallel, and each one can finish only if its clock reaches a value equal to its duration.

Note that we have used just two clocks (x and y) to specify all actions of the case study because we have at most two actions in execution at a given time. This is possible due to the dynamic creation of clocks with the reuse of free clocks.

The model checking is mainly based upon the region graph and zone graph algorithms on daTA. The model checking complexity on TA, as daTA, is exponential in the number of clocks. We can observe that using true-concurrency semantics based DTPN, the generated daTA has a lower number of clocks. We can then apply CTL model checking to check some properties.

5 DISCUSSION AND RELATED WORKS

The transformation of MARTE sequence diagrams to formal specifications, for the formal verification, has been investigated in several approaches like [46, 47, 48, 49, 50, 51], but a few approaches as [12, 52, 53] have taken into consideration time specification in the transformation process. Previous works just deal with the flowing of events in sequence diagrams with implicit expression of time and consider only interleaving semantics. On the contrary, we have proposed a translation approach that supports at the same time timing constraints, explicit actions durations, urgency and structural and temporal non-atomicity of actions. Thus, our approach is done to a true concurrency-based real time formal specification models (DTPN and daTA models). The use of daTA's structures as semantics allows firstly, to express concurrent and parallel behaviors in a natural way, i.e. to distinguish between sequential and parallel runs of actions. Therefore, with non-null duration under timing constraints. This is not the case of the interleaving semantics. Secondly, resetting the state clocks which are used to specify time and timing constraints. Therefore, this will lead to reducing the verification time which is often exponential for large size RTES, and to reducing the number of states and transitions in the graph without loss of information, and then escaping from the explosion of the underlying graph.

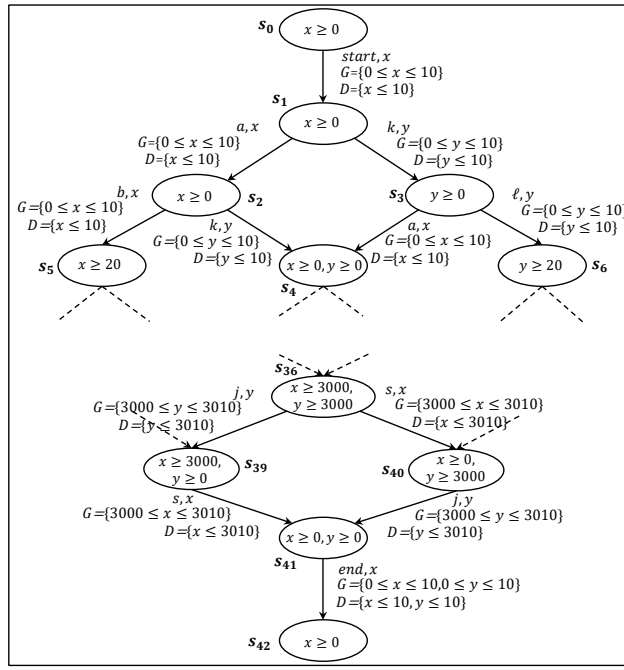


Figure 28. Fragment of daTA corresponding to DTPN

In the proposed method, we borrowed the idea of associating states to events of the sequence diagram as made in [41]. However, the translation method of [41] considers the sequence diagram as a whole entity, in comparison with our translation method, as a set of rules defined to make the translation method inductive. The translation rules start by considering elementary component, which are the events in the sequence diagram. Hence, for each composed component, a translation rule is defined inductively using the translation result of its sub-components. In this manner, a construction of a tool implementing the method may easily be done. As an example, Figure 29 a) shows the asynchronous message events that represent the moments in which the actions send or receive. In terms of the Petri net, each event (e, e', e'') is translated to a transition, an input place and an output place as shown in Figure 29 b).

In [12], authors interpreted parallel activities, modeled in MARTE sequence diagram, by parallel transitions in TCPNIA specification under an interleaving semantics. In this approach, only the execution occurrence duration is modeled. It is specified by a time interval associated to a transition of TCPNIA. In our proposed method, start occurrence, finish occurrence, message occurrence (complete UML name, message occurrence specification), which represent sending and receiving event, and invoking or receiving of operation calls, are considered.

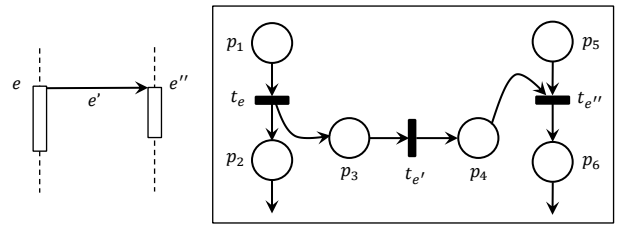


Figure 29. Asynchronous message translating

On the other hand, the proposed works assumed action atomicity hypothesis imposed by the interleaving semantics which handle parallel behaviors as their combined sequential evolution. For more clarification, let us consider the example of Figure 30 b). Using the method proposed in [12], these parallel activities are translated to the TCPNIA specification of Figure 30 c).

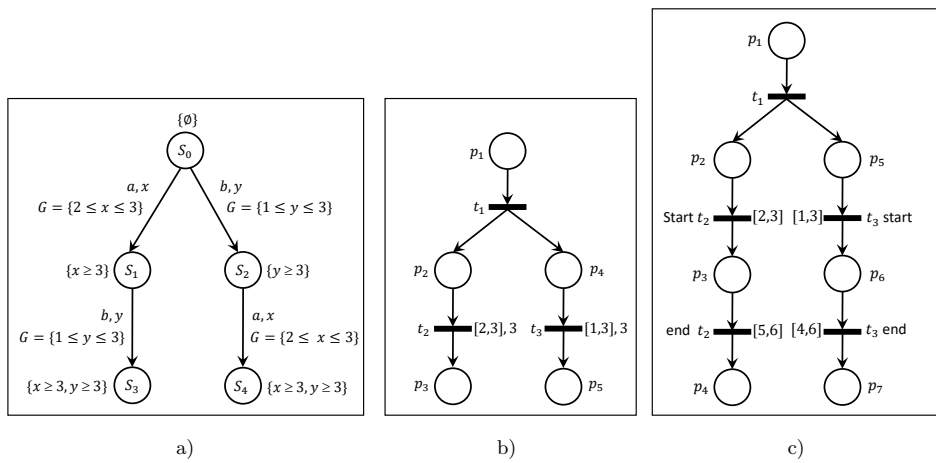


Figure 30. Generating models

For the verification of the required properties, the specification of Figure 30 c) is translated to a Timed Automata. Since the transition execution is instantaneous, there is no way to observe the parallel execution of the two activities. As a solution for such situation, it is possible to interpret each activity having a non-null duration by two sequential transitions modeling the start and the end of the activity. As shown in Figure 30 c), the activity duration is captured in the intermediate state conditioning the execution of the ending transition by the elapsed time. We notice that the composed system may be in the state where the two start transitions are executed before executing the end transitions. Such state captures the parallelism

between the two activities however, such methods augment the number of states, transitions and clocks, which contributes to the state space explosion problem of zone graph corresponding to the Timed Automata specification. As an alternative, the use of DTPN and daTA is an interesting solution (Figure 30 a)).

As a consequence, we remark that the number of transitions of each structure is comparable respectively of those of Figure 30 a). Another advantage concerns the construction of the set of clocks. In our context, a clock is created dynamically during the generation of the semantics models with the reuse of free clocks. On the contrary, other models like Timed Automata, Petri Nets with Deadlines and Time Petri Nets manage [15, 54, 55, 56], at the beginning of modeling, a finite and constant number of clocks recording to the number of actions to execute.

6 CONCLUSION

In this paper, we proposed an operational method for translating MARTE SD specifications to DTPN specifications. As it has been mentioned previously, MARTE SD allows the specification of several kind of behaviors like concurrency, time constraints and action duration. Since, DTPN formal specification model is a true concurrency based semantics, it allows the consideration of the last three behavior characteristics in both syntactic and semantics levels. This latter arguments is the sole motivation of our work. The use of daTA structures as semantics allows properties formal verification, particularly those related to parallel evolution of actions that have non-null duration and under timing constraints. Properties related to reachability may be checked by means of KRONOS and UPPAAL tools, and properties dealing with true concurrency behaviours may be checked using FOCOVE model checker.

In this paper, the translation method has been explained by considering several examples. As for the perspectives of this work, we suggest that it is applied on realistic real-time embedded systems. It could be either integrated to an existing environment system and/or a computer-aided software engineering model checker. It could also be part of a separate formal specification and verification tool. Alternatively, it would be useful to develop a full TCTL model-checker for DTPNs related to MARTE SD specification without passing by Timed Automaton like structures.

REFERENCES

[1] Lee, I.—Leung, J. T.—Son, S.: Handbook of Real-Time and Embedded Systems. New York, Chapman and Hall/CRC, 2008, doi: 10.1201/9781420011746.

[2] Gomes, L.—Fernandes, J. M.: Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation. Information Science Reference, 2010.

[3] Rayner, M.—Hockey, B. A.—Chatzichrisafis, N.—Farrell, K.: OMG Unified Modeling Language Specification. Version 1.3, © 1999 Object Management Group, Inc., 2005.

[4] Gérard, S.: MARTE: A New Standard for Modeling and Analysis of Real-Time and Embedded Systems. Proceedings of Euromicro Conference on Real-Time Systems (ECRTS '07), Pisa, Italy, 2007.

[5] OMG: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, OMG Document Number: Ptc. 2008.

[6] Ge, N.—Pantel, M.—Crégut, X.: Time Properties Dedicated Transformation from UML-MARTE Activity to Time Transition System. ACM SIGSOFT Software Engineering Notes, Vol. 37, 2012, No. 4, pp. 1–8, doi: 10.1145/2237796.2237807.

[7] Ge, M. N.—Cregut, X.: A Framework Dedicated to Time Properties Verification for UML-MARTE Specifications. 2012.

[8] Berthomieu, B.—Vernadat, F.: Time Petri Nets Analysis with TINA. Third International Conference on the Quantitative Evaluation of Systems (QEST '06), 2006, pp. 123–124, doi: 10.1109/QEST.2006.56.

[9] Bošnački, D.—Dams, D.: Integrating Real Time into Spin: A Prototype Implementation. In: Budkowski, S., Cavalli, A., Najm, E. (Eds.): Formal Description Techniques and Protocol Specification, Testing and Verification (PSTV 1998, FORTE 1998). Springer, Boston, MA, IFIP – The International Federation for Information Processing, Vol. 6, 1998, pp. 423–438, doi: 10.1007/978-0-387-35394-4_26.

[10] Bošnački, D.—Dams, D.: Discrete-Time Promela and Spin. In: Ravn, A. P., Rischel, H. (Eds.): Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1998). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1486, 1998, pp. 307–310, doi: 10.1007/bfb0055359.

[11] Holzmann, G. J.: The Model Checker SPIN. IEEE Transactions on Software Engineering, Vol. 23, 1997, No. 5, pp. 279–295, doi: 10.1109/32.588521.

[12] Yang, N.—Yu, H.—Sun, H.—Qian, Z.: Modeling UML Sequence Diagrams Using Extended Petri Nets. Telecommunication Systems, Vol. 51, 2012, No. 2-3, pp. 147-158, doi: 10.1007/s11235-011-9424-5.

[13] Yang, N.-H.—Yu, H.-Q.: Modeling and Verification of Embedded Systems Using Timed Colored Petri Net with Inhibitor Arcs. Journal of East China University of Science and Technology, Vol. 36, 2010, No. 3, pp. 411–417.

[14] Alur, R.—Dill, D.: Automata for Modeling Real-Time Systems. In: Paterson, M. S. (Ed.): Automata, Languages, and Programming (ICALP 1990). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 443, 1990, pp. 322–335, doi: 10.1007/bfb0032042.

[15] Alur, R.—Dill, D. L.: A Theory of Timed Automata. Theoretical Computer Science, Vol. 126, 1994, No. 2, pp. 183–235, doi: 10.1016/0304-3975(94)90010-8.

[16] McMillan, K. L.: Symbolic Model Checking. Kluwer Academic Publishers, 1993, doi: 10.1007/978-1-4615-3190-6.

[17] Courcoubetis, C.—Yannakakis, M.: Minimum and Maximum Delay Problems in Real-Time Systems. Formal Methods in System Design, Vol. 1, 1992, No. 4, pp. 385–415, doi: 10.1007/bf00709157.

[18] Gardey, G.—Roux, O. H.—Roux, O. F.: State Space Computation and Analysis of Time Petri Nets. Theory and Practice of Logic Programming, Vol. 6, 2006, No. 3, pp. 301–320, doi: 10.1017/s147106840600264x.

[19] Bouyer, P.—Fahrenberg, U.—Larsen, K. G.—Markey, N.—Ouaknine, J.—Worrell, J.: Model Checking Real-Time Systems. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (Eds.): Handbook of Model Checking. Springer, Cham, 2018, pp. 1001–1046, doi: 10.1007/978-3-319-10575-8_29.

[20] Belala, N.—Saidouni, D. E.—Boukharrou, R.—Chaouche, A. C.—Seraoui, A.—Chachoua, A.: Time Petri Nets with Action Duration: A True Concurrency Real-Time Model. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), Vol. 4, 2013, No. 2, pp. 62–83, doi: 10.4018/jertcs.2013040104.

[21] Saidouni, D. E.—Belala, N.: Actions Duration in Timed Models. International Arab Conference on Information Technology (ACIT), 2006.

[22] Micskei, Z.—Waeselynck, H.: UML 2.0 Sequence Diagrams' Semantics. LAAS Technical Report No. 08389, 37, 2008.

[23] OMG: UML Profile for Schedulability. Performance, and Time Specification, Vol. 1, 2005.

[24] Siegel, J. M.: Model Driven Architecture (MDA), MDA Guide Rev. 2.0. Object Management Group, Tech. Rep. ORMSC/14-06-0, 2014.

[25] OMG Unified Modeling Language™ (OMG UML). 2013.

[26] André, C.: MARTE Time and Time Constraints Models and Their Applications. 2011.

[27] Merlin, P.: A Study of the Recoverability of Computer Systems. Ph.D. Thesis, Computer Science Department, University of California, 1974.

[28] Ramchandani, C.: Analysis of Asynchronous Concurrent Systems by Petri Nets (No. MAC-TR-120). Massachusetts Institute of Technology, Cambridge Project Mac, 1974.

[29] Sifakis, J.: Use of Petri Nets for Performance Evaluation in Measuring Modelling and Evaluating Computer Systems. 1977.

[30] Alur, R.: Timed Automata. In: Halbwachs, N., Peled, D. (Eds.): Computer Aided Verification (CAV 1999). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1633, 1999, pp. 8–22, doi: 10.1007/3-540-48683-6_3.

[31] Belala, N.: Modèles de Temps et Leur Intérêt à la Vérification Formelle des Systèmes Temps-Réel. Doctoral Dissertation. 2010, doi: 10.13140/RG.2.2.25932.21129.

[32] Störrle, H.: Trace Semantics of Interactions in UML 2.0. Journal of Visual Languages and Computing, 2004.

[33] Cavarra, A.—Küster-Filipe, J.: Formalizing Liveness-Enriched Sequence Diagrams Using ASMs. In: Zimmermann, W., Thalheim, B. (Eds.): Abstract State Machines 2004. Advances in Theory and Practice (ASM 2004). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3052, 2004, pp. 62–77, doi: 10.1007/978-3-540-24773-9_6.

[34] Cengarle, M. V.—Knapp, A.: UML 2.0 Interactions: Semantics and Refinement. Proceedings of the 3rd International Workshop Critical Systems Development with UML (CSDUML'04), 2004, pp. 85–99.

[35] Eichner, C.—Fleischhack, H.—Meyer, R.—Schrimpf, U.—Stehno, C.: Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. In: Prinz, A., Reed, R., Reed, J. (Eds.): SDL 2005: Model Driven. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3530, 2005, pp. 133–148, doi: 10.1007/11506843_9.

[36] Haugen, Ø.—Husa, K. E.—Runde, R. K.—Stølen, K.: STAIRS Towards Formal Design with Sequence Diagrams. Software and Systems Modeling, Vol. 4, 2005, No. 4, Art. No. 355, doi: 10.1007/s10270-005-0087-0.

[37] Küster-Filipe, J.: Modelling Concurrent Interactions. Theoretical Computer Science, Vol. 351, 2006, No. 2, pp. 203–220, doi: 10.1016/j.tcs.2005.09.068.

[38] Hammal, Y.: Branching Time Semantics for UML 2.0 Sequence Diagrams. In: Najm, E., Pradat-Peyre, J. F., Donzeau-Gouge, V. V. (Eds.): Formal Techniques for Networked and Distributed Systems (FORTE 2006). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4229, 2006, pp. 259–274, doi: 10.1007/11888116_20.

[39] Fernandes, J. M.—Tjell, S.—Jorgensen, J. B.—Ribeiro, O.: Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. Sixth International Workshop on Scenarios and State Machines (SCESM'07: ICSE Workshops 2007), IEEE, 2007, doi: 10.1109/scesm.2007.1.

[40] Harel, D.—Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. Software and Systems Modeling, Vol. 7, 2008, No. 2, pp. 237–252, doi: 10.1007/s10270-007-0054-z.

[41] Bowles, J.—Meedeniya, D.: Formal Transformation from Sequence Diagrams to Coloured Petri Nets. 2010 Asia Pacific Software Engineering Conference, IEEE, 2010, pp. 216–225, doi: 10.1109/apsec.2010.33.

[42] Bouneb, M.—Saidouni, D. E.—Ilie, J. M.: Hierarchical System Design Using Refinable Recursive Petri Net. Computing and Informatics, Vol. 37, 2018, No. 3, pp. 635–655, doi: 10.4149/cai_2018_3_635.

[43] Daws, C.—Olivero, A.—Tripakis, S.—Yovine, S.: The Tool KRONOS. In: Alur, R., Henzinger, T. A., Sontag, E. D. (Eds.): Hybrid Systems III (HS 1995). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1066, 1995, pp. 208–219, doi: 10.1007/bfb0020947.

[44] Larsen, K. G.—Pettersson, P.—Yi, W.: UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer, Vol. 1, 1997, No. 1-2, pp. 134–152, doi: 10.1007/s100090050010.

[45] Saïdouni, D. E.—Benamira, A.—Belala, N.—Arfi, F.: FOCOVE: Formal Concurrency Verification Environment for Complex Systems. AIP Conference Proceedings, Vol. 1019, 2008, pp. 375–380, doi: 10.1063/1.2953008.

[46] Ejnioui, A.—Otero, C. E.—Qureshi, A. A.: Formal Semantics of Interactions in Sequence Diagrams for Embedded Software. 2013 IEEE Conference on Open Systems (ICOS), Kuching, Malaysia, 2013, pp. 106–111, doi: 10.1109/icos.2013.6735057.

[47] Saputra, A. B.—Basuki, T. A.—Tirtawangsa, J.: Transformation of UML 2.0 Sequence Diagram into Coloured Petri Nets. 2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA), IEEE, 2014, pp. 243–248, doi: 10.1109/icaicta.2014.7005948.

[48] Meedeniya, D.—Bowles, J.—Perera, I.: SD2CPN: A Model Transformation Tool for Software Design Models. 2014 International Computer Science and Engineering Conference (ICSEC), IEEE, 2014, pp. 354–359, doi: 10.1109/icsec.2014.6978222.

[49] Zafar, N. A.: Formal Specification and Verification of Few Combined Fragments of UML Sequence Diagram. Arabian Journal for Science and Engineering, Vol. 41, 2016, No. 8, pp. 2975–2986, doi: 10.1007/s13369-015-1999-9.

[50] Meedeniya, D.—Perera, I.—Bowles, J.: Tool Support for Transforming Unified Modelling Language Sequence Diagram to Coloured Petri Nets. Maejo International Journal of Science and Technology, Vol. 10, 2016, No. 3, pp. 272–283.

[51] Soares, J. A. C.—Lima, B.—Faria, J. P.: Automatic Model Transformation from UML Sequence Diagrams to Coloured Petri Nets. Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development – Volume 1: AMARETTO, 2018, pp. 668–679, doi: 10.5220/0006731806680679.

[52] Menad, N.—Dhaussy, P.—Drey, Z.—Mekki, R.: Towards a Transformation Approach of Timed UML MARTE Specifications for Observer-Based Formal Verification. Computing and Informatics, Vol. 35, 2016, No. 2, pp. 338–368.

[53] Andrade, V. C.—Peres, L. M.—Del Fabro, M. D.: Handling Global and Local Time and Energy Constraints of Sequence Diagrams. 2018 UKSim-AMSS 20th International Conference on Computer Modelling and Simulation (UKSim), IEEE, 2018, pp. 73–78, doi: 10.1109/uksim.2018.00025.

[54] Laroussinie, F.—Markey, N.—Schnoebelen, P.: Model Checking Timed Automata with One or Two Clocks. In: Gardner, P., Yoshida, N. (Eds.): CONCUR 2004 – Concurrency Theory. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3170, 2004, pp. 387–401, doi: 10.1007/978-3-540-28644-8_25.

[55] Cassez, F.—Roux, O. H.: Structural Translation from Time Petri Nets to Timed Automata. Journal of Systems and Software, Vol. 79, 2006, No. 10, pp. 1456–1468, doi: 10.1016/j.jss.2005.12.021.

[56] D'Aprile, D.—Donatelli, S.—Sangnier, A.—Sproston, J.: From Time Petri Nets to Timed Automata: An Untimed Approach. In: Grumberg, O., Huth, M. (Eds.): Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4424, 2007, pp. 216–230, doi: 10.1007/978-3-540-71209-1_18.

Nadia Chabbat received her B.Eng. degree from the University of Badji Mokhtar, Annaba, Algeria in 2005. In July 2014, she received her M.Sc. degree in computer science from the University of Badji Mokhtar, Annaba, Algeria. Her research domain is formal specification and verification of real-time embedded systems.

Djamel Eddine Saidouni received his B.Eng. degree from the University of Mentouri, Constantine, Algeria, in 1990. He received his Ph.D. in theoretical computer science from the University of Paul Sabatier, Toulouse, France in 1996. His domain research is the formal specification and verification of complex distributed and real time systems.

Radja Boukharrou is currently Associate Professor in the Faculty of New Technologies of Information and Communication at University of Constantine 2, Algeria. She holds her Ph.D. in computer science from Constantine 2 University. Her current research interests include formal modeling and verification, security and privacy in IoT systems, blockchain technology.

Salim Ghanemi graduated as Computer Science Engineer in June 1981 from Constantine University, Algeria. In December 1982, he received his Master degree in computer science from Aston University, Birmingham, England. In November 1987, he publicly discussed his Ph.D. research in the parallel and distributed programming at Loughborough University, Loughborough, England. From September 1988 till now, he has assumed several teaching and supervising research positions at many universities: Badji Mokhtar University, Annaba, Algeria, Philadelphia University, Amman, Jordan and King Saud University at Riyadh, Kingdom of Saudi Arabia. He has many scientific publications on several topics. In his main research focus is parallel programming, image processing, real time processing and formal verification. At present, he is the Head of a research team working on parallel processing on SoC, a project group attached to the Embedded Systems Laboratory at Badji Mokhtar Annaba, LASE. He occupied several administrative duties such as the Head of the Computer Science Department and the Vice Dean of the Engineering Science Faculty.