Timed-pNets: A Communication Behavioural Semantic Model for Distributed Systems (extended version)

Yanwen Chen, Yixiang Chen, Eric Madelaine

Timed-pNets: A Communication Behavioural Semantic Model For Distributed Systems (Extended Version)

Yanwen CHEN, Yixiang CHEN, Eric MADELAINE

RESEARCH REPORT
N° 8526
May 2014
Common Project-Team Scale

ISSN 0249-6399   ISRN INRIA/RR--8526--FR+ENG

Timed-pNets: A Communication Behavioural Semantic Model For Distributed Systems (Extended Version)

Yanwen CHEN, Yixiang CHEN, Eric MADELAINE

Common Project-Team Scale

Research Report n° 8526 — May 2014 — 31 pages

Abstract: This paper presents an approach to build a communication behavioural semantic model for heterogeneous distributed systems that include synchronous and asynchronous communications. Since each node of such system has its own physical clock, it brings the challenges of correctly specifying the system's time constraints. Based on the logical clocks proposed by Lamport and CCSL proposed by Aoste team in INRIA as well as pNets from Oasis team in INRIA, we develop timed-pNets to model communication behaviour for distributed systems. Timed-pNets are tree style hierarchical structures. Each node is associated with a timed specification which consists of a set of logical clocks and some relations on clocks. The leaves are represented by timed-pLTSs and non-leaf nodes are represented by timed-pNets including some holes which are filled by leaves or non-leaf nodes. Both timed-pLTSs and timed-pNets node can be translated to timed specifications. All these notions and methods are illustrated on a simple use-case of car insertion from the area of Intelligent Transportation Systems (ITS) and then TimeSquare tool is used to simulate and check the validity of our model.

Key-words: Formal methods, Heterogeneous distributed systems, Synchronous and asynchronous communications, Timed models, ITS

SCALE is a common team between the I3S Lab (Univ. of Nice Sophia-Antipolis and CNRS) and INRIA Sophia Antipolis Méditerranée

Timed-pNets: un modéle sémantique comportemental pour les systèmes distribués hétérogènes

Résumé : Cet article présente une nouvelle approche pour définir un modèle sémantique comportemental pour des systèmes distribués comportant des communications aussi bien synchrones qu'asynchrones. Chaque site dans ce genre de système ayant sa propre horloge, définir correctement les contraintes temporelles globales du systéme est un défi. À partir des concepts d'horloges virtuelles de Lamport, du langage CCSL introduit par l'équipe AOSTE d'INRIA, et du modèle pNets de l'équipe OASIS, nous développons notre modèle Timed-pNets pour exprimer les comportements et la communication de ces systèmes distribués. Les Timed-pNets sont des structures hiérarchiques arborescentes. À chaque noeud est associée une *spécification temporelle* composée d'un ensemble d'horloges et de relations entre ces horloges. Les noeuds feuilles sont representés par des Timed-pLTSs (systèmes de transitions paramétrés temporisés), et les autres noeuds sont soit recursivement des Timed-pNets, soit des trous (Holes) destinés à être remplis ultérieurement par des Timed-pNets. Nous définissons des algorithmes permettant de synthétiser la spécification temporelle des Timed-pLTSs et des Timed-pNets. Toutes ces notions sont illustrées sur un exemple de conduite automatisée de véhicules, issue du monde des systèmes de transport intelligents (ITS); finalement nous utilisons le logiciel TimeSquare pour simuler notre modèle et en vérifier la validité.

Mots-clés : Méthodes formelles, Systèmes distribués hétérogènes, Communications synchrones et asynchrones, Modl̀es temporisés, Systèmes de transport intelligents

1 Introduction

Heterogeneous distributed systems, as targeted in this paper, can be characterized by the fact that the processors are spatially separated and that a common time base does not exist. Distinct processes in such systems communicate with each other by exchanging messages with unpredictable (but non-zero) transmission delay. Intelligent Transportation System (ITS) is one typical application in this area. It consists of distributed vehicles which are equipped with their own independent clock. Despite each vehicle has a common time base (e.g. local physical clock), there is no global physical clock shared by vehicles. In such a context, it is impossible to build time constrains (e.g. action α from process A and action β from process B happens at the same time) since the processors have no consistent view of the time.

In this paper, we propose a timed model that is able to specify time constrains based on logical time. Logical time has proved its benefits in several domains. It was first introduced by Lamport to represent the execution of distributed systems [Lam78]. It has then been extended and used in distributed systems to check the communication and causality path correctness [Fid91]. Logical time has also been intensively used in synchronous languages [Ber00] [BLGJ91] for its multiform nature. The multiform nature of logical time consists in the ability to use any repetitive event as a reference for the other ones. It is then possible to express temporal properties between various references. In the synchronous domain it has proved to be adaptable to any level of description, from very flexible causal time descriptions to very precise scheduling descriptions [BDS91]. Logical time can be multiform, a global partial order built from local total orders of clocks. Inspired by the CCSL model [And09], we design clock relations to express the systems logical time constraints. So our model is a logical constraint model that is expressed by a set of logical clocks and clock constraints. In an heterogeneous distributed system design cycle, from an initial set of abstract time relations, and through architecture and platform-dependent design decisions, time refinement steps take place which solve in part the constraints between clocks, committing to schedule and placement decisions. The final version should be totally ordered, and then subject to physical timing verification and to physical constraints.

Our model is based on pNets (parameterized networks of synchronized automata)[BBC$^+$09], an expressive and flexible semantic model for the modeling and verification of (untimed) distributed systems. The pNet model describes the behavior of concurrent systems in terms of value-passing labelled transition systems (LTSs), and expresses communication and synchronisation with synchronisation vectors (originating in [Arn94]). It allows to model a large variety of synchronisation mechanisms and has been traditionally used for systems of either synchronously or asynchronously communicating objects, and of distributed components [BBC$^+$09]. The flexibility of the synchronisation vectors mechanism naturally provides descriptions of heterogeneous systems, from point-to-point or multipoint synchronisation, to sophisticated asynchronous queuing policies. Parametrization and hierarchy also makes pNet models compact, and close to the program structure, and as a consequence easy to generate in a compositional way [ABHMS12]. All these advantages attracted us to choose it for modelling the system. However, pNets have no mechanism to describe time constraints neither to explicitly define asynchronous communication behaviors. We propose a novel timed model called timed-pNets by introducing timed specifications into pNets. The timed specifications represent system's logical clocks and their relations so that the system's time-related behavior can be specified.

In recent work about ITS, or more generally on cyber-physical systems (CPS), people use models with continuous time, that is required for expressing the system's physical behavior evolution and control. In this paper, we concentrate on abstract models, that are appropriate to reason about the communication, synchronisation, and overall timing constraints of heterogeneous systems. We assume that in our model physical signals can be well sampled and transformed to digital signals. Therefore, even if we don't build continuous time model for physical world, we do not isolate our model from physical world. Our model

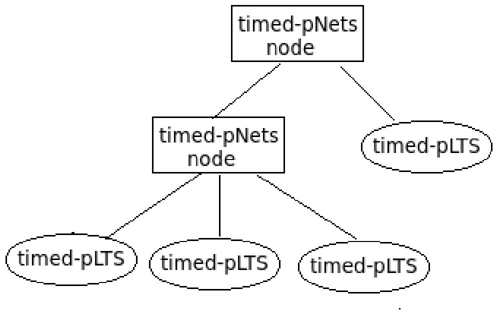


Figure 1: Timed-pNets tree structure

is capable to monitor and control physical behavior by taking sampling values from physical world.

In our previous work [CCM12] we proposed the first version of timed-pNets, including a notion of logical clocks directly imported from CCSL. A set of clock constraints related to the logical clocks were built to describe the system's casual relations. We also presented a simple use-case inspired from ITS systems, and showed how we simulate the set of timed-action traces by using the TimeSquare tool [DM12]. However this model was not sufficient to build a hierarchical timed specification starting from timed-pLTSs.

In this new paper, we enhance the compositional aspects of our specification methodology: a system is modelled as a hierarchy of timed-pNets as Fig.1, where leaves are timed-pLTS, i.e. finite state machines with logical clocks on the transitions, and nodes are synchronisation devices. Products between subnets can be synchronous (modelling local components sharing synchronous clocks), or involve asynchronous communication between unrelated events, that we model as channels.

From such a hierarchical model, we propose procedures for:

- at the bottom level, analyzing timed-pLTSs, and build the timed specifications (sets of clocks and clock constraints) encoding its temporal behaviour,
- for each timed-pNets node, building an abstract timed specification (= at level N), from its lower-level timed specifications (level N-1).

One important point is that Timed Specifications (TSs) are logical characterizations, that can be either provided by the application designer, or computed from the model. The consequence is that the two procedures above can be used arbitrarily in a bottom-up fashion, starting with detailed timed-pLTS and assembling them in a compatible way; or in a top-down fashion, constructing TSs for abstract timed-pNets, using their holes TSs as hypotheses in an assume-guarantee style, and providing later some specific (compatible) implementations for these holes in various contexts.

At each level, we are able to use the TimeSquare tool[DM12] to simulate the possible executions of a timed specification.

This rest of the paper is organized as follows. Section 2 describes the meaning of timed specification including the formal definitions of timed-actions, logical clocks and their relations. Then we give a definition of timed-pLTS in section 3. In section 4 we discuss how to build timed-pNets. The issue of checking the compatibility of timed-pNets is discussed in section 5. The procedure generating timed specification from timed-pLTS and timed-pNets are presented respectively in sections 6, 7. In section 8 we discuss how to build multi-layers timed-pNets systems. Then in section 9 we represent the simulations by using TimeSquare tool. Finally, the paper ends with conclusions and future research as well as some related works.

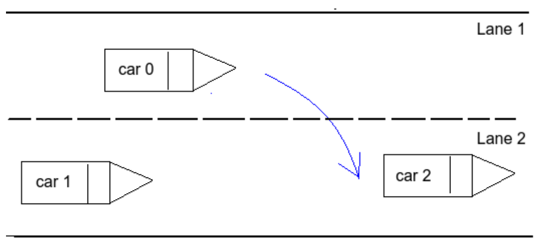


Figure 2: Car Insertion

2 Timed Specification

In this section, we present the preliminary denotations and definitions of timed-actions, logical clocks, clock relations and timed specification.

We shall use one example (Fig.2) to illustrate all definitions and results. We choose a small scenario taken from the field of ITS. It is about an autonomous lane change involving 3 smart cars. These cars are equipped with sensors to detect the physical environment and parameters (e.g. such as cars speed, cars distance, etc.). And they communicate among each other to coordinate their movements and avoid collisions. Assume the three vehicles ($car0$, $car1$ and $car2$) are running on a road as Fig. 2. The scenario of inserting $car0$ between $car1$ and $car2$ may follow the following steps: 0) $car0$ gets a change-lane request (e.g. from a human user); 1) $car0$ sends "notify" requests to $car1$ and $car2$ to get an agreement; 2) $car1$ (resp. $car2$) acknowledges $car0$ "yes" or "no"; 3) $car0$ collects results from $car1$ and $car2$; 4) If both $car1$ and $car2$ answer "yes", $car0$ signals the consensus to $car1$ and $car2$ and then go to step 5, otherwise $car0$ aborts the procedure; 5) $car1$ slows down and/or $car2$ speeds up to leave more space between them for $car0$; 6) $car0$ changes its direction and moves to lane2; 7) $car0$ notifies the end of the procedure with a "finish" signal.

As we do not want to limit ourselves to a specific language and a specific communication model, we follow the pNets assumption on Action Algebra $\mathcal{L}_{\mathcal{A},\mathcal{P}}$ which includes all required operators for building action expressions in the language ($\mathcal{P}$ a set of parameters used to build open expressions, typically expressing data variables)[BBC$^+$09]. We denote an action for sending a message as $!\alpha(m)$ ($m \in \mathcal{P}$) and receiving a message as $?\alpha(m)$ ($m \in \mathcal{P}$), which is similar to CCS or Lotos. For example, a (value-passing) CCS action could be "a?x:int", and an open action expression in the context of Lotos could be "G?x:int?y:int!x+y". Then for building timed-actions we introduce $\mathcal{T}$ as a set of (discrete) timed variables. And we build timed expressions using classical constants and operators over natural numbers.

Definition 1 (Timed-Actions). *Let $\mathcal{T}$ be a set of discrete time variables with domains in the non-negative natural numbers $\mathbb{N}$. The Timed-action Algebra $\mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ is an action set built over $\mathcal{T}$ and $\mathcal{P}$. We call $\alpha(p)^t \in \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ a timed-action in which $\alpha \in \mathcal{A}$ is an action, $p \in \mathcal{P}$ is a parameter, $t \in \mathcal{T}$ is a time variable describing a time delay before the action can be executed,*

We set $\alpha^0 = \alpha$, which means the action α is always ready.

We define a *Clock* as a sequence of occurrences of a timed-action. The clock, in the sense of CCSL, is a logical clock, which is firstly proposed by Lamport[Lam78]. The logical clock means the distance between occurrences is not related with the passage of real time.

To preserve this independence with respect to any notion of global time, in the following definitions and proofs, we use only the notions of co-occurrence ($\alpha\_i \equiv \beta\_j$), and of precedence ($\alpha\_i \prec \beta\_j$) of action occurrences. This will stay valid in any interpretation of the logical time scales.

Definition 2 (Clock). *A Clock C_α is a sequence of occurrences of a timed-action $\alpha(p)^t$. We write:*

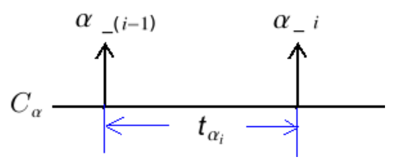


Figure 3: count the delay t_{α_i} when C_α is an independent clock

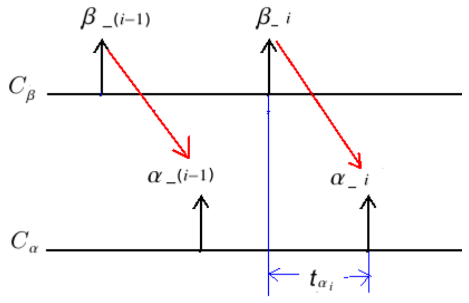


Figure 4: count the delay t_{α_i} when $C_\beta \prec C_\alpha$

$C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$ $(i \in \mathbb{N})$, *in which* $\alpha(p_i)^{t_{\alpha_i}}\_i$ *denotes the* i^{th} *occurrence of clock* C_α.

For simplification, in our paper, an occurrence $\alpha(p_i)^{t_{\alpha_i}}\_i$ can be denoted as α_i for short when not ambiguous.

The assignment of the delay variable t_{α_i} in each occurrence $\alpha(p_i)^{t_{\alpha_i}}\_i$ can be different. The delay variable captures the minimum time (delay) that an action must wait before it can occur after the previous action. More precisely when a clock is independent (has no precedence relation with another clock), the delay is counted from the previous occurrence of the same action as shown in the Fig. 3. If a clock C_β directly precedes a clock C_α, then the delay of the i^{th} occurrence of the timed-action α is counted from the i^{th} occurrence of the timed-action β as shown in the Fig. 4. The relation of coincidence (discussed in the next subsection) does not effect on the way of counting the delay. For example, if there is another clock C_γ that coincides with the clock C_α, then the delay t_{α_i} is still be counted as shown in the Fig.4.

For convenience, we define here two operators on Clocks, expressing respectively time shift, and filtering:

Definition 3 (Clock Offset). *Let* C_α *be a clock built over a timed-action* α, $C_\alpha[i]$ *be the* i^{th} *occurrence of the clock* C_α. *The* <u>n^{th} *offset*</u> *of the clock* C_α *is the clock defined as:* $C_\alpha^{\Delta(n)} = \{C_\alpha[n+1]\_1, C_\alpha[n+2]\_2, \ldots, C_\alpha[n+i]\_i, \ldots\}$.

From the definition we can see that the $(n+1)^{th}$ occurrence of C_α becomes the first occurrence of the new clock $C_\alpha^{\Delta(n)}$, and so on.

Definition 4 (Clock Filtering). *Assume* N' *is a subset of* $\mathbb{N}$. *Let* C_α *be a clock built over a timed-action* α. *The new clock that is filtered from the clock* C_α *by* N' *is denoted as*
$C_\alpha^{N'} = \{C_\alpha[i_1]\_1, C_\alpha[i_2]\_2, \ldots C_\alpha[i_k]\_j, \ldots\}(i_1, i_2, \ldots i_k, \ldots \in N', i_1 < i_2 < \ldots < i_k, \ldots, j, k \in \mathbb{N})$.

For convenience, we will write the filter N' either as a boolean function over $\mathbb{N}$, or as a subset of $\mathbb{N}$, e.g.: $C_\alpha^{\{2n-1\}_{n\in\mathbb{N}}}$ accepts only the odd occurrences of the clock C_α. $C_\alpha^{\{n\geq 8\}}$ filters out the first 8 occurrences.

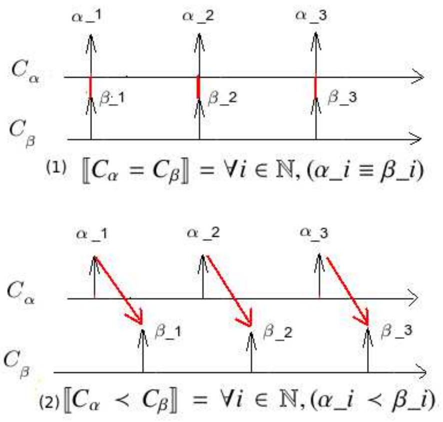
So if $C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$,
then $C_\alpha^{\{2n-1\}_{n\in\mathbb{N}}} = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_3)^{t_{\alpha_3}}\_2, \ldots \alpha(p_{(2n-1)})^{t_{\alpha(2n-1)}}\_n, \ldots\}$
and $C_\alpha^{\{n\geq 8\}} = \{\alpha(p_8)^{t_{\alpha_8}}\_1, \alpha(p_9)^{t_{\alpha_9}}\_2, \ldots\}$

Finally we define Timed Specifications: a timed specification is composed of a set of logical clocks, together with a set of clock relations, expressing the temporal ordering constraints between the clocks. This is an abstract specification in the sense that it captures just enough information to check the time safety (validity of time requirements) of a system, but also the compatibility relation required for assembling sub-systems together. In the next sections we shall describe procedures to compute the Timed Specifications of systems (timed-pLTS and timed-pNets), and to check compatibility.

Definition 5 (Timed Specification)**.** *Let $\mathcal{I}_c$ be the set of occurrences of the clock c. A Timed Specification is a pair $< \mathcal{C}, \mathcal{R} >$ where $\mathcal{C}$ is a set of clocks, $\mathcal{R}$ is a set of clock relations on $\bigcup_{c\in\mathcal{C}} \mathcal{I}_c$.*

2.1 Syntax and Semantic of Clock Relations

A Clock Relation defines the relation between two clocks. With respect to the original definition of clock relations in CCSL [And09], we have slightly different goals, and different needs. In particular we do not need exclusion (that is most important with some families of reactive formalisms). We do not define "subclock" relation in this paper because we need a more concrete way to define how to build a new subclock from original one. Instead, we defined "clock filtering" which can specify the way of selecting action occurrences. Therefore, here we only define two relation operations ('$\prec$', '$=$') to describe the different dependence relations between clocks.



α_1
α_2
α_3
C_α
β_1
β_2
β_3
C_β
(1) $[\![C_\alpha = C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \equiv \beta\_i)$
α_1
α_2
α_3
C_α
β_1
β_2
β_3
C_β
(2) $[\![C_\alpha \prec C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \prec \beta\_i)$

Figure 5: Constraints

- The relation '$C_\alpha = C_\beta$' (C_α coincides with C_β) describes the strict synchronization of clocks. It means that the occurrence of C_α appears if and only if the occurrence of C_β appears. In another word, the clock C_α and C_β tick at the same time. Formally, $[\![C_\alpha = C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \equiv \beta\_i)$ (shown in Fig. 5(1)). This operator can naturally be used to describe synchronous communication.

- The relation '$C_\alpha \prec C_\beta$' (C_α precedes C_β) describes the precedence relation of clocks. It says that the action β from the clock C_β cannot occur until the corresponding action α in the clock C_α occurs. In another word, clock C_α ticks always earlier than clock C_β. Formally $[\![C_\alpha \prec C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \prec \beta\_i)$. As shown in Fig. 5(2), the i^{th} occurrence of the clock C_α always appears earlier than the i^{th} occurrence of the clock C_β. The relation usually relates to the causality induced by asynchronous communication.

2.2 Properties of the logical clock relations

Not surprisingly, these relations have their expected properties: coincidence is an equivalence relation, and precedence is a strict pre-order.

Proposition 1 (Properties of the Coincidence Relation '$=$'). *Given a set of clocks C . The relation '$=$' on the set C is reflexive, symmetric and transitive.*

Proof: This follows from the fact that $\equiv$ is an equivalence relation on timed-action occurrences.
(1) Choose any clock $C_\alpha \in C$. Let its i^{th} $(i \in \mathbb{N})$ occurrence be α_i. The occurrence α coincides with itself. So we know $C_\alpha = C_\alpha$; the coincidence relation is reflexive. (2) Now choose another clock $C_\beta \in C$. If we have the relation of $C_\alpha = C_\beta$, then we know that $\forall i \in \mathbb{N}$, $\alpha\_i \equiv \beta\_i$, which means the action α occurs if and only if the action β occurs. According to the symmetric relation of the operator "$\equiv$", we know that the action β occurs if and only if the action α occurs. So we have $\forall i \in \mathbb{N}$, $\beta\_i \equiv \alpha\_i$. We know $C_\beta = C_\alpha$; the coincidence relation is symmetric. (3) choose another clock $C_\gamma \in C$. If we have relation $C_\alpha = C_\beta$ and $C_\beta = C_\gamma$, then $\forall i \in \mathbb{N}$, $\alpha\_i \equiv \beta\_i \wedge \beta\_i \equiv \gamma\_i$. From the transitivity relation of "$\equiv$", we infer $\forall i \in \mathbb{N}, \alpha\_i \equiv \gamma\_i$; so we know $C_\alpha = C_\gamma$; the coincidence relation is transitive. □

Proposition 2 (The properties of Precedence Relation $' \prec '$). *Given a clock set C. The relation $' \prec '$ on the set C is transitive, but not reflexive, not symmetric.*

This follows from the same properties on the relation $\prec$ on occurrences. The proofs are similar to those of Proposition 1.

Proposition 3 (Substitutivity of "$=$"). *Given four clocks C_α, C_β, C_γ, C_η which are built on the timed-action α, β, γ and η separately. Let $C_\alpha = C_\beta$ and $C_\gamma = C_\eta$. If $C_\alpha \prec C_\gamma$, then we have $C_\beta \prec C_\eta$.*

Proof. According to the coincidence definition, $C_\alpha = C_\beta \Rightarrow \forall i, \alpha\_i \equiv \beta\_i$, and $C_\gamma = C_\eta \Rightarrow \forall i, \gamma\_i \equiv \eta\_i$. If $C_\alpha \prec C_\gamma$, then according to the precedence definition, we know $\forall i, \alpha\_i \prec \gamma\_i$, which means the action α always occurs earlier than the action γ. Since $\forall i, \alpha\_i \equiv \beta\_i$ tells us the action α occurs if and only if the action β occurs, so we know β always occurs earlier than γ $(\forall i, \beta\_i \prec \gamma\_i)$. Similar, since $\forall i, \gamma\_i \equiv \eta\_i$ tells us the action γ occurs if and only if the action η occurs, so we furthermore have the relation $\forall i, \beta\_i \prec \eta\_i$. According to precedence relation definition, we get $C_\beta \prec C_\eta$. □

Example 1. *In this part, we illustrate how to represent timed-actions, clocks, and clock relations for our "car inserting" scenario.*

As shown in the Fig. 6, on-board car systems are modeled by several components including "Initial", "CommIni" "CommRes", "Control", etc. In the figure we only show the components that participate in the protocol.

User's requests are received by the "Initial" component. For our example, the "user" has sent an *insertion* order, encoded here as a "$!Request(Ins)^{t_q}$" timed-action occurrence. The procedure then runs in two phases:
(1) The agreement phase: $car0$ sends a $notify(Ins)$ message to the other two, and wait for their answers. This phase is managed by the "CommIni" process, that communicates to the other cars "ComRes" processes through asynchronous channels. In the model, there is one such channel for each type of message, and for each pair of communicating processes; we use the parameterized structure of pNets to represent such families of processes in the figure, e.g. "channelNtf[m]". The "CommIni" process is in charge of collecting the answers from the other cars asynchronously, and sending the final decision to "Initial". If it is negative, then "Initial" aborts and signals *Cancel* to the user, otherwise we go to the next phase.
(2) The execution phase: this phase is triggered and controlled directly by the "Initial" process. It sends

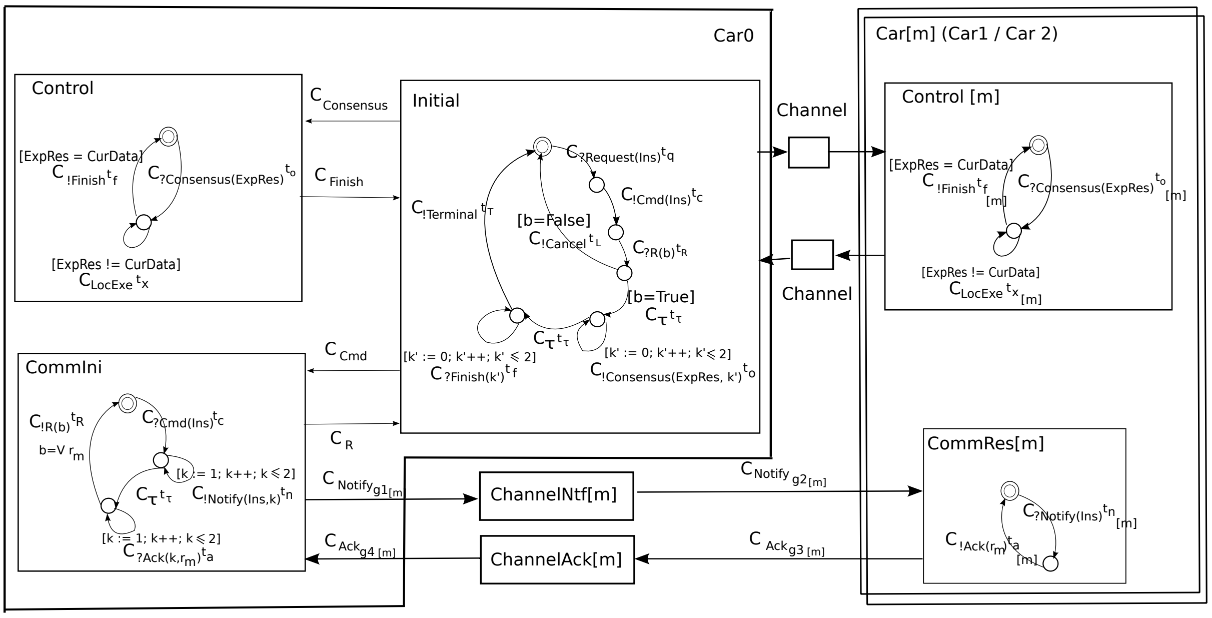


Figure 6: Timed-pNets: Communication Behaviour Model of Cars Insertion Scenario

$C_{!Consensus(ExpRes)^{t_o}}$ to all cars including itself to initiate the execution and to tell each car the final expected result ("ExpRes"). The "Control" process of each car is in charge of the local Execution of the movement (that we leave unspecified here), till the expected result is observed ($[ExpRes = CurData]$). Then the $!Finish$ signals are collected by "Initial", and termination is notified to the user.

We use label transition systems (LTSs) to model each component. Each transition will be triggered by a clock. Precedence relations are used to specify the causality relations of LTSs. For example, in the "CommRes" component, the clock "$C_{?notify(Ins)^{t_n}}$" occurs earlier than the clock "$C_{!ack(r_m)^{t_a}}$". We denote the clock relation as "$C_{?notify(Ins)^{t_n}} \prec C_{!ack(r_m)^{t_a}}$". For simplification, in the following sections, we will omit the parameters and time variables when expressing a clock relation if it is not ambiguous. For example, we use the short version "$C_{?notify} \prec C_{!ack}$" instead of "$C_{?notify(Ins)^{t_n}} \prec C_{!ack(r_m)^{t_a}}$".

In this use-case, we assume for simplicity that the communication inside a car is synchronous (in realistic modern car systems, this hypothesis would have to be refined, since the onboard systems include several process communicating through data buses). Here, the timed-action "$!Cmd(par)^{t_c}$" in the "Initial" process and the timed-action "$?Cmd(par)^{t_c}$" in "CommIni" are always synchronous when the two components communicate and transmit the message "par". So these two clocks coincide: $(C_{Initial.!Cmd(par)^{t_c}} = C_{CommIni.?Cmd(par)^{t_c}})$.

By contrast, communication between two different cars is asynchronous (typically over some wireless ad-hoc network), and we want to be able to take into account the communication time in our analysis. For this we insert a specific asynchronous channel (built as a special timed-pLTS) for each type of message exchanged between cars.

These two mechanisms illustrate our approach to model heterogeneous synchronous/asynchronous systems. In the next section, we show how we formalise this by using our timed-pNets formalism.

3 The Timed-LTS Semantic Model

This section introduces timed transition systems (timed-pLTS), including their special case Channels. We illustrate each definition with a piece of our running example.

Definition 6 (Timed-pLTS). *A Timed-pLTS is a tuple* $< P, S, s_0, A, C, \rightarrow >$, *where*

- *P is a finite set of parameters*
- *S is a set of states*
- *$s_0 \in S$ is the initial state*
- *A is a set of timed-actions*
- *C is a set of clocks over the timed-action set A*
- *$\rightarrow$ is the set of transition: $\rightarrow \subseteq S \times C \times S$. We write $s \xrightarrow{C_\alpha} s'$ for $(s, C_\alpha, s') \in \rightarrow$, in which $\alpha \in A$.*

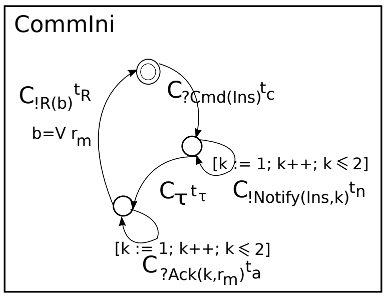

CommIni
$C_{!R(b)}{}^{t_R}$
$b = \vee r_m$
$C_{?Cmd(Ins)}{}^{t_c}$
$[k := 1; k{+}{+}; k \leqslant 2]$
$C_\tau{}^{t_\tau}$
$C_{!Notify(Ins,k)}{}^{t_n}$
$[k := 1; k{+}{+}; k \leqslant 2]$
$C_{?Ack(k,r_m)}{}^{t_a}$

Figure 7: The timed-pLTS of the CommIni component

Example 2. *Consider the "CommIni" component in Fig. 7. The clock relations will correspond to the precedence (causality) relations between the transitions of the LTS, with a special case for the loops on states s_1 (a state for sending notifications) and s_2 (a state for receiving "ack" signals), where the communication events are indexed by $k \in [1..N]$ where N is the (fixed) number of neighbors of the initiating car (here $N = 2$). The first loop on s_1 means car0 sends two notifications to car1 and car2 separately. The second loop on s_2 means car0 receives two "ack" signals from car1 and car2 separately. Moreover, we use a silent action τ to build a clock $C_\tau^{t_\tau}$ that labels the transition to state s_2 when the component finishes sending two notifications. We build the timed-pLTS elements as:*

- *Parameters $P = \{k, Ins, r_m, b, N\}$,*
- *Action algebra $A = \{?Cmd(par)^{t_c},\ !notify(par)^{t_n}, ?ack(k, r_m)^{t_a},\ !R(b)^{t_R}, \tau^{t_\tau}\}$*
- *Clocks $C = \{C_{?Cmd},\ C_{!notify}, C_{?ack},\ C_{!R}, C_\tau\}$*
- *(we do not detail the transition relation here, it is easily deduced from the figure)*

Note that what the system developer has to specify is only the LTS part, the clock constraints will be automatically deduced from the LTS (see section 5).

Channels. We introduce channels to model asynchronous communication behavior. A channel is defined as a special transition system with two timed-events: one for receiving messages, another for sending messages. The two events have a precedence constraint which models the delay of message transmission. For simplification, the channel definition here just describes a simple one place asynchronous buffer, sufficient to illustrate the heterogeneity of synchronous and asynchronous communications. More realistic asynchronous mechanisms are possible (e.g. n-places buffers, lousy channels, or ProActive/GCM request queues with futures [CHS08] but they are not the topic of this paper.

Definition 7 (Channel). *A channel is a transition system with tuple $< P, S, A, C, \prec, \rightarrow >$ in which*

- *P is a finite set of parameters,*
- *S is state set in which $S = \{s_{empty}, s_{data}\}$,*
- *$A = \{in(par)^{t_i}, out(par)^{t_o}\}$ $(par \in P)$ is the timed-action set,*
- *C is a set of clocks over timed-actions A,*
- *$\rightarrow$ is a set of two transitions: $s_{empty} \xrightarrow{C_{?in}} s_{data}$ and $s_{data} \xrightarrow{C_{!out}} s_{empty}$.*

In the channel definition, the timed-action $?in(par)^{t_i}$ is an action for receiving messages from one component, while the timed-action $!out(par)^{t_o}$ is an action for sending the messages to another component as shown in Fig. 8.

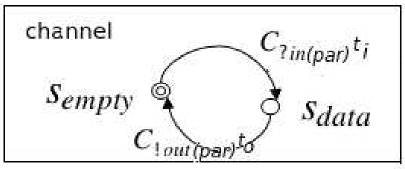


Figure 8: The timed-pLTS of channel Component

4 Timed-pNets Semantic Model

Finally we define Timed-pNets, that is our main structure used to combine sub-systems to build bigger systems. As in the original (untimed) pNets, a Timed-pNet is a generalized composition operator, defining the synchronization between a number of subsystems (holes). In timed-pNets, holes are characterized by an action algebra (a sort); here this is complemented by a Timed-Specification. Building the timed-pNet tree representing a full system will require filling holes with (compatible) sub-nets.

Definition 8 (Timed-pNets). *A Timed-pNet is a tuple $< P, A_G, C_G, J, \widetilde{A}_J, \widetilde{C}_J, \widetilde{R}_J, \overrightarrow{V} >$, where:*

- *P is a finite set of parameters,*
- *A_G is the set of global timed-actions, and C_G is the set of global clocks that are built over A_G,*
- *J is a countable set of argument indexes: each index $j \in J$ is called a hole and is associated with a set of local timed-actions A_j, and an associated Timed Specification $< C_j, R_j >$.*
- *$\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronization vectors of the form:*

- *(binary communication between holes j_1 and j_2)*
  $\overrightarrow{v}^1 =< \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots > \rightarrow C_g$,
  in which $\{C_{!\alpha} = C_{?\alpha} = C_g\}, C_g \in C_G, C_{!\alpha} \in C_{j_1}, C_{?\alpha} \in C_{j_2}, j_1, j_2 \in J$,

- *or (visibility from hole j)*
  $\overrightarrow{v} =< \ldots, C_\alpha, \ldots > \rightarrow C_g$, in which $\{C_\alpha = C_g\}, C_g \in C_G, C_{?\alpha} \in C_j, j \in J$.

  Furthermore, each global clock can be generated by only one synchronization vector:
  $\forall \quad \overrightarrow{v_i}, \overrightarrow{v_{i'}} \in \overrightarrow{V},\ C_{gi} = C_{gi'} \implies \overrightarrow{v_i} = \overrightarrow{v_{i'}}$
  (C_{gi}(resp. $C_{gi'}$) be a global clock generated by the vector $\overrightarrow{v_i}$ (resp. $\overrightarrow{v_{i'}}$), $i, i' \in \mathbb{N}$)

[1]where "..." represents an arbitrary number of holes that do not participate in this synchronization

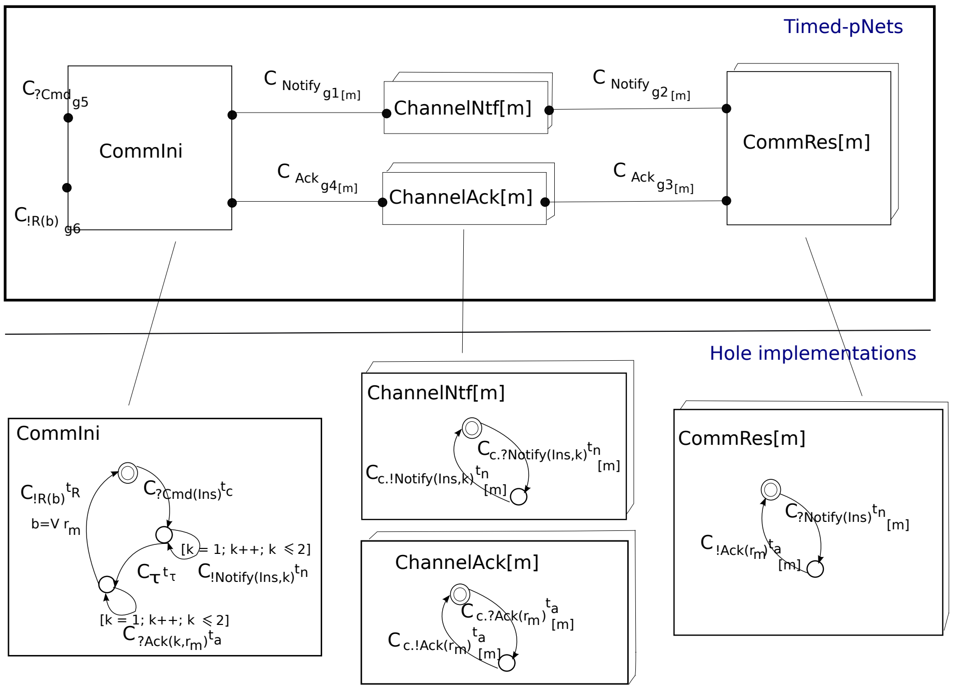


Figure 9: A Timed-pNets with one of its implementations

Remark: We define Nets in a form inspired by the synchronisation vectors of Arnold and Nivat [AP94], that we use to synchronise clocks from different processors. One of the main advantages of using its high abstraction level is that almost all interaction mechanisms encountered so far in the process algebra literature become particular cases of a very general concept: synchronisation vectors. We structure the synchronisation vectors as parts of network. Contrary to synchronisation constraints, the network allows dynamic reconfigurations between different sets of synchronisation vectors. In our timed-pNets, we define two kinds of synchronous vectors. One is communication vector $(< \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots > \to C_g)$. The vector represents the communication of two holes through clock $C_{!\alpha}$ and $C_{?\alpha}$. The two local clocks that come from different holes are put between the two symbols "$<$" and "$>$". The last element of the vector appears behind the symbol "$\to$", and specifies the global clock generated by this synchronous vector. Another vector $(< \ldots, C_{\alpha}, \ldots > \to C_g)$ makes the local clock C_α visible by generating a global clock C_g.

Notations for parameterized systems. In practice, we use parametric notations, both for holes and for synchronization vectors, making the notations more compact and more user-friendly (see next example). These are only abbreviations, their meaning must be understood as a (finite) expansion of the structure.

Using such abbreviations, for a pNet in which j_1, j_2, j are parametric holes holes with indexes k_1, k_2, k, with respective domains Dom_1, Dom_2, Dom, the synchronization vectors will look like:

- binary communication
  Depending on the combination of actions from j_1 and j_2, this vector will generate a family of global actions indexed by a parameter m, that is a function of k_1 and k_2. The domain of m is a subset of the product $Dom_1 x Dom_2$. $< ..., C_{!\alpha[k_1]}, ..., C_{?\alpha[k_2]}, ... > \to C_{g[m]}$,
  in which $\{C_{!\alpha[k_1]} = C_{?\alpha[k_2]} = C_{g[m]}\}$, $C_{g[m]} \in C_G, C_{!\alpha[k_1]} \in C_{j_1}, C_{?\alpha[k_2]} \in C_{j_2}$

- visibility
  Each visible action from hole j generates a corresponding global action. $< ..., C_{\alpha[k]}, ... > \to C_{g[k]}$, in which $\{C_{\alpha[k]} = C_{g[k]}\}, C_{!\alpha[k]} \in C_j, C_{g[k]} \in C_G$.

Example 3. *We use our use case to illustrate how to build a timed-pNets model. To make the example smaller, we have extracted here the respective "communication" subNets of 2 cars, and the channels on which they communicate, and we show how to build a pNet encoding this small subsystem.*

As shown in the Fig.9, the subsystem consists of components "CommIni", "CommRes[m]", "ChannelNtf[m]" and "ChannelAck [m]". The components "ChannelNtf[m]" and "ChannelAck[m]" are channels in which the parameter "[m]" denotes to which car the corresponding channel transmits data. By using the parameter "m", we give a more compact representation of the model. According to our scenario, $car0$ sends a notification to $car1$ (resp. $car2$) via "ChannelNtf[1]" (resp."ChannelNtf[2]"), and then $car1$ (resp.$car2$) answers an "ack" to $car0$ via "ChannelAck[1]" (resp. ChannelAck[2]"). So in the upper layer timed-pNets node, we can link these components by building synchronous vectors. For example:
- the vector[2] $< -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}} > \to C_{ack_{g3_{[1]}}}$ represents the communication between the components "CommRes[1]" and "ChannelAck[1]" and generates the global clock "$C_{ack_{g3_{[1]}}}$". Notice that even though we actually have 7 subnets (CommIni, CommRes[1], CommRes[2], ChannelNtf[1], ChannelNtf[2], ChannelAck[1], ChannelAck[2]), using parameters we represent our pNet and its synchronous vectors with only 4 holes.
- the vector $< C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}, -, C_{c.?notify_{[1]}}, - > \to C_{notify_{g1_{[1]}}}$ represents the communication between the components "*CommIni*" and "$ChannelNtf[1]$" and builds a global clock "$C_{notify_{g1_{[1]}}}$" (remember $C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}$ is the clock built from the clock $C_{!notify}$ by choosing the occurrences with odd indexes). Following the timed-pNets definition, we can formalize this timed-pNets with details as:

- $P = \{k, Ins, m, r_m, b\}$,
- $A_G = \{notify(Ins,k)_{g1_{[m]}}^{t_{g1}}, notify(Ins,k)_{g2_{[m]}}^{t_{g2}}, ack(r_m,k)_{g3_{[m]}}^{t_{g3}},$
  $ack(r_m,k)_{g4_{[m]}}^{t_{g4}},\ ?Cmd(Ins)_{g5}^{t_{g5}}, !R(b)_{g6}^{(t_{g6}}\}$
- $C_G = \{C_{notify_{g1_{[m]}}}, C_{notify_{g2_{[m]}}}, C_{ack_{g3_{[m]}}}, C_{ack_{g4_{[m]}}}, C_{?Cmd_{g5}}, C_{!R_{g6}}\}$
- $J = \{CommIni, CommRes[m], ChannelNtf[m],$
  $ChannelAck[m]\}(m := 1, 2)$

Next we formalize the Timed Specifications of these holes as:

- For the hole "CommIni":

  $A_{CommIni} = \{?Cmd(Ins)^{t_c}, !notify(Ins,k)^{t_n}, ?ack(k,r_m)^{t_a}, !R(b)^{t_R}\}$

  $C_{CommIni} = \{C_{?Cmd}, C_{!notify}, C_{?ack}, C_{!R}\}$

  $$\begin{aligned} R_{CommIni} = \{ & C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}},\quad C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}}, \\ & C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}},\quad C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}, \\ & C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}},\quad C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\} \end{aligned}$$

- For the hole "CommRes[m]" (m := 1, 2):

  $A_{CommRes[m]} = \{?notify(Ins,k)_{[m]}^{t_n}, !ack(k,r_m)_{[m]}^{t_a}\}$

  $C_{CommRes[m]} = \{C_{?notify_{[m]}}, C_{!ack_{[m]}}\}$

  $R_{CommRes[m]} = \{C_{?notify_{[m]}} \prec C_{!ack_{[m]}} \prec C_{?notify_{[m]}}^{\Delta(1)}\}$

- For the hole "ChannelNtf[m]" (m := 1, 2):

  $A_{ChannelNtf[m]} = \{c.?notify(Ins,k)_{[m]}^{t_{n1}}, c.!notify(Ins,k)_{[m]}^{t_{n2}}\}$

[2]where "$-$" represents a single hole that does not participate in this synchronization

$C_{ChannelNtf[m]} = \{C_{c.?notify_{[m]}}, C_{c.!notify_{[m]}}\}$

$R_{channelNtf[m]} = \{C_{c.?notify_{[m]}} \prec C_{c.!notify_{[m]}} \prec C_{c.?notify_{[m]}^{\Delta(1)}}\}$

- For the hole “ChannelAck[m]” (m := 1, 2):

  $A_{ChannelAck[m]} = \{c.?ack(k, r_m)^{t_{a1}}_{[m]}, c.!ack(k, r_m)^{t_{a1}}_{[m]}\}$

  $C_{ChannelAck[m]} = \{C_{c.?ack_{[m]}}, C_{c.!ack_{[m]}}\}$

  $R_{channelAck[m]} = \{C_{c.?ack_{[m]}} \prec C_{c.!ack_{[m]}} \prec C^{\Delta(1)}_{c.?ack_{[m]}}\}$

In the end, we specify the synchronous vectors:

$\overrightarrow{V} = \{$
$V_1 :< C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify(Ins,k=1)}, -, C_{c.?notify(Ins)_{[1]}}, - >\rightarrow C_{notify_{g1_{[1]}}},$
$V_2 :< -, C_{?notify_{[1]}}, C_{c.!notify_{[1]}}, - >\rightarrow C_{notify_{g2_{[1]}}},$
$V_3 :< -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}} >\rightarrow C_{ack_{g3_{[1]}}},$
$V_4 :< C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack(k=1,r_m)}, -, -, C_{c.!ack(r_m)_{[1]}} >\rightarrow C_{ack_{g4_{[1]}}}$
$V_5 :< C^{\{2s\}_{s\in\mathbb{N}}}_{!notify(Ins,k=2)}, -, C_{c.?notify(Ins)_{[2]}}, - >\rightarrow C_{notify_{g1_{[2]}}},$
$V_6 :< -, C_{?notify_{[2]}}, C_{c.!notify_{[2]}}, - >\rightarrow C_{notify_{g2_{[2]}}},$
$V_7 :< -, C_{!ack_{[2]}}, -, C_{c.?ack_{[2]}} >\rightarrow C_{ack_{g3_{[2]}}},$
$V_8 :< C^{\{2s\}_{s\in\mathbb{N}}}_{?ack(k=2,r_m)}, -, -, C_{c.!ack(r_m)_{[2]}} >\rightarrow C_{ack_{g4_{[2]}}}$
$V_9 :< C_{?Cmd}, -, -, - >\rightarrow C_{?Cmd_{g5}},$
$V_{10} :< C_{!R}, -, -, - >\rightarrow C_{!R_{g6}}\}$

Discussion: Timed specification of holes. Let us now argue how the timed specifications of this upper-level timed-pNet holes may have been specified, in a top-down approach, before building their timed-pLTS implementation. This, intuitively, is done from the informal description of the scenario and the knowledge of the top level component and communication architecture:

Take the “CommIni” component as an example, the scenario related to the component is:

(1) the component “CommIni” gets a change-lane request by clock $C_{?cmd}$ from the “Initial” component;

(2) the component “CommIni” sends requests by clock $C_{!notify}$, in sequence, to $car1$ and $car2$ to get agreements;

(3) the component “CommIni” collects results from $car1$ and $car2$ by clock $C_{?ack}$;

(4) the component reports result to “Initial” component by clock $C_{!R}$.

Since step (1) happens earlier than the step (2), the clock $C_{?cmd}$ must precede the clock $C_{!notify}$. Then, in our use case, the component “CommIni" sends notification signal twice, so we have clock relation $\{C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify}\}$. In generally, if there are N neighbors, the clock relation should be $\{C_{?Cmd} \prec C^{\{Ns-(n-1)\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{Ns-(n-2)\}_{s\in\mathbb{N}}}_{!notify} \prec \ldots \prec C^{\{Ns\}_{s\in\mathbb{N}}}_{!notify}\}$. Similar to the step (2), since the component receives “ack” signal twice, so we have the clock relation $\{C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}\}$. Furthermore, the clock $C_{!notify}$ in step (2) should precede the clock $C_{?ack}$ in step (3), so we have the relation $C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack}$ and $C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}$. Finally the scenario goes to step (4), we have the relation $\{C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_{!R}\}$. Since the scenario is repeatable, we specify the clock relation $\{C_{!R} \prec C^{\Delta(1)}_{?cmd}\}$. In the end, we conclude:

$$\begin{aligned} R_{\{CommIni\}} = \{&C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify}, \quad C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify}, \\ &C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack}, \quad C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}, \\ &C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}, \quad C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_{!R} \prec C^{\Delta(1)}_{?cmd}\} \end{aligned}$$

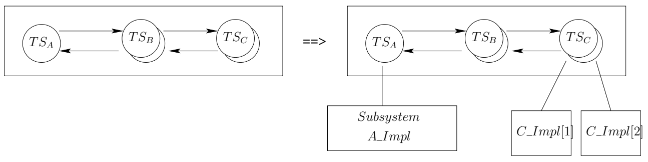


Figure 10: Partial instantiation of a Timed-pNets subsystem

In section 5, we will show that these Timed Specifications are indeed fulfilled by the corresponding timed-pLTS "CommIni", "ComRes", "ChannelNtf", and "ChannelAck".

5 Compatibility

When assembling timed-pNets, the architect has to ensure that the timed-pLTS that will be plugged into a hole indeed matches the hole Timed Specification. The ultimate goal is to provide a refinement-based approach: timed properties proved on an open (abstract) timed-pNet system will be preserved by refinement of Timed Specifications. One of the basic tool for building such refinement is to ensure the compatibility of a subsystem with the enclosing holes before composing the system. E.g. in Fig. 10, the Timed Specification (TS) of the subsystem "A_Impl" must be compatible with TS_A, and each of the "C_Impl" must be compatible (individually) with TS_C.

Our notion of compatibility will be based on the inclusion relations between the Clock relation sets. Before giving its formal definition, we introduce the concepts of "Saturated relation set" and "Relation set inclusion".

Definition 9 (Saturated Relation Set). *Let $TS =< C, R >$ be a timed specification with a set of clocks C and a set of relations R. The saturated relation set (denoted as R^+) is the clock relation set R augmented by all relations possibly deduced from R, by transitivity of precedence and reflexivity, symmetry, and transitivity of coincidence.*

For example, if $R = \{c_1 \prec c_2 \prec c_3\}$ $(c_1, c_2, c_3 \in C)$, then according to the transitivity property of the relation $\prec$, we can get a new relation set $R^+ = \{c_1 \prec c_2 \prec c_3, c_1 \prec c_3, c_1 = c_1, c_2 = c_2, ...\}$

Definition 10 (Inclusion of time specifications). *Given two timed specifications $TS_1 =< C_1, R_1 >$ and $TS_2 =< C_2, R_2 >$. Let R_1^+ (resp. R_2^+) be a set of hidden relations in the TS_1 (resp. TS_2). We say TS_2 includes TS_1 (denoted as $TS_1 \ll TS_2$) if and only if $C_1 \subseteq C_2 \wedge$ $R_1 \subseteq R_2^+$.*

According to the definition, $TS_1 \ll TS_2$ means that the relation existing in the timed specification TS_1 must exist in TS_2 or can be deduced from the relations in TS_2. For example, assume $TS_1 = \{c_1 \prec c_3\}$, $TS_2 = \{c_1 \prec c_2 \prec c_3\}$. According to the transitivity property of the "$\prec$", we can get the the saturated relation set of the TS_2 as $R^+ = \{c_1 \prec c_2 \prec c_3, c_1 \prec c_3, c_1 = c_1, c_2 = c_2, ...\}$. Since the relation in TS_1 can be deduced from the relations in TS_2, we say TS_2 includes TS_1 ($TS_1 \ll TS_2$).

Lemma 1. *If $TS_1 =< C_1, R_1 >$ and $TS_2 =< C_2, R_2 >$ are two timed specifications, then $TS_1 \ll TS_2 \implies R_1^+ \subseteq R_2^+$*

Proof. Taken any two relation r_1, $r_1' \in R_1$. Let $r_1^+ \in R_1^+$ be the relation deduced from the two relations r_1, r_1' in terms of the property P proposed in section 2. Assume $r_1^+ \notin R_2^+$. Since $TS_1 \ll TS_2$, from the definition of inclusion we know $R_1 \subseteq R_2^+$. Furthermore, we know $r_1, r_1' \in R_2^+$. So in the set R_2^+ we can get the relation r_1^+ by using the same property P. So we have $r_1^+ \in R_2^+$ that is contradict with our assumption. Therefore, we have $r_1^+ \in R_2^+$. Moreover, because $r_1^+ \in R_1^+$, so $R_1^+ \subseteq R_2^+$. □

Definition 11 (Compatibility)**.** *Let TS be the timed specification of a timed-pNets hole H, and TS' be the timed specification of an implementation H_Impl. We say H_Impl is* <u>*compatible*</u> *with H, denoted by $H\_Impl \sqsubseteq H$ if and only if $TS \ll TS'$.*

Theorem 1. *Let TS be the timed specification of hole H. Let TS'_1 (resp. TS'_2) be the timed specification of an implementation H_Impl_1 (resp. H_Impl_2). If $H\_Impl_1 \sqsubseteq H$ and $TS'_1 \ll TS'_2$, then $H\_Impl_2 \sqsubseteq H$.*

Proof. Assume $TS'_1 =< C'_1, R'_1 >$, $TS'_2 =< C'_2, R'_2 >$ and $TS =< C, R >$. Let R'^{+}_1 (resp. R'^{+}_2, R^{+}) be the saturated relation from TS'_1 (resp. TS'_2, TS). Since $H\_Impl_1 \sqsubseteq H$, according to the refinement relation, we have $TS \ll TS'_1$. Furthermore, according to the Inclusion definition, we have $R \subseteq R'^{+}_1$. Moreover, because we know that $TS'_1 \ll TS'_2$, according to the Lemma 1, we have $R'^{+}_1 \subseteq R'^{+}_2$. According to the set theory, we know that $R \subseteq R'^{+}_2$. Finally, according to the Inclusion and refinement relation definition, we get $H\_Impl_2 \sqsubseteq H$. □

6 Generating the timed specification of a timed-pLTS

As we see in the Fig.9, timed-pLTSs are concrete implementations of those holes. In order to check the compatibility, we need to generate timed specifications for those concrete timed-pLTSs. Here we propose rules to automatically generate a timed specification from the LTS part of a timed-pLTS. More precisely, given the action algebra and the transition relation of a timed-pLTS, we compute its set of clocks, and the relations between these clocks.

This procedure runs in 4 phases as shown in the Fig. 11. The inputs of the procedure include a timed-pLTS and a set of rules that tells how to set the occurrence relations and its index functions. In step 1, we traverse the timed-pLTS and generate a “symbolic” table that gathers all possible causally related pairs of transitions of the timed-pLTS, and the corresponding relations between clock occurrences. In step 2 we go through the symbolic table and build a “concrete” table in which each column represent one specific “round” of execution through the symbolic table (with concrete index assignments). In the concrete table guards of the timed-pLTS can be resolved, so some of the symbolic transitions may be eliminated. In step 3 we generate a general formula for each relation. In the end (step 4), we lift those occurrence relations to clock relations, and generate the Timed Specification

6.1 Auxiliary functions: Pre/Post sets

Before describing Step 1, we need to define functions computing the pre/post sets of the timed-pLTS states.

For a timed-pLTS transition system $< P, S, s_0, A, C, \rightarrow >$, we denote $PreAct(s, s')$, the set of direct preceding timed-action occurrences of s from s'; and $PostAct(s, s')$ the set of direct succeeding timed-action occurrences of state s towards state s'. Then we denote $PreAct(s)$ (resp. $PostAct(s)$) the set of all direct preceding (resp. succeeding) timed-action occurrences of state s. Furthermore, we define $PreActIndex(s)$ (resp. $PostActIndex(s)$) as the sum of the indexes of the set of preceding (resp. succeeding) timed-action occurrences of state s. The sum corresponds to cases where branching in the LTS allows some executions to go several times through alternative transitions out of some states. Formally:

Definition 12 (Preceding Timed-Action Occurrences)**.** *Let $< P, S, s_0, A, C, \rightarrow >$ be a timed-pLTS transition system. For $s \in S$ and $\alpha(p)^{t_\alpha|b} \in A, (p \in P)$, the direct preceding timed-action occurrence of s is defined as $PreAct(s, s') = \{\alpha\_i | s' \xrightarrow{C_\alpha} s, \alpha\_i \in C_\alpha, \}$ $(s, s' \in S)$. The set of direct preceding timed-action occurrences of s is defined as $PreAct(s) = \bigcup_{s' \in S} PreAct(s, s')$. Furthermore, we denote the index of a preceding timed-action occurrence as $PreActIndex(s, s') = \{i | s' \xrightarrow{C_\alpha} s, \alpha\_i \in C_\alpha(s, s' \in S)\}$, and*

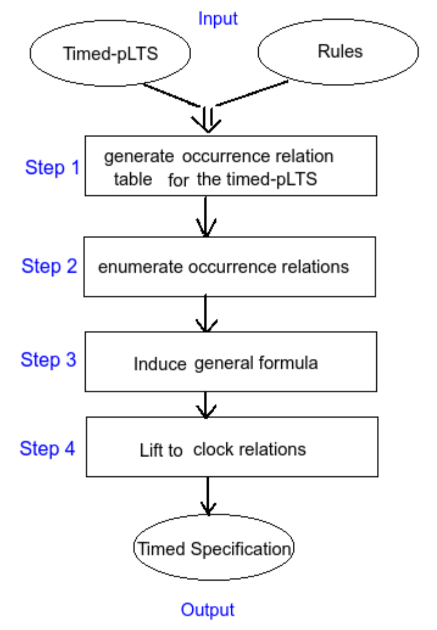


Figure 11: Steps for generating the TS of a timed-pLTS

State	Transition	Occurrence Relations	Index Assignment
s_0	$tr0 : s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$	$!R\_m \prec ?Cmd\_n$	$f_{tr0} : n = m + 1$
s_1	$tr1 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{\tau}} s_2$	$?Cmd\_n \prec \ \tau\_r$	$f_{tr1} : r = n$
	$tr2 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{!Notify}} s_1$	$?Cmd\_n \prec !notify\_i$	$f_{tr2} : i := i + 1$
	$tr3 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{!Notify}} s_1$	$!notify\_i \prec !notify\_(i + 1)$	
	$tr4 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{\tau}} s_2$	$!notify\_i \prec \tau\_r$	$f_{tr4} : r = n$
s_2	$tr5 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{!R}} s_0$	$\tau\_r \prec !R\_m$	$f_{tr5} : m = r$
	$tr6 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{?Ack}} s_2$	$\tau\_r \prec ?ack\_j$	$f_{tr6} : j := j + 1$
	$tr7 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{?Ack}} s_2$	$?Ack\_j \prec ?ack\_(j + 1)$	
	$tr8 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{!R}} s_0$	$?Ack\_j \prec !R\_m$	$f_{tr8} : m = r$

Figure 12: Time assignment for the Timed-pLTS "Car.CommIni"

the sum of the indexes of a set of preceding timed-action occurrences of state s as $PreActIndex(s) = \sum_{s' \in S} PreActIndex(s, s')$.

Definition 13 (Succeeding Timed-Action Occurrences)**.** *Let* $< P, S, s_0, A, C, \rightarrow >$ *be a timed-pLTS transition system. For* $s \in S$ *and* $\alpha(p)^{t_\alpha|b} \in A, (p \in P)$*, the direct succeeding timed-action occurrence of state* s *is defined as* $PostAct(s, s') = \{\alpha\_i | s \xrightarrow{C_\alpha} s', \alpha\_i \in C_\alpha\}$, $(s, s' \in S)$*. The set of direct succeeding timed-action occurrences of state* s *is defined as* $PostAct(s) = \bigcup_{s' \in S} PostAct(s, s')$*. Furthermore, we denote the index of a succeeding timed-action occurrence as* $PostActIndex(s, s') = \{i | s \xrightarrow{C_\alpha} s', \alpha\_i \in C_\alpha\}$, $(s, s' \in S)$*, and the sum of the indexes of a set of succeeding timed-action occurrences of* s *as* $PostActIndex(s) = \sum_{s' \in S} PostActIndex(s, s')$.

6.2 Relations and assignment rules

The computation in Step 1 is based on a set of rules identifying specific configurations of the states in the timed-pLTS traversal. For each such configuration, we define a rule that expresses the relation(s) between the set of preceding and succeeding clock occurrences of the current state, and the changes in the clock occurrence indexes.

The main configurations are: initial state, in which we have to initialize indexes, and increase an index each time the system goes through a new global round; standard state in which we register the increase of one of the involved index; and looping states, in which we have to take care of guards for entering/leaving loops, in terms of a specific "loop counter".

We define a restrictive notion of looping state which are reasonable configurations for timed analysis. A looping state may have one or more loops of arbitrary length, but coming back to the same state. And each loop must start with a transition with a guard taking the precise form of a "loop counter" control, namely `[k=1; k++; k ≤ kMax]` for some counter variable `k`, in which `kMax` may be a positive natural number, or a variable. Loop guards can share a loop counter (see e.g. Fig. 13), so several loops will be executed the same number of times; but otherwise different loop counters must be independent. Of course one could imagine more complex structures for our timed-pLTSs, but this restriction already covers a lot of interesting cases, and make the generation of the Times Specification easier.

In these rules, for simplification, we represent relations on two sets (S_1 (resp. S_2) is a set of occurrences of clocks): $S_1 \prec S_2$ means $\forall \alpha_m \in S_1, \beta_n \in S_2, \alpha_m \prec \beta_n$ $(m, n \in \mathbb{N})$.

(1) **Initial state**. If $PreAct(s_0) \notin \varnothing$, then $PreAct(s_0) \prec PostAct(s_0)$,
[**Assign:** $PostActIndex(s_0) \Leftarrow PreActIndex(s_0) + 1$];

(2) **Standard state**. $\forall s \backslash s_0$, $PreAct(s) \prec PostAct(s)$,
[**Assign:** $PostActIndex(s) \Leftarrow PreActIndex(s)$];

(3) **Looping state**. $\forall s$, if $\exists \alpha . s \xrightarrow{C_\alpha} s$ and the loop executes N times, then

- (3.1) go inside the loop
  $PreAct(s) \prec \alpha\_i$,
  [**Assign:** $i := i + 1$]
- (3.2) stay in loop,
  $\alpha\_i \prec \alpha\_(i+1)$
- (3.3) leave loop:
  - (3.3.1) leave loop to another loop, e.g. $\exists \beta . s \xrightarrow{C_\beta} s$ $(\beta\_j \in PostAct(s,s) \backslash \alpha\_i)$:
    $\alpha\_i \prec \beta\_j$,
    [**Assign:** $j := j + 1$]
  - (3.3.2) to one post-action out of $PostAct(s, s_0)$:
    $\alpha\_i \prec PostAct(s) \backslash PostAct(s, s_0)$,
    [**Assign:**
    $PostActIndex(s) \Leftarrow PreActIndex(s)$].
  - (3.3.3) to one post-action in $PostAct(s, s_0)$:
    $\alpha\_i \prec PostAct(s, s_0)$,
    [**Assign:**
    $PostActIndex(s) \Leftarrow PreActIndex(s) + 1$].

6.3 The Method for Generating Timed Specification

This subsection introduces a method of generating timed specification from timed-pLTS. We state two algorithms and 4 steps.

6.3.1 Step 1: generate occurrence relations table

Algorithm1 uses the rules above to build an occurrence relation table. More precisely each row in the table lists a specific pair of Pre/Post transitions of a state, with the corresponding occurrence relation and index increase function deduced from the corresponding rule.

Algorithm 1 Generate occurrence relations table

Input: a timed-pLTS graph and rules.
Output: A table of occurrence relation with its index assignment function.

```
for each state $s_i$ in LTS graph do
  for each pair $(s_1, s_2)$ such that $s_1 \xrightarrow{C_1} s_i \xrightarrow{C_2} s_2$ do
    insert a row with $State = s_i$, $Transition = s_1 \xrightarrow{C_1} s_i \xrightarrow{C_2} s_2$.
    if $s_i = s_0$ AND $s_i$ has no self-loop then
      apply case (1) rules, adding the relations and assignments in the corresponding rows.
    end if
    if $s_i \neq s_0$ AND $s_i$ has no self-loop then
      apply case (2) rules
    end if
    if $s_i$ includes one self-loop then
      if $s_i = s_0$ then
        apply case (1), (3.1), (3.2) and (3.3.3) rules
      else
        apply case (2), (3.1), (3.2) and (3.3.2) rules
      end if
    else
      if $s_i = s_0$ then
        apply case (1), (3.1), (3.2), (3.3.1) and (3.3.3) rules
      else
        apply case (2), (3.1), (3.2), (3.3.1) and (3.3.2) rules
      end if
    end if
  end for
end for
```

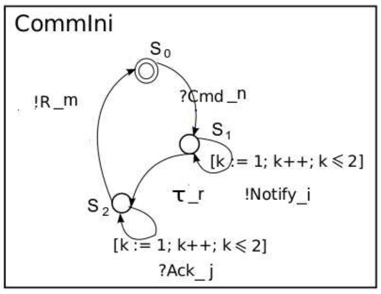


Figure 13: Simplification of CommIni component

Example 4. *Let us take the "CommIni" component from Fig. 7 as an example. We first transform Fig. 7 into Fig. 13 by removing all parameters but adding index variables. Then we generate occurrence relations for each state. For example, we take the state "s_0", from the timed-pLTS graph we get transitions $s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$. According to the rule (1) we have $!R\_m \prec ?Cmd\_n$ and the assignment $n = m+1$ ($n, m \in \mathbb{N}$). Take the state s_1 as another example. Since it includes a self-loop, we discuss apply the* *rules (2), (3.1), (3.2) and (3.3.2). When a transition directly brings to next state without passing the loop, according to the rule (2), we have the relation* $?Cmd\_n \prec \ \tau\_r$ *and assignment* $r = n$*. When a transition enters the loop, according to the rule (3.1), we have the relation* $?Cmd\_n \prec\ !notify\_i$ *and assignment* $i := i + 1$ *(*$i \in \mathbb{N}$*). When a transition stays in the loop, according to the rule (3.2), we can get the relation "*$!notify\_i \prec\ !notify\_i + 1$*" (*$i \in \mathbb{N}$*). Then when a transition leaves the loop, according to the rule (3.3.2), we have the relation* $!notify\_i \prec \tau\_r$ *and the assignment* $r = n$ *(*$r \in \mathbb{N}$*).*

6.3.2 Step 2: Enumerate occurrence relations

Now we go through the symbolic occurrence table built in step 1 and build a "concrete" table in which each column represents one specific "round" of execution through the symbolic table (with concrete index assignments). In the concrete table the guards of the timed-pLTS can be resolved, so some of the symbolic transitions (rows of the table) may be eliminated.

In the guards (including the loop control guards), there may be some parameters occurring in a symbolic form. Before we run the algorithm in step 2, we need to instantiate these parameters, to be able to compute the guards. In particular the maximum value of the loop counters (in our use-case, corresponding to the number of neighbor cars) must be fixed.

Moreover, we must set a bound (N) to the number of rounds that we shall unfold in the algorithm. This bound should be large enough for the generalization procedure in step 3 to work properly.

For each round of traveling, we compute a set of occurrence relations. The indexes of these occurrences tell the (logical) times of the actions that have occurred till this round. For loops, the loop control guard says that if a transition satisfies the initial condition "$k = 1$", then the transition goes into the loop. Each time after executing the loop, the variable k increases by 1. Then the transition continues to execute the loop till the condition $k \leq kMax$ is not satisfied.

We present algorithm 2 to enumerate these relations. The results of the algorithm are illustrated in the table in Fig. 14 in which the r^{th} column presents a set of occurrence relations in the round r, and the j^{th} rows presents a sequence of relations on two clock occurrences.

Example 5. *Take the component "commIni" as an example, we enumerate its occurrence relations. Let all occurrence index variables initially be* 0 *(*$m, n, r, i, j := 0$*) and the loop control variable* k *be* 1 *(*$k := 1$*). Starting from* s_0*, we get the transition* $tr0 : s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$*. From the first line of the Fig. 12, we get* $n = 1$ *(because* $m = 0$ *and* $f_{tr0} : n = m + 1$*) and so we get the relation* $!R\_0 \prec ?Cmd\_1$*. Then the transition goes to* s_1*. Since* $k := 1$*, the transition goes into the self-loop. So we get the transition* $tr2 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{!Notify}} s_1$*. From the third line of Fig. 12, we can compute* $i = 1$ *(because* $f_{tr3} : i := i + 1$*) and then we get the relation* $?Cmd\_1 \prec !Notify\_1$*. According to the loop control, we know k increases by 1 (*$k{+}{+}$*), so* $k = 2$*. Since the condition* $k \leq 2$ *still is satisfied, the transition goes into the self-loop again. According to the transition* $tr3 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{!Notify}} s_1$*, then we get the relation* $!notify\_1 \prec !notify\_2$*. Then k increases by 1 (*$k{+}{+}$*), so at this time* $k = 3$ *that cannot satisfy the condition* $k \leq 2$*. So the transition goes out of the loop, then we have* $tr4 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{\tau}} s_2$*. According to the table 12, we know* $r = 1$ *(because* $f_{tr4} : r = n$*). Then the state* s_2 *is similar as the state* s_1*. In the end of this inner loop we get the first column of the Fig. 14. Remark that the rows corresponding to transitions* $tr1$ *and* $tr5$ *from Fig. 12 have been eliminated in this process, because the corresponding loops cannot exit immediately. Then by repeating the second round, third round, etc, we can get the relations listed in the second column, the third column of Fig. 14, etc., until we rich column* N*.*

6.3.3 Step 3: Generalize the occurrence relations

In table 14, in each line we get a sequence of occurrence relations. To induce the corresponding general relation, we transfer the problem to finding a general formula for a sequence of nature numbers. We could use here standard arithmetic method that are able to deduce polynomial formulas generating natural number sequences. However, such a general approach would make difficult to estimate the minimum number of unfoldings required for finding the general formula. But in fact, due to our hypothesis on the independence of the loop control counters, the formula we seek here will be linear in the clock indexes, and the length of unfolding may be estimated from the maximum value of the loop indexes. A proof of this property, and a detailed estimation of the bound, is out of the scope of this paper. The result of generalisation is shown in "column s" in Fig. 14.

Example 6. *Let us take the second line in the table 14 as an example. The sequence occurrence index of the clock $C_{?Cmd}$ is $\{1, 2, 3, \ldots\}$. This sequence is generated by formula: $a_n = n$. The sequence occurrence index of the clock $C_{!Notify}$ is $\{1, 3, 5, \ldots\}$, that is generated by $a_n = 2n - 1$. So in the second line, the relation of the s round ($\forall s < 0$)is $?Cmd\_s \prec !Notify\_\{2s-1\}$.*

Algorithm 2 Unfold occurrence relation table

Input: A symbolic occurrence table with a clock set C with n clocks. $C = \{C_1, C_2, \ldots C_n\}$
Output: enumerate occurrence relations of N rounds in the matrix R[j][r], in which j is the index of rows and r is the index of columns (rounds).

```
for each $C_i$ do
  $Indexof(C_i) := 0$ {initialisation}
end for
set var j, r :=0
var $s := s_0$
set var $C_\alpha :=$ anyone from PreAct($s$)
set var $C_\beta :=$ one from PostAct($s$) that satisfies a certain guard
set var $s' \leftarrow \{s' | s' \xrightarrow{C_\alpha} s\}$
set var $s'' \leftarrow \{s'' | s \xrightarrow{C_\beta} s''\}$
while $r \leq N$ do
  while C $\neq \emptyset$ do
    if $s = s_0$ then
      $r++; j := 0$
    end if
    for each row in table do
      if $tr = s' \xrightarrow{C_\alpha} s \xrightarrow{C_\beta} s''$ then
        $Indexof(C_\beta) \leftarrow$ compute by $f_{tr}$
        $R[j][r] = \alpha\_Indexof(C_\alpha) \prec \beta\_Indexof(C_\beta)$
        $C \leftarrow C - C_\alpha - C_\beta$
        $j++$
        $s' \leftarrow s$; $s \leftarrow s''$; $s'' \leftarrow$ one from PostAct(s) that satisfies a certain guard;
        $C_\alpha := C_\beta$
        $C_\beta := \{C_\beta | s \xrightarrow{C_\beta} s''\}$
      end if
    end for
  end while
  reset C with n clocks $C = \{C_1, C_2, \ldots C_n\}$
end while
```

1^{st} round	2^{nd} round	3^{rd} round	s^{th} round	...	clock relations
$!R\_0 \prec ?Cmd\_1$	$!R\_1 \prec ?Cmd\_2$	$!R\_2 \prec ?Cmd\_3$	$!R\_(s-1) \prec ?Cmd\_s$	...	$C_{!R} \prec C^{\Delta(1)}_{?Cmd}$
$\boxed{?Cmd\_1 \prec !notify\_1}$	$\boxed{?Cmd\_2 \prec !notify\_3}$	$\boxed{?Cmd\_3 \prec !notify\_5}$	$\boxed{?Cmd\_s \prec !notify\_(2s-1)}$	...	$C_{?Cmd} \prec C^{\{2s-1\}}_{!notify}$
$!notify\_1 \prec !notify\_2$	$!notify\_3 \prec !notify\_4$	$!notify\_5 \prec !notify\_6$	$!notify\_(2s-1) \prec !notify\_2s$	...	$C^{\{2s-1\}}_{!notify} \prec C^{\{2s\}}_{!notify}$
$!notify\_2 \prec \tau\_1$	$!notify\_4 \prec \tau\_2$	$!notify\_6 \prec \tau\_3$	$!notify\_2s \prec \tau\_s$	...	$C^{\{2s\}}_{!notify} \prec C_{\tau}$
$\tau\_1 \prec ?ack\_1$	$\tau\_2 \prec ?ack\_3$	$\tau\_3 \prec ?ack\_5$	$\tau\_s \prec ?ack\_(2s-1)$	...	$C_{\tau} \prec C^{\{2s-1\}}_{?ack}$
$?ack\_1 \prec ?ack\_2$	$?ack\_3 \prec ?ack\_4$	$?ack\_5 \prec ?ack\_6$	$?ack\_(2s-1) \prec ?ack\_2s$	...	$C^{\{2s-1\}}_{?ack} \prec C^{\{2s\}}_{?ack}$
$?ack\_2 \prec !R\_1$	$?ack\_4 \prec !R\_2$	$?ack\_6 \prec !R\_3$	$?ack\_2s \prec !R\_s$	...	$C^{\{2s\}}_{?ack} \prec C_{!R}$

Figure 14: Steps 2-3-4: Unfold rounds, generalize, and deduce clock relations

6.3.4 Step4: lifting to clock relations

In the last step, we lift the concurrence relations to clock relations, using the clock operators "lift" and "filter" from definitions 3 and 4. This step is straightforward, and the result is shown in the last column of Fig. 14.

7 Generating the timed specification of a timed-pNet

A timed-pNets node actually consists of a set of holes (J) with timed specifications (TS_j), synchronous vectors (V_i), and global clocks (C_G) generated from the synchronous vectors. Therefore, generating the external timed specification for a timed-pNets node (called global timed specification TS_g) boils down to compute the global clock relations from the local timed-specifications of its holes (TS_j) and the coincidence relations deduced from the synchronous vectors (V_i), using the properties on clock relations from section 2.2. Formally:

Definition 14 (Global Clock Relation Set)**.** *Given a timed-pNet $T\text{-}pNets = < P, A_G, C_G, J, \widetilde{A}_J, \widetilde{C}_J, \widetilde{R}_J, \overrightarrow{V} >$ The global time specification of T-pNets is the pair $< C_G, \mathcal{R}_G >$, where $\mathcal{R}_G$ is the Global Clock Relation Set deduced from:*
- all local clocks relations $\mathcal{R}_j$ from its holes,
- the (coincidence) relations deduced from all its synchronization vectors
- symmetry and transitivity of coincidence, transitivity of precedence.

During this logical saturation process, it may happen that contradictory relations are deduced, when 2 clocks would be proved both coincident and precedent, or precedent both ways. This we call a conflict:

Definition 15 (Clock Conflicts)**.** *Given a timed specification $< C, \mathcal{R} >$:*
- two clocks C_α and C_β in C are in conflict if either $C_\alpha = C_\beta \wedge (C_\alpha \prec C_\beta \cup C_\beta \prec C_\alpha) \in \mathcal{R}$ or $C_\alpha \prec C_\beta \wedge C_\beta \prec C_\alpha \in \mathcal{R}$
- the Global Clock Conflict Set of a timed-pNet is the set of pairs of clocks in conflict in its Global Clock Relation Set.

Example 7. *Let's take Fig. 9 as an example. From the user specification in example 3 (page 11), we know the clock relations of these holes are:*

- $R_{\{CommIni\}} = \{C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify},\ C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify},\ C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack},\ C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack},\ C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack},\ C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_{!R} \prec C^{\Delta(1)}_{?cmd}\}$
- $R_{\{ChannelNtf[m]\}} = \{C_{c.?notify_{[m]}} \prec C_{c.!notify_{[m]}} \prec C_{c.?notify^{\Delta(1)}_{[m]}}\}$,
- $R_{\{ChannelAck[m]\}} = \{C_{c.?ack_{[m]}} \prec C_{c.!ack_{[m]}} \prec C^{\Delta(1)}_{c.?ack_{[m]}}\}$,
- $R_{\{CommRes[m]\}} = \{C_{?notify_{[m]}} \prec C_{!ack_{[m]}} \prec C^{\Delta(1)}_{?notify_{[m]}}\}$.

Besides, we derive the clock relations from the synchronous communications defined by synchronous vectors as:

- $R_{V_1} = \{C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} = C_{c.?notify_{[1]}} = C_{notify_{g1_{[1]}}}\}$,
- $R_{V_2} = \{C_{c.!notify_{[1]}} = C_{?notify_{[1]}} = C_{notify_{g2_{[1]}}}\}$,
- $R_{V_3} = \{C_{!ack_{[1]}} = C_{c.?ack_{[1]}} = C_{ack_{g3_{[1]}}}\}$,
- $R_{V_4} = \{C_{c.!ack_{[1]}} = C^{\{2s-1\}}_{?ack} = C_{ack_{g4_{[1]}}}\}$,
- $R_{V_5} = \{C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} = C_{c.?notify_{[2]}} = C_{notify_{g1_{[2]}}}\}$,
- $R_{V_6} = \{C_{c.!notify_{[2]}} = C_{?notify_{[2]}} = C_{notify_{g2_{[2]}}}\}$,
- $R_{V_7} = \{C_{!ack_{[2]}} = C_{c.?ack_{[2]}} = C_{ack_{g3_{[2]}}}\}$,
- $R_{V_8} = \{C_{c.!ack_{[2]}} = C^{\{2s\}}_{?ack} = C_{ack_{g4_{[2]}}}\}$,
- $R_{V_9} = \{C_{?Cmd} = C_{?Cmd_{g5}}\}$,
- $R_{V_{10}} = \{C_{!R} = C_{!R_{g6}}\}$.

Take e.g. the relation between the global clocks $C_{notify_{g1_{[1]}}}$ and $C_{notify_{g2_{[1]}}}$. They are generated by the synchronous vectors V_1 and V_2, and we deduce, from the relations of hole $ChannelNtf_{[1]}$ and the relations of these two vectors, that:
$C_{notify_{g1_{[1]}}} =_{(R_{V_1})} C_{c.?notify_{[1]}} \prec_{(R'_{ChannelNtf[1]})} C_{c.!notify_{[1]}} =_{(R_{V_2})} C_{notify_{g2_{[1]}}}$, *and conclude* $C_{notify_{g1_{[1]}}} \prec C_{notify_{g2_{[1]}}}$.

The formal definition above is not very practical. We will show now that from a simple case analysis on the interaction between the synchronization vectors, we can compute a set of global clock relations that is sufficient to generate the Global Clock Relation Set. The following theorem defines the case analysis procedure, and states its correctness (all relations computed are correct). The next theorem will prove its completeness. In one particular case, this case analysis procedure may detect a local conflict between two global actions, more precisely between two synchronization vectors representing communication between the same 2 holes. In this case, we shall signal the conflict, but produce no relations between these actions. Other types of conflicts could be created by configurations involving more than 2 holes. These cannot be detected at the level of this case-analysis procedure; a full conflict detection procedure is out of the scope of this paper.

Theorem 2 (Global clock relation analysis). *Given a timed-pNet T-pNets* $=< P, A_G, C_G, J, \widetilde{A}_J, \widetilde{C}_J, \widetilde{R}_J, \overrightarrow{V} >$. *Let H_α, H_β, H_γ be three holes of T-pNets and $C_{H_\alpha}, C_{H_\beta}, C_{H_\gamma} \subset \widetilde{C}_J$ be the sets of clocks of holes H_α, H_β and H_γ. Let the clocks $C_{\alpha_1}, C_{\alpha_2} \in C_{H_\alpha}$, the clocks $C_{\beta_1}, C_{\beta_2} \in C_{H_\beta}$, the clock $C_{\gamma_1} \in C_{H_\gamma}$, with $C_{H_\alpha} \bigcap C_{H_\beta} \bigcap C_{H_\gamma} = \varnothing$). For each pair of global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$, we enumerate the pairs of synchronization vectors able to generate them, and match them with the following cases (note that both pairs $(C_{a_{g1}}, C_{a_{g2}})$ and $(C_{a_{g2}}, C_{a_{g1}})$ will be enumerated, so we do not consider symmetric conditions in the cases below). Each match may add a clock relation in the Global Clock Relation Set $\mathcal{R}$:*

- ***(Case1:)*** *If the global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from synchronous vectors*
  $< \ldots, C_{\alpha_1}, \ldots, C_{\beta_1}, \ldots > \rightarrow C_{a_{g1}}$ *and*
  $< \ldots, C_{\alpha_2}, \ldots, C_{\beta_2}, \ldots > \rightarrow C_{a_{g2}}$,
  which are related to two holes C_{H_α} and C_{H_β} as shown in Fig. 15(1), then
  - *if* $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} = C_{\beta_2}$ *then* $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$.

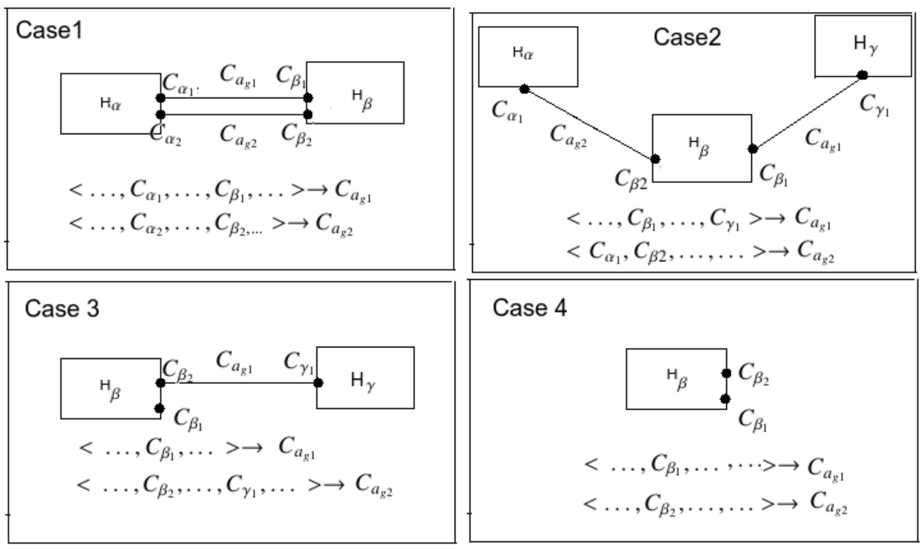

Case1
H_α
H_β
C_{α_1}
$C_{a_{g1}}$
C_{β_1}
C_{α_2}
$C_{a_{g2}}$
C_{β_2}
$< \dots, C_{\alpha_1}, \dots, C_{\beta_1}, \dots > \rightarrow C_{a_{g1}}$
$< \dots, C_{\alpha_2}, \dots, C_{\beta_2, \dots} > \rightarrow C_{a_{g2}}$
Case2
H_α
H_γ
H_β
C_{α_1}
$C_{a_{g2}}$
C_{γ_1}
$C_{a_{g1}}$
$C_{\beta 2}$
C_{β_1}
$< \dots, C_{\beta_1}, \dots, C_{\gamma_1} > \rightarrow C_{a_{g1}}$
$< C_{\alpha_1}, C_{\beta 2}, \dots, \dots > \rightarrow C_{a_{g2}}$
Case 3
H_β
H_γ
C_{β_2}
$C_{a_{g1}}$
C_{γ_1}
C_{β_1}
$< \dots, C_{\beta_1}, \dots > \rightarrow C_{a_{g1}}$
$< \dots, C_{\beta_2}, \dots, C_{\gamma_1}, \dots > \rightarrow C_{a_{g2}}$
Case 4
H_β
C_{β_2}
C_{β_1}
$< \dots, C_{\beta_1}, \dots, \dots > \rightarrow C_{a_{g1}}$
$< \dots, C_{\beta_2}, \dots, \dots > \rightarrow C_{a_{g2}}$

Figure 15: The 4 cases of theorem 2

- *if* $C_{\alpha_1} \prec C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$ *then* $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- *if* $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$ *or if* $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_2} \prec C_{\beta_1}$ *then conflict found.*

- ***(Case2:)*** *If the global clock* $C_{a_{g1}}$ *and* $C_{a_{g2}}$ *are generated from the synchronous vectors* $< \dots, C_{\beta_1}, \dots, C_{\gamma_1} > \rightarrow C_{a_{g1}}$ *and* $< C_{\alpha_1}, C_{\beta_2}, \dots, \dots > \rightarrow C_{a_{g2}}$*, which are related to three holes* C_{H_α}*,* C_{H_β} *and* C_{H_γ} *as shown in Fig. 15, then*
  - *if* $C_{\beta_1} = C_{\beta_2}$ *then* $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
  - *if* $C_{\beta_1} \prec C_{\beta_2}$ *then* $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- ***(Case3:)*** *If the global clock* $C_{a_{g1}}$ *and* $C_{a_{g2}}$ *are generated from the synchronous vectors* $< \dots, C_{\beta_1}, \dots > \rightarrow C_{a_{g1}}$ *and* $< \dots, C_{\beta_2}, \dots, C_{\gamma_1}, \dots > \rightarrow C_{a_{g2}}$ *as shown in Fig. 15(3). then*
  - *if* $C_{\beta_1} = C_{\beta_2}$ *then* $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
  - *if* $C_{\beta_1} \prec C_{\beta_2}$ *then* $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- ***(Case4:)*** *If the global clock* $C_{a_{g1}}$ *and* $C_{a_{g2}}$ *are generated from the synchronous vectors* $< \dots, C_{\beta_1}, \dots > \rightarrow C_{a_{g1}}$ *and* $< \dots, C_{\beta_2}, \dots, \dots > \rightarrow C_{a_{g2}}$ *as shown in Fig. 15(4). then*
  - *if* $C_{\beta_1} = C_{\beta_2}$ *then* $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
  - *if* $C_{\beta_1} \prec C_{\beta_2}$ *then* $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- ***(Otherwise)*** *In any other case, this pair of clocks is **NOT** directly related in* $\mathcal{R}$

Proof. For each of the cases, we prove that the deduced relation is indeed correct with respect to definition 14.

- **Case1:** From the two synchronous vectors $< \dots, C_{\alpha_1}, \dots, C_{\beta_1}, \dots > \rightarrow C_{a_{g1}}$, $< \dots, C_{\alpha_2}, \dots, C_{\beta_2}, \dots > \rightarrow C_{a_{g2}}$,
  we know that $C_{\alpha_1} = C_{\beta_1} = C_{a_{g1}}$ and $C_{\alpha_2} = C_{\beta_2} = C_{a_{g2}}$. (1) If $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} = C_{\beta_2}$, according to the transitivity property of "=", we get the relation $C_{a_{g1}} = C_{a_{g2}}$.

(2) If $C_{\alpha_1} \prec C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$, then we have $C_{a_{g1}} = C_{\alpha_1} \prec C_{\alpha_2} = C_{a_{g2}}$. So using substitutivity of $=$ w.r.t. $\prec$, we get the relation $C_{a_{g1}} \prec C_{a_{g2}}$.

- **Case2:** From the two synchronous vectors $< \ldots, C_{\beta_1}, \ldots, C_{\gamma_1} > \rightarrow C_{a_{g1}}$ and $< C_{\alpha_1}, C_{\beta_2}, \ldots, \ldots > \rightarrow C_{a_{g2}}$, we know that $C_{\beta_1} = C_{\gamma_1} = C_{a_{g1}}$ and $C_{\alpha_1} = C_{\beta_2} = C_{a_{g2}}$. (1) If $C_{\beta_1} = C_{\beta_2}$, then according to the transitivity property of "=", we know that $C_{a_{g1}} = C_{a_{g2}}$. (2) If $C_{\beta_1} \prec C_{\beta_2}$, since $C_{a_{g1}} = C_{\beta_1} \prec C_{\beta_2} = C_{a_{g2}}$, then we have the relation $C_{a_{g1}} \prec C_{a_{g2}}$.

- **Case3 and Case4:** The proofs are similar to Case2.

□

Example 8. *Let's take again Fig. 9 as an example to compute the clock relation between* $C_{notify_{g2_{[1]}}}$ *and* $C_{ack_{g3_{[1]}}}$. *We know the two global actions are generated by the vectors*
V_2*:* $< \ldots, C_{c.!notify_{[1]}}, \ldots, C_{?notify_{[1]}}, \ldots > \rightarrow C_{notify_{g2_{[1]}}}$ *and*
V_3 *:* $< \ldots, C_{!ack_{[1]}}, \ldots, C_{c.?ack_{[1]}}, \ldots > \rightarrow C_{ack_{g3_{[1]}}}$,
so we are in case 2), and we know from $TS_{\{CommRes_{[1]}\}}$ *that* $C_{?notify_{[1]}} \prec C_{!ack_{[1]}}$, *so we conclude* $C_{notify_{g2_{[1]}}} \prec C_{ack_{g3_{[1]}}}$.

Theorem 3 (Completeness). *There exist four and only four combinations of synchronous vectors, as listed in Theorem 2, for deducing a relation between a pair of global clocks.*

Proof. From the timed-pNets definition, we know that there are two ways to build a global clock: binary communication and visibility. So there are 3 combinations:

(1) both global clocks are generated by binary communication

(2) one global clock is generated by binary communication and another one is generated by visibility

(3) both global clocks are generated by visibility

Now we analyze the three situations one by one.
Given a timed-pNet *T-pNets* $=< P, A_G, C_G, J, \widetilde{A}_J, \widetilde{C}_J, \widetilde{R}_J, \overrightarrow{V} >$.
(1) Let $< \ldots, C_\alpha, \ldots, C_\beta > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, C_\eta > \rightarrow C_{g2}$ (C_α, C_β, C_γ,$C_\eta \in \widetilde{C}_J$) be two synchronous vectors generating the global clocks C_{g1} and C_{g2}. Obviously the four local clocks C_α, C_β, C_γ,C_η cannot be in one hole since the synchronous vectors build binary communications between holes. If the four local clocks come from two holes, then the possible combinations are C_α and C_γ are in one hole, the other two are in another hole. Or C_α and C_η are in one hole, the other two are in another hole. Case 1 of the theorem 2 covers the both situations. If the four local clocks come from three holes, then any two local clocks that come from different synchronous vectors must be in one hole, and the rest two local clocks are in other two different holes. For example, C_α and C_γ are in one hole, the other two are in other two holes separately. Case 2 of the theorem 2 covers the situation. Furthermore, the four local clocks cannot be in 4 holes (or more than 4 holes), otherwise there is no local clock relations in $\widetilde{R}_J$ can be used to deduce global clock relations, so no direct clock relation can be built between C_{g1} and C_{g2}.
(2) Let $< \ldots, C_\alpha, \ldots, C_\beta > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, > \rightarrow C_{g2}$ (C_α, C_β, $C_\gamma \in \widetilde{C}_J$) be two synchronous vectors to generate the global clocks C_{g1} and C_{g2}. Similar to the proof in the previous situation, the three local clocks cannot be in one hole and cannot be in 3 holes or more. So the only possible combination is that C_γ is in the same hole that one of the others. Case 3 of the theorem 2 covers the situation.
(3) Let $< \ldots, C_\alpha, \ldots, > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, > \rightarrow C_{g2}$ (C_α, $C_\gamma \in \widetilde{C}_J$) be two synchronous vectors

to generate the global clocks C_{g1} and C_{g2}. The two local clocks cannot be in 2 different holes, otherwise there is no local relation can be find between them. So the only possible situation is the two local clocks are in the same hole. Case 3 of the theorem 2 covers the situation.
In conclusion, if the relation of two global clock $C_{g1}, C_{g2} \in C_G$ can be deduced by the local clock relations from $\widetilde{R}_J$, they must belong to one of the four cases of theorem 2. $\square$

8 Assembling a multi-layers timed-pNets system

After generating a timed specification for a timed-pNets node, we can use the generated timed specification to prove that it would be compatible with the specification of a hole of a higher-level pNet. This way, a layered tree structure can be built as shown in the Fig. 16. In this structure, each layer uses an abstraction of its lower layer. The clocks in the lower layer (at level N) are transparent to its abstract layer (at level N+1) in which only holes with its timed specification (TS_j), synchronous vectors (V_i) and global clocks (C_g) can be seen.

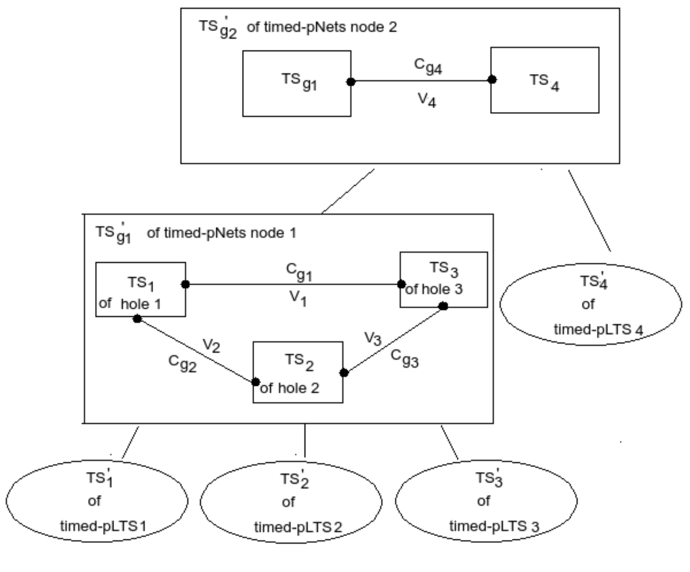


Figure 16: Layered Structure

As we have already mentioned, this construction can be done in a very flexible way either bottom-up or top-down. The result timed-pNet system can be open (if it still contains some unfilled holes at the leaves), or closed if all holes are filled with timed-pNets and timed-pLTS.

Example 9. *We now have all elements required for checking the compatibility of our timed-pLTSs with the holes of the upper layer pNet. Let us look at "CommIni" as an example:*

- *the relation set of the hole "CommIni" for open timed-pNets is* $R_{CommIni} = \{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}, C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}}, C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}}, C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}, C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}, C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$*,*

- *the relation set of the "CommIni" timed-pLTS component from Fig. 14 as* $R'_{Commini} = \{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{\tau} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$.

Since we can easily get $R_{\{CommIni\}} \subseteq R'_{Commini}$*, according to Inclusion definition we have* $TS_{\{CommIni\}} \ll TS'_{Commini}$ *. Therefore, from the compatibility definition, we know that the "CommIni" timed-pLTS is compatible with the hole "CommIni".*

The validations that have been defined in our paper, namely the compatibility of hole composition, and the conflict detection between timed-pNets synchronization vectors, ensure some specific validity properties of the global Time Specification of the system, as defined by Definition 14. However, this does not mean that there cannot be more complex conflicts in the interaction between more than 2 holes of a timed-pNets, or more specific timed properties that can be computed from refined implementations of some sub-nets. In the next section, we show how to use simulation with the TimeSquare tool, to address such cases.

9 Simulation

In this section we explain how to use TimeSquare [DM12] to detect complex conflicts of timed-pNets. Two inputs are required by TimeSquare (see the Fig. 17). One is an open timed-pNets system. Another is a set of refined implementations. If a closed timed-pNets composed by those refined implementations has no conflict, we say the closed timed-pNets is safe. Otherwise, the TimeSquare reports violations, which means that conflicts exist in the closed timed-pNets system. Before running simulations, the two inputs are translated into timed specifications that are acceptable format for TimeSquare. The way of generating timed specification is described in section 6 and 7.

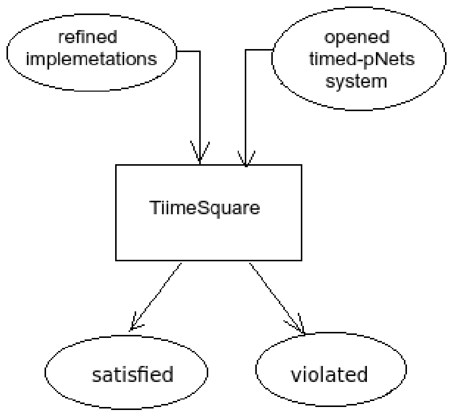


Figure 17: Property Checking by TimeSquare

9.1 Simulation 1:

- We take the system shown in the Fig. 9 as an example. We first build an open timed-pNet node with the timed specifications of holes (TS: $TS_{\{CommIni\}}$, $TS_{\{ChannelNtf[m]\}}$, $TS_{\{ChannelAck[m]\}}$, $TS_{\{CommRes[m]\}}$) and synchronous vectors (V_i), by which we can generate global clock relations (we call it an abstract specification). From section 7, we can get the abstract specification $TS_g =< C_g, R_g >$ with $R_g = \{C_{?Cmd_{g5}} \prec C_{notify_{g1_{[m]}}} \prec C_{notify_{g2_{[m]}}} \prec C_{ack_{g3_{[m]}}} \prec C_{ack_{g4_{[m]}}} \prec C_{!R_{g6}};$ $C_{notify_{g1_{[1]}}} \prec C_{notify_{g1_{[2]}}}; C_{ack_{g4_{[1]}}} \prec C_{ack_{g4_{[2]}}}\}$. Then we import the timed specifications of the refined implementations of those holes (TS': $TS'_{\{CommIni\}}$, $TS'_{\{ChannelNtf[m]\}}$, $TS'_{\{ChannelAck[m]\}}$, $TS'_{\{CommRes[m]\}}$) to replace TS. The timed-pNets node that composed by these refined implementations is called closed timed-pNets node. And its global clock relations is named concrete specification TS'_g.

- *Result of Simulation 1:* The Fig. 18 illustrates the concrete specification TS'_g. In this figure, each line demonstrate a clock and the red arrows demonstrates the precedence relations. For simplification, here we represent two cycles of simulation. From the figure we can see that the abstract specification TS_g is satisfied by the refined concrete system since we have $TS_g \ll TS'_g$.

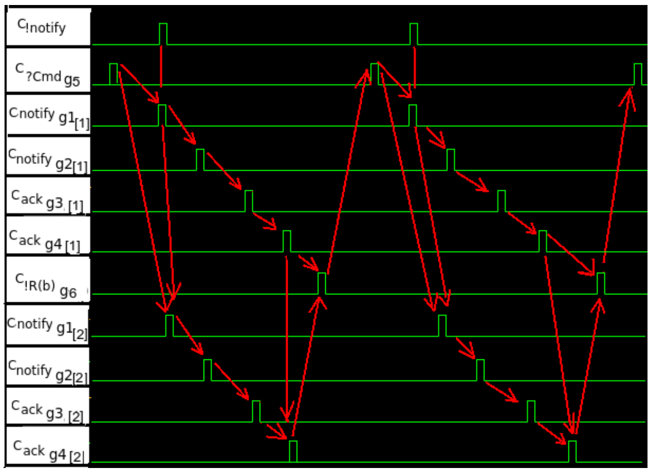
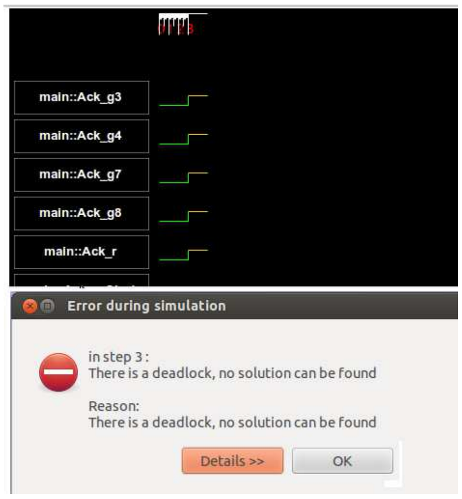


Figure 18: system's specification checking

9.2 Simulation 2:

- In this simulation, we choose $TS'_{\{UpdatedCommIni\}} = \{C_{?Cmd} \prec C^{\{2s-1\}}_{!Notify} \prec C^{\{2s-1\}}_{?Ack} \prec C^{\{2s\}}_{!Notify} \prec C^{\{2s\}}_{?Ack} \prec C_{!R} \prec C^{\Delta(1)}_{?Cmd}\}$, $TS'_{\{UpdatedCommRes[m]\}} = \{C_{?NotifyInfo_{[m]}} \prec C_{ExchangeInfo_{[m]}} \prec C_{!Ack_{[m]}}\}$ and we add a synchronous vector between hole $CommRes[1]$ and $CommRes[2]$ to get a new relation $R_{V_{new}} = \{C_{ExchangeInfo_{[1]}} = C_{ExchangeInfo_{[2]}} = C_{ExchangeInfo_{g11}}\}$. Obviously, the updated implementation of hole $CommIni$ is compatible with the abstract timed specification of this hole $TS_{\{CommIni\}}$ since we have $TS_{\{CommIni\}} \ll TS'_{\{UpdatedCommIni\}}$. And the same to the other two holes $CommRes[m]$ since we have $TS_{\{CommRes[m]\}} \ll TS'_{\{UpdatedCommRes[m]\}}$.
- *Result of simulation 2:* By simulation, we found violations as shown in Fig.19.



Error during simulation

in step 3 :
There is a deadlock, no solution can be found

Reason:
There is a deadlock, no solution can be found

Figure 19: Conflict Detected

- *Analyzing the result:* By analyzing our updated closed timed-pNets, we found the conflict is caused by a cycle represented in the Fig.20. In this Figure, according to the theorem 2, we can get the set of global relations as $\{C_{Notify_{g1_{[2]}}} \prec C_{Notify_{g2_{[2]}}} \prec C_{ExchangeInfo_{g7}} \prec C_{Ack_{g3_{[1]}}} \prec C_{Ack_{g4_{[1]}}}\}$. Obviously, relation $\{C_{Notify_{g1_{[2]}}} \prec C_{Ack_{g4_{[1]}}}\}$ is hold in terms of the transitivity property of precedence relations. However, by using the theorem 2 again, from the $TS'_{\{UpdatedCommIni\}}$ we can get the relation $\{C_{Ack_{g4_{[1]}}} \prec C_{Notify_{g1_{[2]}}}\}$ which is contradict with the relation $\{C_{Notify_{g1_{[2]}}} \prec C_{Ack_{g4_{[1]}}}\}$. To fix the conflict, we need to find another implementation that still compatible with these holes but without making conflicts. For our example, we can just simply change the $TS'_{\{UpdatedCommIni\}}$ to $TS'_{\{FixedCommIni\}} = \{C_{?Cmd} \prec C^{\{2s-1\}}_{!Notify} \prec C^{\{2s\}}_{!Notify} \prec C^{\{2s-1\}}_{?Ack} \prec C^{\{2s\}}_{?Ack} \prec C_{!R} \prec C^{\Delta(1)}_{?Cmd}\}$. And in the end, by simulation, no conflict exists any more.

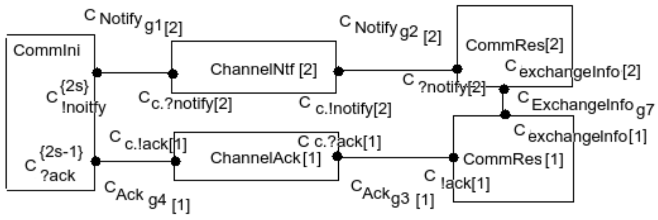


Figure 20: system's specification checking

10 Conclusion

This paper proposed a flexible time-related behavioral semantic model (called Timed-pNets) for modeling communication behavior of distributed systems. We specify a system with several components and communications between them. We are able to build a hierarchical tree structure for composing complicated component-based systems. The refinement and compatibility are considered in the paper. An concrete example throughout the paper is given to represent how to build a hierarchical specification and how to refine the system. In the end, we use TimeSquare to check the compatibility of the refined system.

Three advantages are implied in our model: first, by introducing logical clock relations, timed-pNets model is able to specify the system's time-related communication behavior constrains without relying on physical common clock; second, by using timed specifications, our model is easy to be composed and has the capability of building a hierarchical structure; the last but not the least, our model can flexible model heterogeneous communication including synchronous and asynchronous communications by introducing channel LTS. We believe that the timed-pNets model is helpful for analyzing the time-related behaviors for distributed systems including cyber physical systems.

After checking the system compatibility, another interesting point is to check system's physical time constrains such as deadline property that expresses whether system communications can be successfully finished before a certain deadline. To check this, we shall choose a reference clock and specify the delay constrains in terms of the reference clock. In this paper, even though we define delay variables for actions, we do not provide a way to specify delay constrains here, because it is not a main topic for this paper. In our future work we will extend the model for specifying the delay constrains and check system's time properties. Furthermore, we plan to extend the current model timed-pNets to a new duration-pNets so that we can describe the system's behaviors whose execution takes time. Meanwhile, we will propose a way to define delay constrains for checking the system's time properties. It also looks interesting to translate our system to boolean automata to verify the system's properties.

11 Related work

Some formal models or frameworks with time constraints have been proposed to describe time systems.

Globally Asynchronous Locally Synchronous (GALS) [Cha84] is a model of computation that allows to design computer systems consisting of several synchronous components interacting with each other with asynchronous communication, e.g., FIFOs. It can be used both in software and hardware. In software, these synchronous components usually are specified as Finite State Machines (FSMs) and the asynchronous communication between them is modeled with a buffer [CGJ$^+$94]. The idea of the GALS approach provides a methodology for combining concurrent embedded systems within loosely coupled systems. Similar to the idea of GALS, M. Serrano designed the HipHop language [BNS11] that follows the synchronous reactive model of the Multiclock Esterel [BS01] to specify reactive program by dealing with abstract events and their reactions.

Our model partly takes this idea to specify both synchronous and asynchronous communication. The main difference is that we specify the synchronous components as timed specifications (a set of clocks and clock relations) instead of FSMs. Moreover, the synchronous communications are specified by the coincidence relations between clocks that come from different timed specifications, while the asynchronous communications are modeled by channels in which precedence relations are applied on two clocks.

Timed-automata [AD94] is famous for modeling the behavior of real-time systems. They provided a simple and powerful way to annotate state transition graphs with time constraints using finitely real-value clocks. Closure properties, decision problems as well as automatic verification of real-time requirements were considered in timed-automata and supported by several tools like UPPAAL [BLL$^+$95]. Timed-automata can be a good reference for building and verifying timed models. However, the clocks in timed-automata are a finite set of real-valued clocks whose values increase all with the same speed. This feature does not help us to model systems consisting of independent-clock devices, since these clocks from different devices may have different speed.

BIP [BBS06] is a framework for the incremental composition of heterogeneous components. It allows building complex systems by thecoordinating the behavior of a set of atomic components. The methodology based on the theory that components are obtained as the superposition of three layers. The lowest layer are the component behaviors that are described as automata extended with data and functions. The intermediate layer includes a set of connectors describing the interactions between transitions of the behaviors. The upper layer is a set of priority rules describing scheduling policies for interactions. BIP encompasses heterogeneity. It provides a powerful mechanism for structuring interactions involving strong synchronization (rendezvous) or weak synchronization (broadcast). Synchronous execution is characterized as a combination of properties of the three layers. However, modeling timed components in BIP involves references to a specific "tick" port expressing the passage of (discrete) time, and such "tick" events must be synchronized between various components of a system, before computing worst case execution time (WCET) or task period properties. With contrast to this approach, we do not want to assume the existence of a single reference clock, but rather let the various clocks as unrelated as possible.

Modeling and Analysis of Real-Time and Embedded System (MARTE)[www] is an special extension of UML for modeling real-time embedded systems. It defines the standard model-based description for real-time and embedded systems and provides facilities to annotate models with information required to perform specific analysis. It supports modeling and analysis of component-based architectures, as well as a variety of different computational paradigms (asynchronous, synchronous, and timed). Recently, the idea of logical time and Clock Constraint Specification Language (CCSL) has been introduced into MARTE for its extension of modeling distributed systems. So it can be used to check the communication and causality path correctness by introducing event relations into models. However MARTE UML is large and complex. It comprises many different concepts and semantics that we do not need. Since we would like to keep the semantic as simple as possible, we choose pNet instead of MARTE to imply our idea.

Programming Temporally Integrated Distributed Embedded Systems (PTIDES) [ELM$^+$12] serves as a coordination language for model-based design of distributed real-time embedded systems. It extends the discrete-event model of computation with a carefully chosen relationship between real time and model time. PTIDES provides a framework for exploring a family of execution strategies for distributed embedded systems so that it can directly confront the multiform nature of time in distributed systems. Simulations of it can simultaneously have many time lines, with events that are logically or physically placed on these time lines. As far as we know, they have some semantics for the interactions between events to model the communications of distributed systems. However, we need more mature communication mechanism like asynchronous communication, broadcasting, as well as more compatible way of modeling system, which are not yet supported in PTIDES.

Timed Petri nets (TdPNs) [VRdFECG99] is one of several mathematical modeling languages for the description of distributed systems. It is widely used for the modeling and analysis of concurrent systems with time-dependent behavior like communication systems. It includes a set of directed bipartite graphs, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre- and/or post- conditions for which transitions (signified by arrows). Each arc associates with an interval (or bag of intervals). In TdPNs, each token has an age. This age is initially set to a value belonging to the interval of the arc which has produced it or set to zero if it belongs to the initial marking. Afterwards, ages of tokens evolve synchronously with time. A transition may be fired if tokens with age belonging to the intervals of its input arcs may be found in the current configuration. Compare to timed Petri nets we use a total different way by means of label transition system (LTS) to model the system's behavior. Our model graph comprises some number of states, with arcs between them labeled by activities of the system. We choose to model our system by means of action based LTS because: 1) Our goal is to check the correctness of system's communication behavior, not to verify the correctness of programming computations. So we hide the unnecessary detail information like state variables, and just highlight the information that related to communication behavior like actions. 2) By this way, we can easily model our system in a compact and hierarchical way since the action based LTS do not show the details information of states (each state is an abstract node).

Spatio-temporal consistence language (STeC) [Che12], provides a location-triggered specification as well as operational semantics and denotational semantics [WCZ13] for describing distributed system with time and location constraints. Syntax and semantics of the language have been proposed to address the issue of spatial-temporal consistence for real-time systems. The language specifies the time and location constrains for each action, and then computes the execution time of processes. However, since our model mainly focus on time properties, we set the information as parameters that are sent by physical part of our system. Typically, in our use-case we sometimes need check car's locations, but such data is not treated at the same level as time information. This is quite different from what the STeC does by adding location constrains. This way, we can separate our model from physical part of system and focus on modeling the communication behavior of Cyber part.

These previous efforts are important since they provide crucial insights on building the timed-model for real-time systems, or contribute to mechanisms and strategies that can be used to model our system.

Thanks. This work was partially funded by the INRIA Associated Team DAESD between INRIA and ECNU; by NSFC (No. 61021004 and No. 61370100); by Shanghai Knowledge Service Platform Project (No. ZF1213). And we give great thanks to Frederic MALLET who took time to discuss with us and gave us bunches of advice.

References

[ABHMS12] Rabéa Ameur-Boulifa, Ludovic Henrio, Eric Madelaine, and Alexandra Savu. Behavioural Semantics for Asynchronous Components. Rapport de recherche RR-8167, INRIA, December 2012.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2):183 – 235, 1994.

[And09] Charles André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Rapport de recherche RR-6925, INRIA, 2009.

[AP94] André Arnold and John Plaice. Finite transition systems: semantics of communicating systems. Prentice Hall International (UK) Ltd., 1994.

[Arn94] André Arnold. Finite transition systems: semantics of communicating systems. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.

[BBC+09] Tomás Barros, Rabéa Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed Fractal components. Annals of Telecommunications, 64(1–2), jan 2009. also Research Report INRIA RR-6491.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), 11-15 September 2006, Pune, India, pages 3–12. IEEE Computer Society, 2006.

[BDS91] Frédéric Boussinot and Robert De Simone. The esterel language. Proceedings of the IEEE, 79(9):1293–1304, 1991.

[Ber00] Gérard Berry. The foundations of esterel. In Proof, language, and interaction, pages 425–454, 2000.

[BLGJ91] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: the signal language and its semantics. Science of computer programming, 16(2):103–149, 1991.

[BLL+95] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In Proc. of Workshop on Verification and Control of Hybrid Systems III, LNCS 1066, pages 232–243. Springer–Verlag, October 1995.

[BNS11] Gérard Berry, Cyprien Nicolas, and Manuel Serrano. Hiphop: a synchronous reactive extension for hop. In Proceedings of the 1st ACM SIGPLAN international workshop on Programming language and systems technologies for internet clients, pages 49–56. ACM, 2011.

[BS01] Gérard Berry and Ellen Sentovich. Multiclock esterel. In Correct Hardware Design and Verification Methods, pages 110–125. Springer, 2001.

[CCM12] Yanwen Chen, Yixiang Chen, and Eric Madelaine. Timed-pNets: A formal communication behavior model for real-time cps systems. In Workshop on Trustworthy Cyber Physical Systems, TCPS, sep 2012.

[CGJ+94] Massimoliano Chiodo, Paolo Giusto, Attila Jurecska, Harry C Hsieh, Alberto Sangiovanni-Vincentelli, and Luciano Lavagno. Hardware-software codesign of embedded systems. Micro, IEEE, 14(4):26–36, 1994.

[Cha84] D. M. Chapiro. Globally-asynchronous locally-synchronous systems. PhD thesis, Stanford Univ., CA., October 1984.

[Che12] Yixiang Chen. Stec: A location-triggered specification language for real-time systems. In ISORC Workshops, pages 1–6. IEEE, 2012.

[CHS08] Denis Caromel, Ludovic Henrio, and Bernard Paul Serpette. Asynchronous sequential processes. Information and Computation, Volume 207, Issue 4, 2008.

[DM12] Julien Deantoni and Frédéric Mallet. TimeSquare: Treat your Models with Logical Time. In Sebastian Nanz Carlo A. Furia, editor, TOOLS - 50th International Conference on Objects, Models, Components, Patterns - 2012, volume 7304 of Lecture Notes in Computer Science - LNCS, pages 34–41, Prague, Tchèque, République, May 2012. Czech Technical University in Prague, in co-operation with ETH Zurich, Springer.

[ELM+12] John Eidson, Edward A. Lee, Slobodan Matic, Sanjit A. Seshia, and Jia Zou. Distributed real-time software for cyber-physical systems. Proceedings of the IEEE (special issue on CPS), 100(1):45 – 59, January 2012.

[Fid91] Colin Fidge. Logical time in distributed computing systems. Computer, 24(8):28–33, 1991.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7):558–565, 1978.

[VRdFECG99] V Valero Ruiz, David de Frutos Escrig, and F Cuartero Gomez. On non-decidability of reachability for timed-arc petri nets. In Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on, pages 188–196. IEEE, 1999.

[WCZ13] Hengyang Wu, Yixiang Chen, and Min Zhang. On denotational semantics of spatial-temporal consistency language–stec. In Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on, pages 113–120. IEEE, 2013.

[www] Modeling and analysis of real-time and embedded system. http://www.omgmarte.org/.

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93

06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399