Timed-pNets: a communication behavioural semantic model for distributed systems

Yanwen Chen, Yixiang Chen, Eric Madelaine

▶ **To cite this version:**

Yanwen Chen, Yixiang Chen, Eric Madelaine. Timed-pNets: a communication behavioural semantic model for distributed systems. Frontiers of Computer Science, 2014, 8, pp.24. 10.1007/s11704-014-4096-4 . hal-01086091

HAL Id: hal-01086091

https://inria.hal.science/hal-01086091

Submitted on 25 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Timed-pNets: a communication behavioural semantic model for distributed systems

Yanwen CHEN[1,2,3], **Yixiang CHEN** (✉)[1], **Eric MADELAINE**[2,3]

1 MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China

2 INRIA Sophia Antipolis Méditérannée, Sophia Antipolis 06902, France,

3 University of Nice Sophia Antipolis, CNRS, Sophia Antipolis 06900, Franc

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 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 behaviours 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. Non-leaf nodes (called timed-pNets nodes) are synchronisation devices that synchronize the behaviours of subnets (these subnets can be leaves or non-leaf nodes). Both timed-pLTSs and timed-pNets nodes 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). In the end the TimeSquare tool is used to simulate and check the validity of our model.

Keywords ITS, logical time, formal method, timed specification, synchronous and asynchronous communication

Received March 8, 2014; accepted July 2, 2014

E-mail: yxchen@sei.ecnu.edu.cn, alenechenyanwen@gmail.com

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 delays. Intelligent transportation system (ITS) is one typical application in this area. It consists of distributed vehicles which are equipped with their own independent clocks. Despite each vehicle has a common time base (e.g. local physical clock), there is no global physical clock shared by these vehicles. In such a context, it is impossible to build time constraints (e.g. action α from process A and action β from process B happens at the same time) since the processes 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. It has been proven that logical time brings benefits to several domains. It was first introduced by Lamport to represent the execution of distributed systems [1]. Then it was extended and used in distributed systems to check the communication and causality path correctness [2]. Logical time has also been intensively used in synchronous languages [3,4] 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 been proven to be adaptable to any level of description, from very flexible causal time descriptions to very precise scheduling descriptions [5]. Logical time can be multiform, a global partial order built from local total orders of clocks. Inspired by the CCSL model [6], we design clock relations to specify the systems logical time constraints. The logical time constraints in this paper do not rely on global physical clocks. We consider the constraints as dependency relations. For example, a coincidence relation in this paper forces two actions to occur simultaneously (e.g. they wait for the same data to be ready to trigger the execution of the two actions), which means that one action (e.g. "α") can occur if and only if another action (e.g. "β") occurs. Thus the term "simultaneous" (or "at the same time") in this paper does not mean the two actions really occur at the same timestamp (e.g. at 20:30 of the same global physical clock), but they must logically coincide (i.e. α occurs if and only if β occurs, but the timestamps of them may not be the same). So our model is a logical constrained model that is expressed by a set of logical clocks and clock constraints. In a 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 verifications and to physical constraints.

Our model is based on pNets (parametrized networks of synchronized automata) [7], an expressive and flexible semantic model for the modeling and verification of (untimed) distributed systems. The pNet model describes the behaviours of concurrent systems in terms of value-passing labelled transition systems (LTSs), and expresses communications and synchronisation with synchronisation vectors (originating in [8]). 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 [7]. The flexibility of the synchronisation vector mechanism naturally provides descriptions of heterogeneous systems, from point-to-point or multipoint synchronisation, to sophisticated asynchronous queuing policies. Parametrization and hierarchy also make pNet models compact, and close to the program structure, and as a consequence easy to generate in a compositional way [9]. All these advantages attracted us to choose it for modelling the systems. However, pNets have no mechanism to describe time constraints neither to explicitly define asynchronous communication behaviours. We propose a novel timed model called timed-pNets by introducing timed specifications into pNets. The timed specifications represent system logical clocks and their relations so that the system time-related behaviours 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 physical behaviour evolution and control. In this paper, we concentrate on abstract models, that are appropriate to reason about the communications, 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 though we do not build continuous time model for the physical world, we do not isolate our model from it. Our model is capable to monitor and control physical behaviours by taking samples from the physical world.

In our previous work [10] we proposed the first version of timed-pNets, including the notion of logical clocks directly imported from CCSL. A set of clock constraints related to the logical clocks were built to describe the system casual relations. We also represented a simple use-case inspired from ITS systems, and illustrated the simulation results with execution traces by the TimeSquare tool [11]. However, this model was not sufficient to build hierarchical timed specifications starting from timed-pLTSs.

In this paper, we enhance the compositional aspects of our specification methodology: a system is modelled as a hierarchy of timed-pNets as shown in Fig. 1, where leaves are timed-pLTSs (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 communications between unrelated events, that we model as channels.

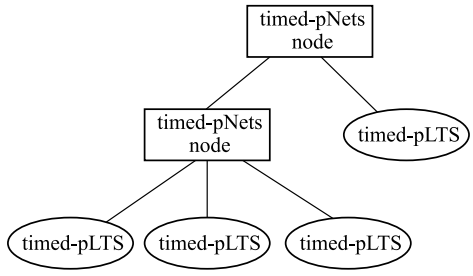

Fig. 1 Timed-pNets tree structure

From such a hierarchical model, we propose the proce-

dures as follows:

- At the bottom level, analyzing timed-pLTSs, and building the timed specifications (sets of clocks and clock relations) to specify the temporal behaviours,
- For each timed-pNet 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 designers, or computed from the model. The consequence is that the procedures above can be used arbitrarily in a bottom-up fashion, starting with detailed timed-pLTSs 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 [11] to simulate the possible executions of timed specifications.

The remainder of the paper is organized as follows. Section 2 represents the technical background including the definitions of pLTS and pNets. Section 3 describes the meaning of timed specifications including the formal definitions of timed-actions, logical clocks and their relations. Then we give the definition of timed-pLTSs in Section 4. In Section 5 we discuss how to build timed-pNets. The issue of checking the compatibility of timed-pNets is discussed in Section 6. The procedures of generating timed specifications from timed-pLTSs and timed-pNets are presented respectively in Sections 7 and 8. In Section 9 we discuss how to build multi-layers timed-pNets systems. Then in Section 10 we represent the simulations by using the TimeSquare tool. Related works are listed in Section 11. Finally, we summarize the current results and outline future work.

2 Technical background

The pNets model [7] is a formalism to represent the behaviours of distributed applications in terms of value-passing labelled transition systems (LTSs) [12]. The behaviours are modelled hierarchically by composition of classical LTSs [13] using a variant of synchronisation vectors [8]. To encode both families of processes and data value passing communications, LTSs and synchronisation vectors are enriched with parameters [14] that are used as communication arguments, in state definitions, and in synchronisation operators. This enables compact and generic description of parametrized and dynamic topologies. In the following we recall definitions of pLTS and pNets [7]. We start by giving the notion of parametrized actions that are basic elements for pLTSs.

Definition (parametrized actions) Let P be a set of names, $\mathcal{L}_{\mathcal{A},\mathcal{P}}$ a term algebra built over P, including at least a distinguished sort A for actions, and a constant action τ. We call $v \in P$ a parameter, and $a \in \mathcal{L}_{\mathcal{A},\mathcal{P}}$ a parametrized action, $\mathcal{B}_{\mathcal{A},\mathcal{P}}$ is the set of boolean expressions (guards) over$\mathcal{L}_{\mathcal{A},\mathcal{P}}$.

Definition (pLTS) A parametrized LTS is a tuple $\langle P, S, s_0, L, \rightarrow\rangle$ where:

- P is a finite set of parameters, from which we construct the term algebra $\mathcal{L}_{\mathcal{A},\mathcal{P}}$.
- S is a set of states; each state $s \in S$ is associated to a finite indexed set of free variables $fv(s) = \widetilde{x}_{J_s} \subseteq P$.
- $s_0 \in S$ is the initial state.
- L is the set of labels, $\rightarrow$ the transition relation $\rightarrow \subset S \times L \times L$.
- Labels have the form $l = \langle \alpha, e_b, \widetilde{x}_{J_s} := \widetilde{e}_{J_s}\rangle$ such that if $s \rightarrow s$, then:
  1) α is a parametrized action, expressing a combination of inputs $iv(\alpha) \subseteq P$ (defining new variables) and outputs $oe(\alpha)$ (using action expressions).
  2) $e_b \in \mathcal{B}_{\mathcal{A},\mathcal{P}}$ is the optional guard.
  3) The variables $\widetilde{x}_{J'_s}$ are assigned during the transition by the optional expressions $\widetilde{e}_{J'_s}$ with the constraints: $fv(oe(\alpha)) \subset iv(\alpha) \cup \widetilde{x}_{J_s}$ and $fv(e_b) \cup fv(\widetilde{e}_{J'_s}) \subseteq iv(\alpha) \cup \widetilde{x}_{J_s} \cup \widetilde{x}_{J'_s}$.

The networks of pLTSs (called pNets) are inspired by the synchronisation vectors of Arnold and Nivat [8], that is used to synchronise a (potentially infinite) number of processes. The holes in pNets can be indexed by a parameter, to represent (potentially unbounded) families of similar arguments. We represent the definition of pNets taken from [7] as below:

Definition (pNets) A pNet is a tuple $\langle P, pA_G, J, \widetilde{p}_J, \widetilde{O}_J, \overrightarrow{V}\rangle$ where: P is a set of parameters, $pA_G \subset \mathcal{L}_{\mathcal{A},\mathcal{P}}$ is its set of (parametrized) external actions, J is a finite set of holes, each hole j being associated with (at most) a parameter $p_j \in P$ and with a sort $O_j \subset \mathcal{L}_{\mathcal{A},\mathcal{P}}$. $\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronisation vectors of the form: $\overrightarrow{v} = \langle \alpha_g, \{\alpha_t\}_{i\in I, t\in B_i}\rangle$ such that: $I \subseteq J \wedge B_i \subseteq Dom(p_i) \wedge \alpha_i \in O_i \wedge fv(\alpha_i) \subset P$.

Each hole in the pNet has a parameter p_j , expressing that this "parametrized hole" corresponds to as many actual pro-

cesses as necessary in a given instantiation of its parameter. In other words, the parametrized holes express parametrized topologies of processes synchronised by a given Net. Each parametrized synchronisation vector in the pNet expresses a synchronisation between some instances ($\{t\}_{t\in B_i}$) of some of the pNet holes ($I \subseteq J$). The hole parameters being part of the variables of the action algebra can be used in communications and synchronisations between the processes.

3 Timed specification

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

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 three smart cars. These cars are equipped with sensors to detect the physical environment and parameters (e.g. such as cars' speeds, cars distances, etc.). And they communicate among each other to coordinate their movements and avoid collisions. Assume the three vehicles (*car*0, *car*1 and *car*2) are running on a road as Fig. 2.

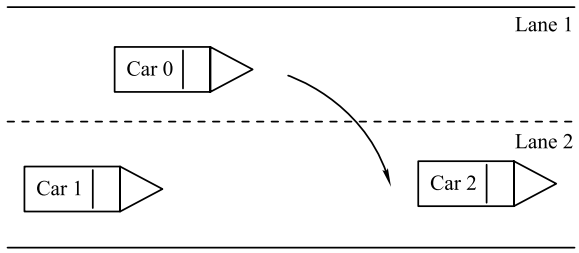

Fig. 2 Car insertion

The scenario of inserting *car*0 between *car*1 and *car*2 may follow the following steps: 1) *car*0 gets a change-lane request (e.g. from a human user); 2) *car*0 sends "notify" requests to *car*1 and *car*2 to ask for an agreement; 3) *car*1 (resp. *car*2) acknowledges *car*0 "yes" or "no"; 4) *car*0 collects the results from *car*1 and *car*2; 5) if both *car*1 and *car*2 answer "yes", *car*0 signals the consensus to *car*1 and *car*2 and then goes to step 5, otherwise *car*0 aborts the procedure; 6) *car*1 slows down and/or *car*2 speeds up to leave more space between them for *car*0; 7) *car*0 changes its direction and moves to lane 2; 8) *car*0 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 ($\mathcal{P}$ is a set of parameters that are used to build open expressions, typically expressing data variables) [7]. In our model, we define $\mathcal{L}_{\mathcal{A},\mathcal{P},\mathcal{T}}$ as the timed-action algebra, in which $\mathcal{T}$ is a set of (discrete) timed variables. We denote for example α ($\in \mathcal{L}_{\mathcal{A},\mathcal{P},\mathcal{T}}$) as an action name, then we consider α, $!\alpha(m)$ and $?\alpha(m)$ as timed-actions. α means that the timed-action executes locally but not delivers messages. $!\alpha(m)$ ($m \in \mathcal{P}$) is denoted as sending a message and $?\alpha(m)$ ($m \in \mathcal{P}$) as receiving a message.

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 [1]. 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:
$C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$ ($i \in \mathbb{N}$), 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 ith occurrence of the timed-action α is counted from the ith occurrence of the timed-action β as shown in the Fig. 4. The relation of coincidence (discussed in the next subsec-

tion) 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.

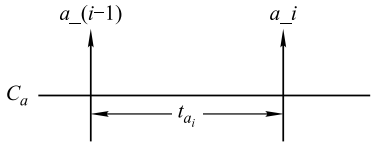

Fig. 3 Count the delay t_{α_i} when C_α is an independent clock

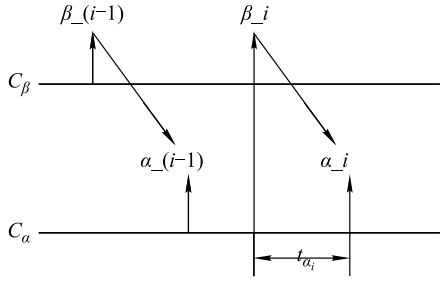

Fig. 4 Count the delay t_{α_i} when $C_\beta \prec C_\alpha$

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 ith occurrence of the clock C_α. The nth offset 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. $C_\alpha^{\{n\geqslant 8\}}$ filters out the first 8 occurrences.

More clearly, 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 $C_\alpha^{\{2n-1\}_{n\in\mathbb{N}}} = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_3)^{t_{\alpha_3}}\_2, \ldots \alpha(p_{(2n-1)})^{t_{\alpha_{(2n-1)}}}\_n, \ldots\}$. $C_\alpha^{\{n\geqslant 8\}} = \{\alpha(p_8)^{t_{\alpha_8}}\_1, \alpha(p_9)^{t_{\alpha_9}}\_2, \ldots\}$

Finally in this section 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, and the compatibility required for assembling subsystems together. In the following sections we shall describe procedures to compute the Timed Specifications of systems (timed-pLTSs and timed-pNets), and to check their compatibilities.

Definition 5 (timed specification) Let $\mathcal{I}_C$ be the set of occurrences of the clock C. A rimed specification is a pair $\langle \mathfrak{C}, \mathcal{R} \rangle$ where $\mathfrak{C}$ is a set of clocks, $\mathcal{R}$ is a set of clock relations on $\bigcup_{C\in\mathfrak{C}} \mathcal{I}_C$.

3.1 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 [6], 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 constraints between clocks.

- 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 other words, the two clocks are synchronized. Formally, $[\![C_\alpha = C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \equiv \beta\_i)$ (shown in Fig. 5(a)). This operator can naturally be used to describe synchronous communications.
- 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 other words, 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(b), the ith occurrence of the clock C_α always appears earlier than the ith occurrence of the clock C_β. The relation usually relates to the causalities induced by asynchronous communications.

3.2 Properties of the logical clock relations

Not surprisingly, these relations have their expected proper-

ties: coincidence is an equivalence relation, and precedence is a strict pre-order.

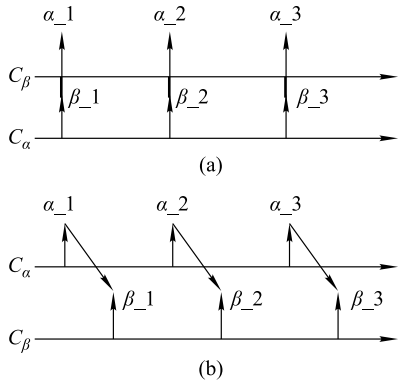

Fig. 5 Clock relations. (a) $[C_\alpha = C_\beta] = \forall i \in N, (\alpha_i \equiv \beta_i)$; (b) $[C_\alpha \prec C_\beta] = \forall i \in N, (\alpha_i \prec \beta_i)$

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 the ith ($i \in \mathbb{N}$) occurrence be α_i. The occurrence α coincides with itself ($C_\alpha = C_\alpha$). So we know 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 $\forall i \in \mathbb{N}, \beta\_i \equiv \alpha\_i$ ($C_\beta = C_\alpha$). Thus 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 $\forall i \in \mathbb{N}, \alpha\_i \equiv \gamma\_i$ ($C_\alpha = C_\gamma$). Thus 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, 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, we furthermore have the relation $\forall i, \beta\_i \prec \eta\_i$. According to the definition of precedence relations, 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 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: *car*0 sends a $notify(Ins)$ message to the other two cars, and waits for their answers. This phase is managed by the "CommIni" process, that communicates to the "ComRes" processes of other cars through asynchronous channels. In the model, there is one such channel for each type of messages, and for each pair of communicating processes; we use the parametrized structure of timed-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 a final decision to "Initial" component. If it is negative, then "Initial" aborts and signals *Cancel* to the user, otherwise the procedure goes to the next phase.

2) The execution phase: this phase is triggered and controlled directly by the "Initial" process. It sends $C_{!Consensus(ExpRes)^{t_o}}$ 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 a termination signal 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

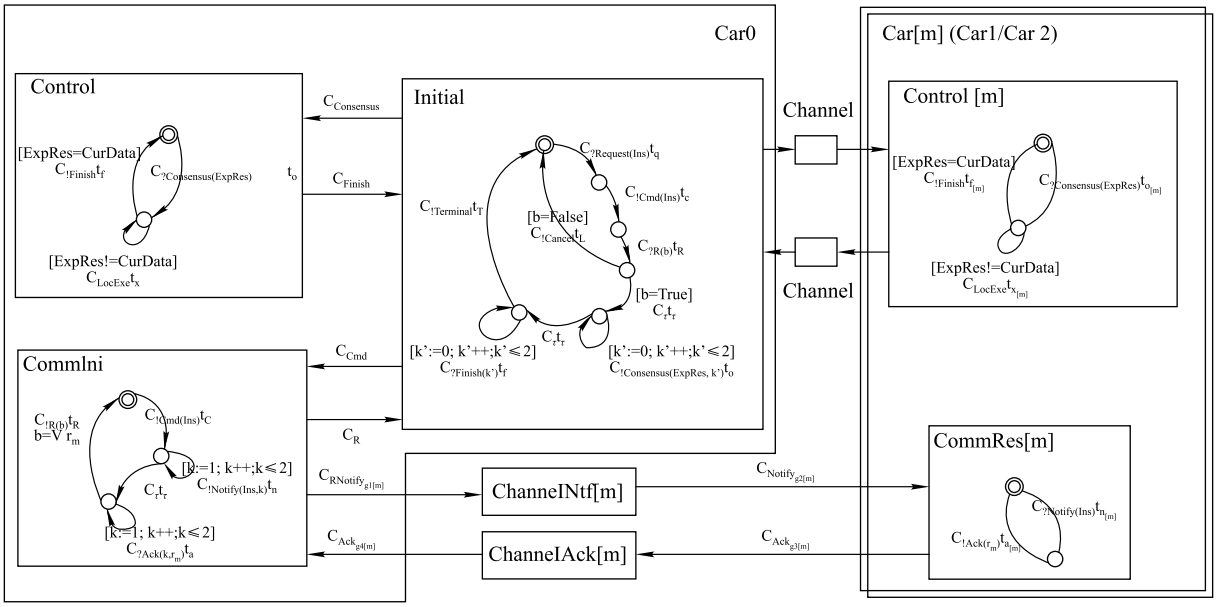

Fig. 6 Timed-pNets: communication behaviour model of cars insertion scenario

"$C_{?notify(Ins)^{t_n}}$" 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 processes 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, communications between two different cars are asynchronous (typically over some wireless ad-hoc network). Since we want to be able to take into account the communication time in our analysis, we insert a specific asynchronous channel (built as a special timed-pLTS) for each type of messages exchanged between cars.

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

4 Timed-pLTS 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 $\langle P, S, s_0, A, C, \rightarrow \rangle$, 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 transitions: $\rightarrow \subseteq S \times C \times S$. We write $s \xrightarrow{C_\alpha} s'$ for $(s, C_\alpha, s') \in \rightarrow$, in which $\alpha \in A$.

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 $k \in [1, 2, \ldots, N]$ (N is the (fixed) number of neighbors of the initiating car (here $N = 2$)). The first loop on s_1 means that $car0$ sends a notification signal to $car1$ and $car2$, separately. The second loop on s_2 means that $car0$ collects "ack" signals from $car1$ and $car2$. Moreover, we use the 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 clock relations here, they can be easily deduced from the figure.

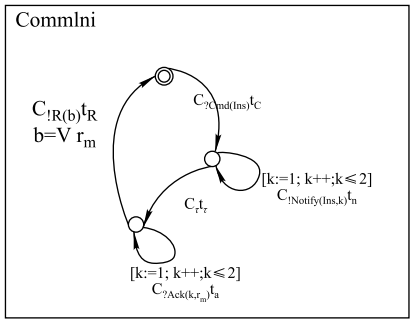

Fig. 7 The timed-pLTS of the CommIni component

Note that the system designers only need specify the timed-pLTSs. The clock relations can be automatically deduced from the timed-pLTSs (see Section 6).

4.1 Channels

We introduce channels to model asynchronous communication behaviours. A channel is defined as a special transition system with two timed-events: one for receiving messages, another for sending messages. A precedence relation is applied on the two events to model a delay of message transmissions. For simplification, the definition of channels 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 [15] but they are not the topics of this paper.

Definition 7 (channel)
A channel is a transition system with tuple $\langle P, S, A, C, \prec, \rightarrow \rangle$ in which

- P is a finite set of parameters.
- S is the 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 the set of clocks over timed-actions A.
- $\rightarrow$ is the set of 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.

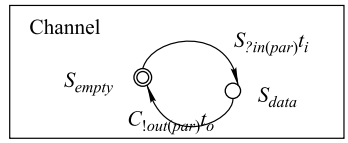

Fig. 8 The timed-pLTS of channel component

5 Timed-pNets semantic model

Here we define timed-pNets that are the nets of timed-pLTSs. Similar to the (untimed) pNets, a timed-pNet is a generalized composition operator, defining the synchronizations between a number of subsystems (holes). In timed-pNets, holes are characterized by timed-specifications. Building timed-pNets to represent a full system requires filling holes with (compatible) sub-nets.

Definition 8 (timed-pNets) A timed-pNet is a tuple $\langle P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} \rangle$, where:

- P is a finite set of parameters.
- A_G is a set of global timed-actions, and $\mathfrak{C}_G$ 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 $\langle \mathfrak{C}_j, \mathcal{R}_| \rangle$.
- $\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronization vectors of the form:

1) (binary communication between holes j_1 and j_2)
$\overrightarrow{v}^{1)} = \langle \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots \rangle \rightarrow C_g$,
in which $\{C_{!\alpha} = C_{?\alpha} = C_g\}, C_g \in \mathfrak{C}_G, C_{!\alpha} \in \mathfrak{C}_{j_1}, C_{?\alpha} \in \mathfrak{C}_{j_2}, j_1, j_2 \in J$;
2) or (visibility from hole j)
$\overrightarrow{v} = \langle \ldots, C_\alpha, \ldots \rangle \rightarrow C_g$, in which $\{C_\alpha = C_g\}, C_g \in \mathfrak{C}_G, C_{?\alpha} \in \mathfrak{C}_j, j \in J$.

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. $\overrightarrow{v_{i'}}$), $i, i' \in \mathbb{N}$)

Remark We define timed-pNets in a form inspired by the

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

synchronisation vectors of Arnold and Nivat [8], that we use to synchronise clocks from different processes. 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 the communication vector ($\langle \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots \rangle \rightarrow C_g$). 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 "→", and specifies the global clock generated by this synchronous vector. Another vector ($\langle \ldots, C_\alpha, \ldots \rangle \rightarrow C_g$) makes the local clock C_α visible by generating a global clock C_g. For both kinds of synchronous vectors, the local clocks (that appear between "⟨" and "⟩") are transparent to the upper layer nodes. Only the global clocks (the last elements in the synchronous vectors) can be observed from the upper level. These global clocks can be used for building a higher level timed-pNets node.

Moreover, from the Definition 8 we can see that the synchronous vectors only catch the coincidence relations between clocks (for describing synchronous communications), which makes our timed-pNets models cannot directly specify asynchronous communications. So when modelling asynchronous communications, we need introduce channels into systems. The two subsystems that asynchronously communicate with each other are connected by a channel in which a communication delay is modelled. Example 3 shows us how to take advantage of channels to specify asynchronous communications.

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 timed-pNet in which j_1, j_2, j are parametric holes with indexes k_1, k_2, k, with respective domains Dom_1, Dom_2, Dom, the synchronization vectors will look like:

1) 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 $Dom_1 \times Dom_2$. $\langle \ldots, C_{!\alpha[k_1]}, \ldots, C_{?\alpha[k_2]}, \ldots \rangle \rightarrow C_{g[m]}$, in which $\{C_{!\alpha[k_1]} = C_{?\alpha[k_2]} = C_{g[m]}\}$, $C_{g[m]} \in \mathfrak{C}_G, C_{!\alpha[k_1]} \in \mathfrak{C}_{j_1}, C_{?\alpha[k_2]} \in \mathfrak{C}_{j_2}$

2) Visibility. Each visible action from hole j generates a corresponding global action. $\langle \ldots, C_{\alpha[k]}, \ldots \rangle \rightarrow C_{g[k]}$, in which $\{C_{\alpha[k]} = C_{g[k]}\}, C_{\alpha[k]} \in \mathfrak{C}_j, C_{g[k]} \in \mathfrak{C}_G$.

Example 3 We take 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 those cars, and the channels by which they communicate. Hereafter, we show how to build the timed-pNets of 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, *car*0 sends a notification to *car*1 (resp. *car*2) via "ChannelNtf[1]" (resp."ChannelNtf[2]"), and then *car*1 (resp.*car*2) answers an "ack" to *car*0 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:

1) The vector[2] $\langle -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}} \rangle \rightarrow C_{ack_{g3_{[1]}}}$ represents the communications 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.

2) the vector $\langle C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify}, -, C_{c.?notify_{[1]}}, - \rangle \rightarrow C_{notify_{g1_{[1]}}}$ represents the communications between the components "*CommIni*" and "*ChannelNtf*[1]" and builds a global clock "$C_{notify_{g1_{[1]}}}$" (remember $C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify}$ is the clock built from the clock $C_{!notify}$ by choosing the occurrences with the odd indexes).

Following the timed-pNets definition, we can formalize this timed-pNets with details as:

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

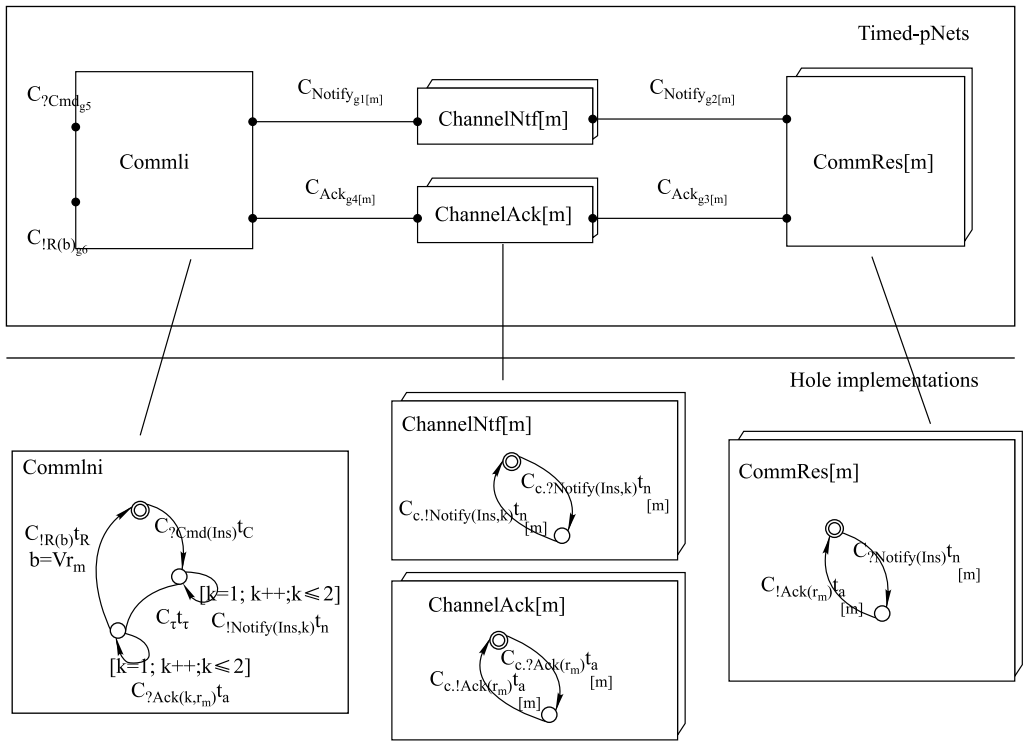

Fig. 9 A Timed-pNets with one of its implementations

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

Next we formalize the Timed Specifications of these holes as:

- For the hole "CommIni":

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In the end, we specify the synchronous vectors:

$\overrightarrow{V} = \{$

$V_1 : \langle C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify(Ins,k=1)}, -, C_{c.?notify(Ins)_{[1]}}, -\rangle \rightarrow C_{notify_{g1_{[1]}}}$,

$V_2 : \langle -, C_{?notify_{[1]}}, C_{c.!notify_{[1]}}, -\rangle \rightarrow C_{notify_{g2_{[1]}}}$,

$V_3 : \langle -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}}\rangle \rightarrow C_{ack_{g3_{[1]}}}$,

$V_4 : \langle C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack(k=1,r_m)}, -, -, C_{c.!ack(r_m)_{[1]}}\rangle \rightarrow C_{ack_{g4_{[1]}}}$

$V_5 : \langle C^{\{2s\}_{s\in\mathbb{N}}}_{!notify(Ins,k=2)}, -, C_{c.?notify(Ins)_{[2]}}, -\rangle \rightarrow C_{notify_{g1_{[2]}}}$,

$V_6 : \langle -, C_{?notify_{[2]}}, C_{c.!notify_{[2]}}, -\rangle \rightarrow C_{notify_{g2_{[2]}}}$,

$V_7 : \langle -, C_{!ack_{[2]}}, -, C_{c.?ack_{[2]}}\rangle \rightarrow C_{ack_{g3_{[2]}}}$,

$V_8 : \langle C^{\{2s\}_{s\in\mathbb{N}}}_{?ack(k=2,r_m)}, -, -, C_{c.!ack(r_m)_{[2]}}\rangle \rightarrow C_{ack_{g4_{[2]}}}$

$V_9 : \langle C_{?Cmd}, -, -, -\rangle \rightarrow C_{?Cmd_{g5}}$,

$V_{10} : \langle C_{!R}, -, -, -\rangle \rightarrow C_{!R_{g6}}\}$

Discussion Timed specification of holes. Let us now argue how the timed specifications of the holes in the upper-level

timed-pNet node may have been specified, in a top-down approach, before building their timed-pLTS implementations. This, intuitively, is done from the informal description of the scenario and the architecture knowledge of the top level components and communications:

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 ask for agreements;
3) the component "CommIni" collects results from $car1$ and $car2$ by clock $C_{?ack}$;
4) the component reports a 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 a 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 the "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:

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

In Section 6, we will show that the timed specifications of the holes in Fig. 9 are indeed fulfilled by the corresponding timed-pLTS of "CommIni", "ComRes", "ChannelNtf", and "ChannelAck".

6 Compatibility

When assembling timed-pNets, the architect has to ensure that the timed-pLTSs that will be plugged into holes indeed match the timed specifications of these holes. The ultimate goal is to provide a refinement-based approach: the time properties proved on an open (abstract) timed-pNet system will be preserved by the refinements of Timed Specifications. One of the basic approaches for building such refinements is to ensure the compatibility of a subsystem with the enclosing holes before composing the system. For example, 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 is 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 $TS = \langle \mathfrak{C}, \mathcal{R} \rangle$ be a timed specification with a set of clocks $\mathfrak{C}$ and a set of relations $\mathcal{R}$. The saturated relation set (denoted as $\mathcal{R}^+$) is the clock relation set $\mathcal{R}$ augmented by all relations possibly deduced from $\mathcal{R}$, by transitivity of precedence and reflexivity, symmetry, and transitivity of coincidence.

For example, if $\mathcal{R} = \{C_1 \prec C_2 \prec C_3\}$ $(C_1, C_2, C_3 \in \mathfrak{C})$, then according to the transitivity property of the relation $\prec$, we can get the saturated relation set $\mathcal{R}^+ = \{C_1 \prec C_2 \prec C_3, C_1 \prec C_3, C_1 = C_1, C_2 = C_2, \ldots\}$

Definition 10 (inclusion of time specifications) Given two timed specifications $TS_1 = \langle \mathfrak{C}_1, \mathcal{R}_1 \rangle$ and $TS_2 = \langle \mathfrak{C}_2, \mathcal{R}_2 \rangle$. Let $\mathcal{R}_1^+$ (resp. $\mathcal{R}_2^+$) be the saturated relation of TS_1 (resp. TS_2). We say TS_2 includes TS_1 (denoted as $TS_1 \ll TS_2$) if and only if $\mathfrak{C}_1 \subseteq \mathfrak{C}_2 \wedge \mathcal{R}_1 \subseteq \mathcal{R}_2^+$.

According to the definition, $TS_1 \ll TS_2$ means that the relation existing in the timed specification TS_1 must exist in

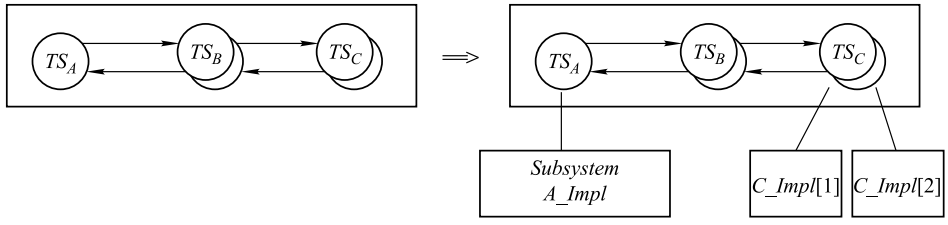

Fig. 10 Partial instantiation of a timed-pNets subsystem

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 $\mathcal{R}_2^+ = \{C_1 \prec C_2 \prec C_3, C_1 \prec C_3, C_1 = C_1, C_2 = C_2, \ldots\}$. 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 $TS_1 = \langle \mathfrak{C}_1, \mathcal{R}_1 \rangle$ and $TS_2 = \langle \mathfrak{C}_2, \mathcal{R}_2 \rangle$ are two timed specifications, then $TS_1 \ll TS_2 \implies \mathcal{R}_1^+ \subseteq \mathcal{R}_2^+$

Proof Taken any two relation r_1, $r_1' \in \mathcal{R}_1$. Let $r_1^+ \in \mathcal{R}_1^+$ be the relation deduced from the two relations r_1, r_1' in terms of the property P proposed in Section 3. Assume $r_1^+ \notin \mathcal{R}_2^+$. Since $TS_1 \ll TS_2$, from the definition of inclusion we know $\mathcal{R}_1 \subseteq \mathcal{R}_2^+$. Furthermore, we know $r_1, r_1' \in \mathcal{R}_2^+$. So in the set $\mathcal{R}_2^+$ we can get the relation r_1^+ by using the same property P. So we have $r_1^+ \in \mathcal{R}_2^+$ that is in contradict with our assumption. Therefore, we have $r_1^+ \in \mathcal{R}_2^+$. Moreover, because $r_1^+ \in \mathcal{R}_1^+$, we can get $\mathcal{R}_1^+ \subseteq \mathcal{R}_2^+$.

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 compatible 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 $TS_1' = \langle \mathfrak{C}_1', \mathcal{R}_1' \rangle$, $TS_2' = \langle \mathfrak{C}_2', \mathcal{R}_2' \rangle$ and $TS = \langle \mathfrak{C}, \mathcal{R} \rangle$. Let $\mathcal{R}_1'^+$ (resp. $\mathcal{R}_2'^+$, $\mathcal{R}^+$) 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 $\mathcal{R} \subseteq \mathcal{R}_1'^+$. Moreover, because we know that $TS_1' \ll TS_2'$, according to the Lemma 1, we have $\mathcal{R}_1'^+ \subseteq \mathcal{R}_2'^+$. According to the set theory, we know that $\mathcal{R} \subseteq \mathcal{R}_2'^+$. Finally, according to the Inclusion and refinement relation definition, we get $H\_Impl_2 \sqsubseteq H$.

7 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 the timed specifications from the timed-pLTSs. More precisely, given the action algebras and the transition relations of a timed-pLTS, we can get a 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 tell 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 represents 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.

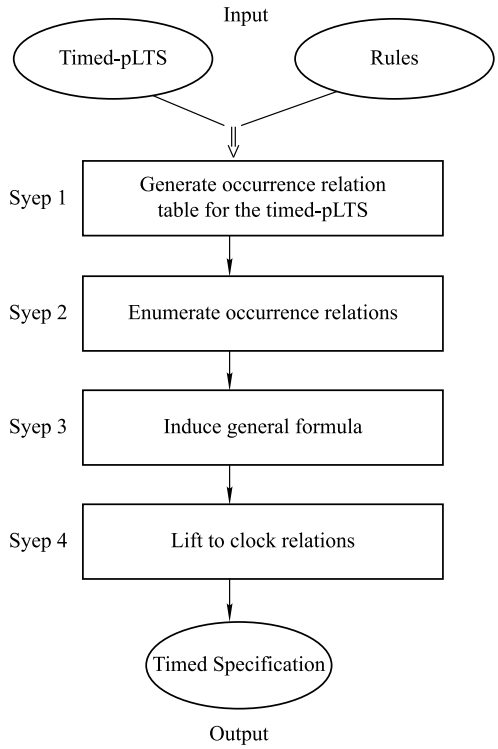

Fig. 11 Steps for generating the TS of a timed-pLTS

7.1 Auxiliary functions: Pre/Post sets

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

For a timed-pLTS transition system $\langle P, S, s_0, A, C, \rightarrow \rangle$, we denote $PreAct(s, s')$ as the set of direct preceding timed-

action occurrences of s from s', and $PostAct(s, s')$ as the set of direct succeeding timed-action occurrences of state s towards state s'. Then we denote $PreAct(s)$ (resp. $PostAct(s)$) as 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 the cases where branches in the LTS allow some executions to go several times through alternative transitions out of some states.

Definition 12 (preceding timed-action occurrences) Let $\langle P, S, s_0, A, C, \rightarrow\rangle$ be a timed-pLTS transition system. For $s \in S$ and $\alpha(p)^{t_\alpha} \in A, (p \in P)$, 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 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 rimed-action occurrences] Let $\langle P, S, s_0, A, C, \rightarrow\rangle$ be a timed-pLTS transition system. For $s \in S$ and $\alpha(p)^{t_\alpha} \in A, (p \in P)$, 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 $PostActIndex(s) = \sum_{s' \in S} PostActIndex(s, s')$.

7.2 Relations and assignment rules

The computation in Step 1 is based on a set of rules that identify specific configurations of the states in the timed-pLTS traversal. For each such configuration, we have 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 the guards for entering/leaving loops, in terms of a specific "loop counter".

We define a restrictive notion of looping state which is reasonable configuration 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. 12), so several loops will be executed the same number of times; 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 $PreAct(s_0) \notin \emptyset$, 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:
   a) go inside the loop
      $PreAct(s) \prec \alpha\_i$,
      [**Assign:** $i := i + 1$];
   b) stay in loop,
      $\alpha\_i \prec \alpha\_(i + 1)$;
   c) leave a loop:
      I) 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:** $j := j + 1$];
      II) to one post-action out of $PostAct(s, s_0)$:
         $\alpha\_i \prec PostAct(s) \backslash PostAct(s, s_0)$,
         [**Assign:**
         $PostActIndex(s) \Leftarrow PreActIndex(s)$];
      III) to one post-action in $PostAct(s, s_0)$:
         $\alpha\_i \prec PostAct(s, s_0)$,
         [**Assign:**
         $PostActIndex(s) \Leftarrow PreActIndex(s) + 1$].

Table 1 Time assignment for the timed-pLTS "Car.CommIni"

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	$tr5 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{!R}} s_0$	$\tau\_r \prec !R\_m$	$f_{tr5} : m = r$
	$tr6 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{?Ack}} s_2$	$\tau\_r \prec ?ack\_j$	$f_{tr6} : j := j + 1$
	$tr7 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{?Ack}} s_2$	$?Ack\_j \prec ?ack\_(j + 1)$	
	$tr8 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{!R}} s_0$	$?Ack\_j \prec !R\_m$	$f_{tr8} : m = r$

7.3 The method for generating timed specification

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

7.3.1 Step 1: generate occurrence relations table

The Algorithm 1 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 --C_1--> s_i --C_2--> s_2 do
    insert a row with State = s_i, Transition = s_1 --C_1--> s_i --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 ≠ 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), a), b) and III) rules
      else
        apply case 2), a), b) and II) rules
      end if
    else
      if s_i = s_0 then
        apply case (1), a), b), I) and III) rules
      else
        apply case (2), a), b), I) and II) rules
      end if
    end if
  end for
end for
```

Example 4 Let us take the "CommIni" component from Fig. 7 as an example. We first transform Fig. 7 into Fig. 12 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 apply the rules 2), a), b) and II). 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 the assignment $r = n$. When a transition enters the loop, according to the rule a), we have the relation $?Cmd\_n \prec !notify\_i$ and the assignment $i := i + 1$ ($i \in \mathbb{N}$). When a transition stays in the loop, according to the rule b), 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 II), we have the relation $!notify\_i \prec \tau\_r$ and the assignment $r = n$ ($r \in \mathbb{N}$).

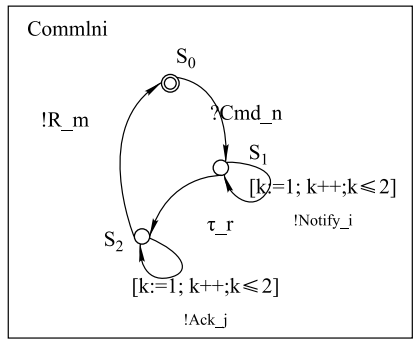

Fig. 12 Simplification of CommIni component

7.3.2 Step 2: enumerate occurrence relations

Now we go through the symbolic occurrence table built in

Step 1 and then generate 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 the 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 $k \leqslant kMax$ is not satisfied.

We present Algorithm 2 to enumerate these relations. The results of the algorithm are illustrated in the table in the Table 2 in which the rth column presents a set of occurrence relations in the round r, and the jth row 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 Table. **??**, we get $n = 1$ (because $m = 0$ and $f_{tr0} : n = m + 1$) and so we get the relation $!R\_0 < ?Cmd\_1$. 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 Table. **??**, we can compute $i = 1$ (because $f_{tr3} : i := i + 1$) and then we get the relation $?Cmd\_1 < !Notify\_1$. According to the loop control, we know k increases by 1 ($k + +$), so $k = 2$. Since the condition $k \leqslant 2$ is still satisfied, the transition goes into the self-loop again. According to the transition $tr3 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{!Notify}} s_1$,

Algorithm 2 Unfold occurrence relation table

Input: A symbolic occurrence table with a clock set C with n clocks. $C = \{C_1, C_2, \dots C_n\}$

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_α := anyone from PreAct(s)
set var C_β := 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 $r \leqslant N$ **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}
        $R[j][r] = \alpha\_Indexof(C_\alpha) < \beta\_Indexof(C_\beta)$
        $C \leftarrow C - C_\alpha - C_\beta$
        $j + +$
        $s' \leftarrow s$; $s \leftarrow s''$; $s'' \leftarrow$ 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 $C = \{C_1, C_2, \dots C_n\}$
end while

Table 2 Steps 2–4: Unfold rounds, generalize, and deduce clock relations

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

then we get the relation $!notify\_1 <!notify\_2$. Then k increases by 1 ($k++$), so at this time $k = 3$ that cannot satisfy the condition $k \leqslant 2$. 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 1, 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 Table 2. Remark that the rows corresponding to transitions $tr1$ and $tr5$ from the Table 1 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 the Table 2, etc., until we reach the column N.

7.3.3 Step 3: Generalize the occurrence relations

In the Table 2, in each line we get a sequence of occurrence relations. To induce the corresponding general relation, we transfer the problem to find a general formula for a sequence of nature numbers. We could use here standard arithmetic methods (e.g. Neville's algorithm [16]) that are able to automatically deduce polynomial formulas for a given set of unrepeatable natural number sequences. However, such a general approach would make difficult to estimate the minimum number of unfolding 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 the Table 2.

Example 6 Let us go on the Fig. 12 as an example. Since the loop counter is two, so we need unfold the relations at most for three times. As shown in the second line of the Table 2, the sequence of occurrence indexes of the clocks $C_{?Cmd}$ and $C_{!Notify}$ are $\{1, 2, 3\}$ and $\{1, 3, 5\}$. According to the Neville's algorithm, we can get the general formulas for the clock $C_{?Cmd}$ as $a_n = n$, and for the clock $C_{!Notify}$ as $a_n = 2n - 1$. So in the second line, the relation of the s round ($\forall s < 0$)is $?Cmd\_s <!Notify\_\{2s - 1\}$.

7.3.4 Step 4: 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 the Table 2.

8 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 ($\mathfrak{C}_G$) 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.

Definition 14 (global clock relation set) Given a timed-pNet $T\text{-}pNets = \langle P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} \rangle$ The global time specification of $T\text{-}pNets$ is the pair $\langle \mathfrak{C}_G, \mathcal{R}_G \rangle$, where $\mathcal{R}_G$ is the global clock relation set deduced from:
1) all local clocks relations $\mathcal{R}_j$ from its holes;
2) the (coincidence) relations deduced from all its synchronization vectors;
3) symmetry and transitivity of coincidence, transitivity of precedence.

During this logical saturation process, it may happen that contradictory relations are deduced, when two clocks would be proved both coincident and precedent, or precedent both ways. We call this clock conflicts.

Definition 15 (clock conflicts) Given a timed specification $\langle \mathfrak{C}, \mathcal{R} \rangle$:
1) two clocks C_α and C_β in $\mathfrak{C}$ are in conflict if either $C_\alpha = C_\beta \wedge (C_\alpha \prec C_\beta \vee C_\beta \prec C_\alpha) \in \mathcal{R}$ or $C_\alpha \prec C_\beta \wedge C_\beta \prec C_\alpha \in \mathcal{R}$
2) 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 us take Fig. 9 as an example. From the user specification in Example 3 (page 10), we know the clock relations of these holes are:

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

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

as:

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

Take the relation between the global clocks $C_{notify_{g1_{[1]}}}$ and $C_{notify_{g2_{[1]}}}$ as an example. It is generated by the synchronous vectors V_1 and V_2, as well as the relations of hole $ChannelNtf_{[1]}$. We deduce the relations $C_{notify_{g1_{[1]}}} =_{(\mathcal{R}_{V_1})} C_{c.?notify_{[1]}} \prec_{(\mathcal{R}'_{ChannelNtf_{[1]}})} C_{c.!notify_{[1]}} =_{(\mathcal{R}_{V_2})} C_{notify_{g2_{[1]}}}$, and conclude $C_{notify_{g1_{[1]}}} \prec C_{notify_{g2_{[1]}}}$.

The formal definition above is not very practical. Actually by analysing on the interactions between the synchronization vectors, we can compute a set of global clock relations that is sufficient to generate the global clock relation set. The Theorem 2 defines the case analysis procedure, and states its correctness (all relations computed are correct). The next Theorem 3 will prove its completeness. In Theorem 2, one particular case may detect a local conflicts between two global actions, more precisely between two synchronization vectors representing communications between the two holes. In this case, we shall signal the conflicts, but produce no relations between these actions. Other types of conflicts could be created by configurations involving more than two 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 $T\text{-}pNets = \langle P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V}\rangle$. Let $H_\alpha, H_\beta, H_\gamma$ be three holes of $T\text{-}pNets$ and $\mathfrak{C}_{H_\alpha}, \mathfrak{C}_{H_\beta}, \mathfrak{C}_{H_\gamma} \subset \widetilde{\mathfrak{C}}_J$ be the sets of clocks of holes H_α, H_β and H_γ. Let the clocks $C_{\alpha_1}, C_{\alpha_2} \in \mathfrak{C}_{H_\alpha}$, the clocks $C_{\beta_1}, C_{\beta_2} \in \mathfrak{C}_{H_\beta}$, the clock $C_{\gamma_1} \in \mathfrak{C}_{H_\gamma}$, with $\mathfrak{C}_{H_\alpha} \bigcap \mathfrak{C}_{H_\beta} \bigcap \mathfrak{C}_{H_\gamma} = \emptyset$). For each pair of global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$, we enumerate the pairs of synchronization vectors that are 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 $\mathcal{R}$:

Case 1 If the global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from synchronous vectors
$\langle \ldots, C_{\alpha_1}, \ldots, C_{\beta_1}, \ldots\rangle \rightarrow C_{a_{g1}}$ and
$\langle \ldots, C_{\alpha_2}, \ldots, C_{\beta_2}, \ldots\rangle \rightarrow C_{a_{g2}}$,
which are related to two holes C_{H_α} and C_{H_β} as shown in Fig. 13(a), then:

1) if $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$;
2) if $C_{\alpha_1} \prec C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$;
3) 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 a conflict is found.

Case 2 If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors
$\langle \ldots, C_{\beta_1}, \ldots, C_{\gamma_1}\rangle \rightarrow C_{a_{g1}}$ and
$\langle C_{\alpha_1}, C_{\beta_2}, \ldots, \ldots\rangle \rightarrow C_{a_{g2}}$, which are related to three holes C_{H_α}, C_{H_β} and C_{H_γ} as shown in Fig. 13(b), then:

1) if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$;
2) if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.

Case 3 If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors
$\langle \ldots, C_{\beta_1}, \ldots\rangle \rightarrow C_{a_{g1}}$ and
$\langle \ldots, C_{\beta_2}, \ldots, C_{\gamma_1}, \ldots\rangle \rightarrow C_{a_{g2}}$ as shown in Fig. 13(c) then:

1) if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$;
2) if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.

Case 4 If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors $\langle \ldots, C_{\beta_1}, \ldots\rangle \rightarrow C_{a_{g1}}$ and $\langle \ldots, C_{\beta_2}, \ldots, \ldots\rangle \rightarrow C_{a_{g2}}$ as shown in Fig. 13(d) then:

1) if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
2) if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.

Otherwise In other cases, this pair of clocks are **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.

Case 1 From the two synchronous vectors $\langle \ldots, C_{\alpha_1}, \ldots, C_{\beta_1}, \ldots\rangle \rightarrow C_{a_{g1}}$, $\langle \ldots, C_{\alpha_2}, \ldots, C_{\beta_2}, \ldots\rangle \rightarrow C_{a_{g2}}$,
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 $C_{a_{g1}} = C_{a_{g2}}$.

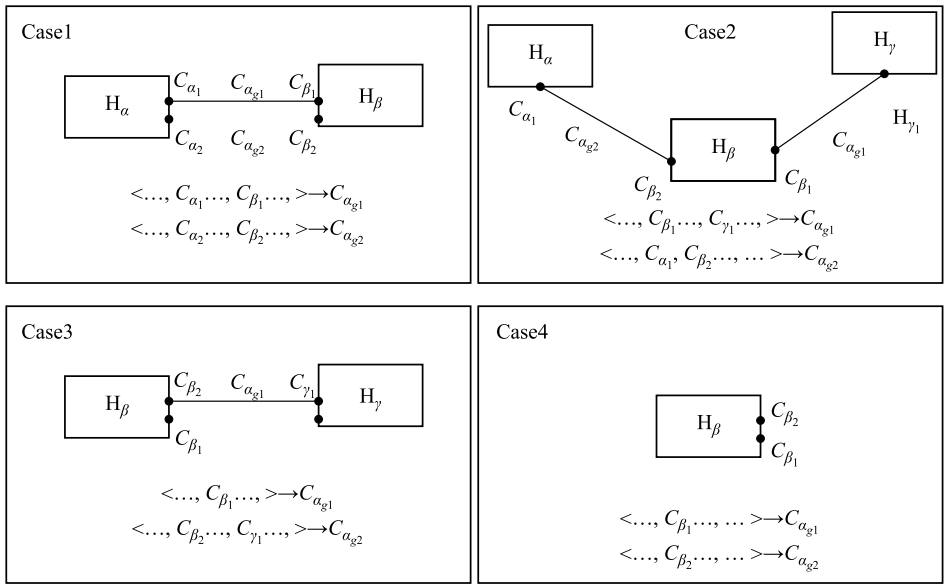

Fig. 13 The four cases of Theorem 2

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 the substitutivity of = w.r.t. $\prec$, we get the relation $C_{a_{g1}} \prec C_{a_{g2}}$.

Case 2 From the two synchronous vectors $\langle \ldots, C_{\beta_1}, \ldots, C_{\gamma_1} \rangle \rightarrow C_{a_{g1}}$ and $\langle C_{\alpha_1}, C_{\beta_2}, \ldots, \ldots \rangle \rightarrow C_{a_{g2}}$, 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 $C_{a_{g1}} \prec C_{a_{g2}}$.

Case 3 and Case 4 The proofs are similar to Case2.

Example 8 Let us take again the 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 clocks are generated by the vectors V_2: $\langle \ldots, C_{c.!notify_{[1]}}, \ldots, C_{?notify_{[1]}}, \ldots \rangle \rightarrow C_{notify_{g2_{[1]}}}$ and $V_3 : \langle \ldots, C_{!ack_{[1]}}, \ldots, C_{c.?ack_{[1]}}, \ldots \rangle \rightarrow C_{ack_{g3_{[1]}}}$. 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 three 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 timed-pNets $T\text{-}pNets = \langle P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} \rangle$.

1) Let $\langle \ldots, C_\alpha, \ldots, C_\beta \rangle \rightarrow C_{g1}$ and $\langle \ldots, C_\gamma, \ldots, C_\eta \rangle \rightarrow C_{g2}$ (C_α, C_β, C_γ, $C_\eta \in \widetilde{\mathfrak{C}}_J$) 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. The 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. The Case 2 of the Theorem 2 covers the situation. Furthermore, the four local clocks cannot be in four holes (or more than four holes), otherwise there is no local clock relations in $\widetilde{\mathcal{R}}_J$ can be used to deduce global clock relations, so no direct clock relation can be built between C_{g1} and C_{g2}.

2) Let $\langle \ldots, C_\alpha, \ldots, C_\beta \rangle \rightarrow C_{g1}$ and $\langle \ldots, C_\gamma, \ldots, \rangle \rightarrow C_{g2}$ (C_α, C_β, $C_\gamma \in \widetilde{\mathfrak{C}}_J$) 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 three holes or more. So the only possible combination is that C_γ and C_α (or C_β) are in the same hole. The rest

one is in another hole. The case 3 of the theorem 2 covers the situation.

3) Let $\langle \ldots, C_\alpha, \ldots, \rangle \rightarrow C_{g1}$ and $\langle \ldots, C_\gamma, \ldots, \rangle \rightarrow C_{g2}$ (C_α, $C_\gamma \in \widetilde{\mathfrak{C}_J}$) be two synchronous vectors to generate the global clocks C_{g1} and C_{g2}. The two local clocks cannot be in two different holes, otherwise there is no local relation can be find between them. So the only possible situation is that the two local clocks are in the same hole. The Case 4 of the Theorem 2 covers the situation.

In conclusion, if the relation of two global clock C_{g1}, $C_{g2} \in \mathfrak{C}_G$ can be deduced by the local clock relations from $\widetilde{\mathcal{R}_J}$, they must belong to one of the four cases of theorem 2.

9 Assembling a multi-layer 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. 14. In this structure, each layer is 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 ($\mathfrak{C}_G$) can be seen.

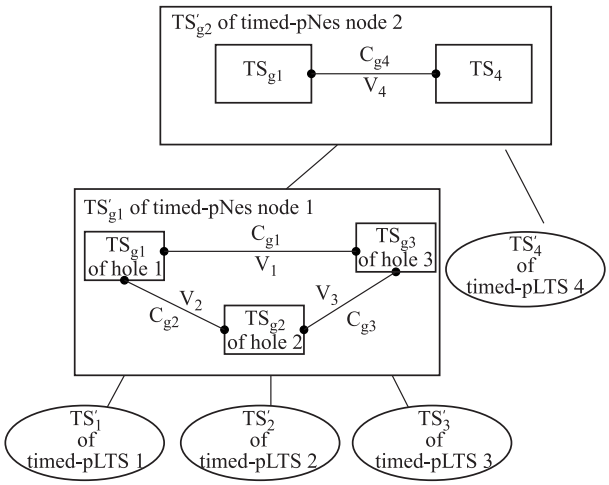

Fig. 14 Layered structure

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

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:

1) the relation set of the hole "CommIni" for open timed-pNets is $\mathcal{R}_{CommIni} = \{C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify}, C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify}, C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack}, C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}, C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}, C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_{!R} \prec C^{\Delta(1)}_{?cmd}\}$,
2) the relation set of the "CommIni" timed-pLTS component from the Table 2 is $\mathcal{R}'_{Commini} = \{C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_\tau \prec C_{!R} \prec C^{\Delta(1)}_{?cmd}\}$.

Since we can easily get $\mathcal{R}_{\{CommIni\}} \subseteq \mathcal{R}'_{Commini}$, 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 interactions between more than two holes of 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.

10 Simulation

In this section we explain how to use TimeSquare [11] to detect complex conflicts of timed-pNets. Two inputs are required by TimeSquare (see the Fig. 15). One is an open timed-pNets system. Another is a set of refined implementations. If the closed timed-pNets composed by those refined implementations have no conflicts, we say the closed timed-pNets are safe. Otherwise, the TimeSquare reports violations, which means that conflicts exist in the closed timed-pNets. Before running simulations, the two inputs are translated into timed specifications that are acceptable format for TimeSquare. The way of generating timed specifications is described in Sections 7 and 8.

10.1 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

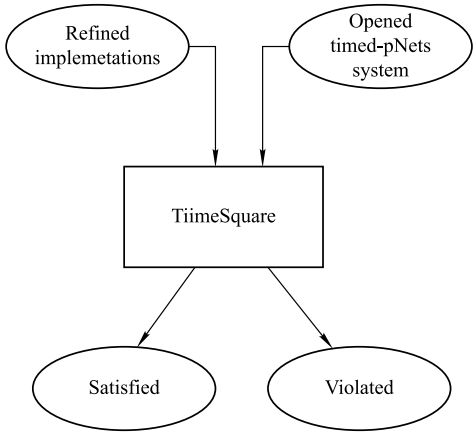

Fig. 15 Property checking by TimeSquare

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 8, we can get the abstract specification $TS_g = \langle \mathfrak{C}_g, \mathcal{R}_g \rangle$ with $\mathcal{R}_g = \{C_{?Cmd_{g5}} \prec C_{notify_{g1_{[m]}}} \prec C_{notify_{g2_{[m]}}} \prec C_{ack_{g3_{[m]}}} \prec C_{ack_{g4_{[m]}}} \prec C_{!R_{g6}}; C_{notify_{g1_{[1]}}} \prec C_{notify_{g1_{[2]}}}; C_{ack_{g4_{[1]}}} \prec C_{ack_{g4_{[2]}}}\}$. 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. 16 illustrates the concrete specification TS'_g. In this figure, each line demonstrate a clock and the black 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 $TS_g \ll TS'_g$.

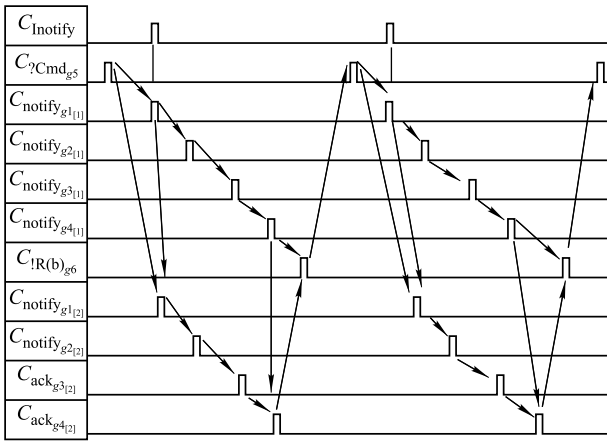

Fig. 16 System specification checking

10.2 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 $\mathcal{R}_{V_{new}} = \{C_{ExchangeInfo_{[1]}} = C_{ExchangeInfo_{[2]}} = C_{ExchangeInfo_{g11}}\}$. 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. 17.

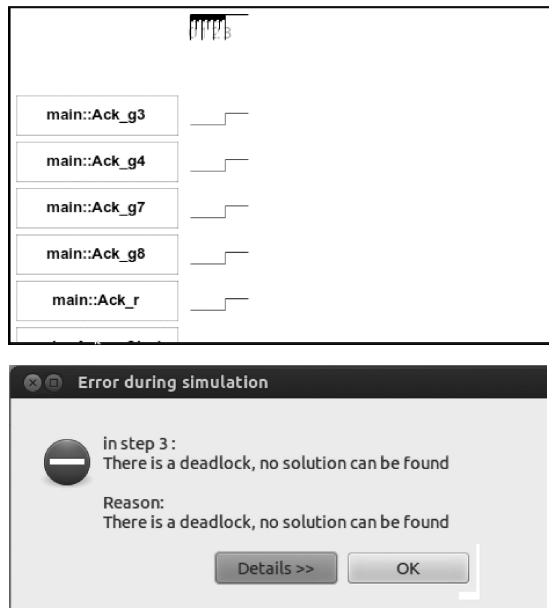

Fig. 17 Conflict Detected

- Analyzing the result: by analyzing our updated closed timed-pNets, we found out that the conflict is caused by a cycle represented in the Fig. 18. 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 theo-

rem 2 again, from the $TS'_{\{UpdatedCommIni\}}$ we can get the relation $\{C_{Ack_{g4_{[1]}}} \prec C_{Notify_{g1_{[2]}}}\}$ which is in contradict with the relation $\{C_{Notify_{g1_{[2]}}} \prec C_{Ack_{g4_{[1]}}}\}$. To fix the conflict, we need to find another implementation that is still compatible with these holes 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.

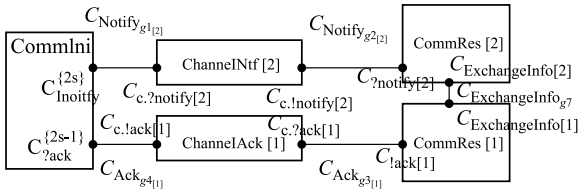

Fig. 18 System specification checking

11 Related work

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

Globally asynchronous locally synchronous (GALS) [17] 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 [18]. The idea of the GALS approach provides a methodology for combining concurrent embedded systems within loosely coupled systems. Similar to the idea of GALS, Serrano designed the HipHop language [19] that follows the synchronous reactive model of the Multiclock Esterel [20] 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 [21] is famous for modeling the behaviour 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 [22]. 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 [23] is a framework for the incremental composition of heterogeneous components. It allows building complex systems by coordinating the behaviour 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 behaviours 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 behaviours. 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) [24] is a 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 com-

plex. 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) [25] 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) [26] 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 behaviour 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 behaviour. 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 communication behaviour, 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 behaviour 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) [27], provides a location-triggered specification as well as operational semantics and denotational semantics [28] 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 behaviour 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.

12 Conclusion

This paper proposed a flexible time-related behavioural semantic model (called timed-pNets) for modeling communication behaviours of distributed systems. This model is able to build a hierarchical tree structure by composing the components of the systems. The refinement and compatibility of the timed-pNets are considered in the paper. A concrete example throughout the paper is given to represent how to build hierarchical specifications and how to refine them. In the end, we use the TimeSquare tool to check the compatibility of the refined system.

Three advantages are implied in our model: first, by introducing logical clock relations, timed-pNets are able to specify the system time-related communication behaviour constrains without relying on a 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, timed-pNets can model the heterogeneous systems including synchronous and asynchronous communications by taking advantage of channels. We believe that the timed-pNets are helpful for analyzing the

time-related behaviours of distributed systems including cyber physical systems.

However, since our timed specifications include only two relations (coincidence and precedence), they are not simple to specify some system behaviours like choice. This paper is the foundation for our model, and we are working on the next step, including logical compositions of timed specifications and time bound analysis. Besides, a natural question arises about the scalability and the efficacy of the proposed analysis approach on larger case studies. In our future work, we plan to apply the proposed analysis techniques on larger case studies and check the scalability. We also plan to write a tool to automatically generate timed specifications and check its performance.

Apart from the compatibilities, our model can also be used to check the logical correctness properties. But more interesting properties are time properties such as deadline property that expresses whether system communications can be successfully finished before a certain deadline. To check this property, 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 time properties. Furthermore, we plan to extend the current model timed-pNets to a new duration-pNets so that we can describe the system behaviours whose executions take time. It also looks interesting to translate our model to boolean automata to verify the system properties.

Acknowledgements This work was partially funded by the INRIA Associated Team DAESD between INRIA and ECNU; by NSFC (61321064 and 61370100); by Shanghai Knowledge Service Platform Project (ZF1213). And we give great thanks to Frederic Mallet and Julien Deantoni who took time to discuss with us and gave us bunches of advices. We are also indebted to the anonymous referees for their suggested improvements.

References

1. Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7): 558–565
2. Fidge C. Logical time in distributed computing systems. Computer, 1991, 24(8): 28–33
3. Berry G. The foundations of esterel. In: Proceedings of Proof, Language, and Interaction. 2000, 425–454
4. Benveniste A, Le Guernic P, Jacquemot C. Synchronous programming with events and relations: the signal language and its semantics. Science of computer programming, 1991, 16(2): 103–149
5. Boussinot F, De Simone R. The esterel language. Proceedings of the IEEE, 1991, 79(9): 1293–1304
6. André C. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Research Report RR-6925, INRIA, 2009 (in French)
7. Barros T, Boulifa R, Cansado A, Henrio L, Madelaine E. Behavioural models for distributed Fractal components. Annals of Telecommunications, 2009, 64(1–2): 25–43
8. Arnold A, Plaice J. Finite Transition Systems: Semantics of Communicating Systems. Prentice Hall International (UK) Ltd., 1994
9. Ameur-Boulifa R, Henrio L, Madelaine E, Savu A. Behavioural Semantics for Asynchronous Components. Research Report RR-8167, INRIA, 2012 (in French)
10. Chen Y, Chen Y, Madelaine E. Timed-pNets: a formal communication behavior model for real-time cps systems. In: Proceedings of Workshop on Trustworthy Cyber Physical Systems. 2012
11. Deantoni J, Mallet F. TimeSquare: Treat your models with logical time. In: Proceedings of the 50th International Conference on Objects, Models, Components, Patterns. 2012, 34–41
12. Milner R. Communicating and Mobile Systems: the π-Calculus. New York, NY, USA: Cambridge University Press, 1999
13. Milner R. Communication and Concurrency. Prentice-Hall, Inc., 1989
14. Cansado A, Madelaine E. Specification and verification for grid component-based applications: from models to tools. In: Proceedings of Formal Methods for Components and Objects. 2009, 180–203
15. Caromel D, Henrio L, Serpette B P. Asynchronous sequential processes. Information and Computation, 2008, 207(4): 459–495
16. Bulirsch R, Stoer J. Introduction to Numerical Analysis. Springer Heidelberg, 2002
17. Chapiro D M. Globally-asynchronous locally-synchronous systems. Dissertation for the Doctoral Degree. California: Stanford University. 1984
18. Chiodo M, Giusto P, Jurecska A, Hsieh H C, Sangiovanni-Vincentelli A, Lavagno L. Hardware-software codesign of embedded systems. IEEE Micro, 1994, 14(4): 26–36
19. Berry G, Nicolas C, Serrano M. 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. 2011, 49–56
20. Berry G, Sentovich E. Multiclock esterel. In: Proceedings of Correct Hardware Design and Verification Methods. 2001, 110–125
21. Alur R, Dill D L. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183–235
22. Bengtsson J, Larsen K G, Larsson F, Pettersson P, Yi W. Uppaal — a tool suite for automatic verification of real–time systems. In: Proceedings of Workshop on Verification and Control of Hybrid Systems III, LNCS 1066. 1995, 232–243
23. Basu A, Bozga M, Sifakis J. Modeling heterogeneous real-time components in BIP. In: Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods. 2006, 3–12
24. Graf S, Gérard S, Haugen Ø, Ober L, Selic B. Modeling and analysis of real-time and embedded system. Lecture Notes in Computer Science, 2006, 3844: 58–66
25. Eidson J, Lee E A, Matic S, Seshia S A, Zou J. Distributed real-time software for cyber-physical systems. Proceedings of the IEEE (special issue on CPS), 2012, 100(1): 45–59

26. Valero Ruiz V, Frutos Escrig d D, Cuartero Gomez F. On non-decidability of reachability for timed-arc petri nets. In: Proceedings of the 8th International Workshop on Petri Nets and Performance Models. 1999, 188–196
27. Chen Y. Stec: a location-triggered specification language for real-time systems. In: Proceedings of the ISORC Workshops. 2012, 1–6
28. Wu H, Chen Y, Zhang M. On denotational semantics of spatial-temporal consistency language–stec. In: Proceedings of the 2013 International Symposium on Theoretical Aspects of Software Engineering. 2013, 113–120

Yanwen Chen is a PhD student registed both in the East China Normal University and in the University de Nice Sophia Antipolis. She got her MS degree in both universities in the domain of computer science. Her research interests include distributed systems, cloud computing and formal methods. Now her research topic concerns the formal modeling and real-time scheduling of Cyber Physical Systems.

Yixiang Chen is a full professor for computer science and technology at Software Engineering Institute (SEI), East China Normal University (ECNU), where he is coordinating trustworthy software, Internet of things and cloud computing related research activities. Professor Chen is the director of the Engineering Research Center for Software/Hardware Co-design Technology and Application, MoE of China. He is Member of the Cloud Computing Experts Association of Chinese Institute of Electronics and Vice Chair of Shanghai Pudong IoT Alliance.

Eric Madelaine is a senior researcher at INRIA in Sophia-Antipolis, France. He has a diploma from Ecole Polytechnique, Paris, a PhD thesis from university of Paris 7, and an Habilitation from university of Nice. His research interests range from semantics of programming languages, distributed and cloud computing, component-based software, formal methods, methods and tools for specification and verification of complex programs. He has served in many international conferences committees, including Euromicro, FESCA, SEAA, FMCO, and FACS.