Local Planning Semantics: A Semantics for Distributed Real-Time Systems*

Mahieddine Dellabani
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
mahieddine.dellabani@univ-grenoble-alpes.fr

Jacques Combaz
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
saddek.bensalem@univ-grenoble-alpes.fr

Saddek Bensalem
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
jacques.combaz@univ-grenoble-alpes.fr

Marius Bozga
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
marius.bozga@univ-grenoble-alpes.fr

Abstract

Design, implementation and verification of distributed real-time systems are acknowledged to be very hard tasks. Such systems are prone to different kinds of delay, such as execution time of actions or communication delays implied by distributed platforms. The latter increase considerably the complexity of coordinating the parallel activities of running components. Scheduling such systems must cope with those delays by proposing execution strategies ensuring global consistency while satisfying the imposed timing constraints. In this paper, we investigate a formal model for such systems as compositions of timed automata subject to multiparty interactions, and propose a semantics aiming to overcome the communication delays problem through anticipating the execution of interactions. To be effective in a distributed context, scheduling an interaction should rely on (as much as possible) local information only, namely the state of its participating components. However, as shown in this paper these information is not always sufficient and does not guarantee a safe execution of the system as it may introduce deadlocks. Moreover, delays may also affect the satisfaction of timing constraints, which also corresponds to deadlocks in the former model. Thus, we also explore methods for analyzing such deadlock situations and for computing deadlock-free scheduling strategies when possible.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory
Keywords and Phrases Distributed Real-Time Systems, Timed Automata, Formal Verification
Digital Object Identifier 10.4230/LITES-v006-i001-a001
Received 2017-10-03 **Accepted** 2018-10-08 **Published** 2019-02-18

* This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #730080.

1 Introduction

Nowadays, real-time systems are ubiquitous in several application domains, and such an emergence led to an increasing need of performance: resources, availability, concurrency, etc. This expansion initiates a shift from the use of single processor based hardware platforms, to large sets of interconnected and distributed computing nodes. Moreover, it prompts the birth of a new family of systems known as *Networked Embedded Systems*, that are intrinsically distributed. Such an evolution stems from an increase in complexity of real-time software embedded on such platforms (e.g. electronic control in avionics and automotive domains [13]), and the need to integrate formerly isolated systems [24] so that they can cooperate as well as share resources improving thus functionality and reducing costs.

To deal with such complexity, the community of safety critical systems often restricts its scope to predictable systems, which are represented with domain specific models (e.g. periodic tasks, synchronous systems, time-deterministic systems) for which the range of possible executions is small enough to be easily analyzed, allowing the pre-computation of optimal control strategies. *Networked Embedded Systems* usually describe a set of real-time systems, distributed across several platforms, and interacting through a network. Because of their adaptive behavior, the standard practice when implementing such systems is not to rely on models for pre-computation of execution strategies but rather to design systems dynamically adapting at runtime to the actual context of execution. Such approaches, however, do not offer any formal guarantee of timeliness. Also, the lack of a priori knowledge on system behavior leaves also little room for static optimization.

Model-based development is one promising approach in building distributed real-time systems. First, an application model expressing a timed abstraction of the application behavior is built. This abstraction is platform independent, meaning that it does not consider any hardware specification such as communication delays or CPU(s) speed, which allows to: *(i)* model the system at early stages without any knowledge of the target platform, and *(ii)* verify the obtained model against some safety properties (functional requirements). Thereafter, the application source code, which represents the actual implementation of the system on a given platform, is automatically generated from the high level model. Then, the big challenge becomes how to verify the timing behavior of the implementation, since a lot of assumptions drop such as atomic execution of actions or timeless communication delays. In this paper, we propose a model-based approach aiming to mitigate the communication delays of distributed platforms. In this approach, systems consist of components represented as timed automata that may synchronize on particular actions to coordinate their activities. We contribute to this research field by proposing a different semantics than the usual semantics of timed automata. This semantics aims to distinguish between the decision dates for executing interactions and their actual execution dates by introducing a notion of scheduling on a semantics level. The idea behind this practice is to distinguish between the date at which interactions are executed and the date at which the execution decisions are effectively made. This will particularly help to anticipate the execution of interactions at least some delay beforehand, corresponding to the actual worst estimation of communication delays of a given platform, which will alleviate the effect of those delays on the system behavior.

This work is an extension of our work presented in [16]. We extend our previous work by *(1)* defining a more mature and realistic semantics for planning interactions. Especially, we introduce a lower bound horizon for planning interactions, that is, we impose at least a minimum delay, representing communication delays, between the effective planning of an interaction and its execution. In other words, immediate (timeless) planning is no longer allowed. Thereafter, we show that by enforcing a minimum delay between interactions planning and their executions, we may engender situational blocking situations that were nonexistent at first. We provide *(2)* a formal characterization of such blocking situations and *(3)* suggest an execution strategy aiming to avoid the latter. Furthermore, *(4)* if no execution strategy guarantying safe executions can be found, we propose an alternative method based on real-time controller synthesis approach as well as a discussion comparing both approaches.

The rest of the paper is organized as follows. Section 2 gives preliminary definitions of timed automata with respect to multiparty interactions as well as predicates definitions needed for the rest of the paper. In Section 3, we present our extended local planning semantics, discuss its relation with the usual timed automata semantics, and give sufficient conditions that formally characterize its deadlock states. Then, Section 4 presents an execution strategy aiming to enforce the correctness of the presented approach and find a deadlock free execution scenario when possible. Additionally, Section 5 explains how the local planning semantics can be formalized as a real-time controller synthesis problem and provide, thus, an alternative method for finding an execution strategy for such semantics. We also highlight the key issues met during our reflection and discuss important modeling points when using this technique. An implementation of the proposed is approach is described in Section 6 along with experimental results conducted on real-life case studies. Finally, the related works are presented in Section 7 and the conclusion given in Section 8.

2 Timed Systems and Properties

In the framework of the present paper, components are timed automata and systems are compositions of timed automata with respect to multiparty interactions. The timed automata we use are essentially the ones from [4], however, slightly adapted to embrace a uniform notation throughout the paper.

Given a set of clock $\mathcal{X}$, a clock constraint is an expression of the form:

$$c := \mathit{true} \mid x \sim ct \mid x - y \sim ct \mid c \wedge c \mid \mathit{false},$$

with $x, y \in \mathcal{X}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{Z}$. We denote by $\mathcal{C}(\mathcal{X})$ the set of clock constraints over $\mathcal{X}$.

▶ **Definition 1** (Component). A *component* is a tuple $B = (\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathtt{tpc})$ where $\mathcal{L}$ is a finite set of *locations*, $\ell_0 \in \mathcal{L}$ is an *initial* location, $\mathcal{A}$ a finite set of *actions*, $\mathcal{X}$ is a finite set of *clocks*, $\mathcal{T} \subseteq \mathcal{L} \times (\mathcal{A} \times \mathcal{C}(\mathcal{X}) \times 2^{\mathcal{X}}) \times \mathcal{L}$ is a set of *transitions* labeled with an action, a guard, and a set of clocks to be reset, and $\mathtt{tpc} : \mathcal{L} \to \mathcal{C}(\mathcal{X})$ assigns a *time progress condition* $\mathtt{tpc}(\ell)$ to each location $\ell \in \mathcal{L}$. Notice that time progress conditions are backward closed, that is, they are restricted to conjunctions of constraints of the form $x \leq ct$.

Throughout the paper, we assume components that are deterministic timed automata, that is, at a given location ℓ and for a given action a, there is at most one outgoing transition from ℓ labeled by a. Given a timed automaton $(\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathtt{tpc})$, we write $\ell \xrightarrow{a,g,r} \ell'$ if there exists a transition $\tau = \big(\ell, (a, g, r), \ell'\big) \in \mathcal{T}$. We also denote by $\mathit{guard}(a, \ell)$ the clock constraints of the transition labeled by a and outgoing from ℓ if it exists, and false otherwise and we write:

$$\mathit{guard}(a, \ell) = \begin{cases} g, & \text{if } \exists \tau = \big(\ell, (a, g, r), \ell'\big) \in \mathcal{T} \\ \mathit{false}, & \text{otherwise} \end{cases}$$

Before recalling the semantics of a component, we first fix some notations. Let $\mathcal{V}(\mathcal{X})$ be the set of all clock valuation functions $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and v_0 be the clock valuation assigning zero to all clocks. For a clock constraint c, $c(v)$ is a boolean value corresponding to the evaluation of c on v. For a valuation $v \in \mathcal{V}$ and for $\delta \in \mathbb{R}_{\geq 0}$, $v + \delta$ is the valuation satisfying $(v + \delta)(x) = v(x) + \delta$ for any x, while for a subset of clocks r, $v[r]$ is the valuation obtained from v by resetting clocks of r, i.e., $v[r](x) = 0$ for $x \in r$, $v[r](x) = v(x)$ otherwise. We denote by $c + \delta$ the clock constraint c shifted by δ, i.e. such that $(c + \delta)(v)$ iff $c(v + \delta)$. We also consider the classical *backward* and *forward* operators [30] on clock constraints, i.e. $(\swarrow c)(v)$ iff $\exists \delta \geq 0 \ . \ c(v + \delta)$ and $(\nearrow c)(v)$ iff $\exists \delta \geq 0 \ . \ c(v - \delta)$. In what follows, we also use two variants of the backward operator considering lower bounds $l \in \mathbb{Z}_{\geq 0}$ and upper bounds $u \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$: $(\swarrow_l c)(v)$ iff $\exists \delta \geq l \ . \ c(v + \delta)$ and $(\swarrow_l^u c)(v)$ iff $\exists \delta \ . \ l \leq \delta \leq u \wedge c(v + \delta)$.

▶ **Definition 2** (Semantics). A component $B = (\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathsf{tpc})$ defines the labeled transition system (LTS) $(\mathrm{Q}, q_0, \mathcal{A} \cup \mathbb{R}_{>0}, \rightarrow)$ where:

- $\mathrm{Q} = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ denotes the *states* of B, $q_0 = (\ell_0, v_0)$ is the initial state.
- $\rightarrow \subseteq \mathrm{Q} \times (\mathcal{A} \cup \mathbb{R}_{>0}) \times \mathrm{Q}$ denotes the set of transitions between states according to the rules:
  - $(\ell, v) \xrightarrow{a} (\ell', v[r])$ if $\ell \xrightarrow{a,g,r} \ell'$ and $g(v)$ and $\mathsf{tpc}(\ell')(v[r])$ (action step).
  - $(\ell, v) \xrightarrow{\delta} (\ell, v + \delta)$ if $\mathsf{tpc}(\ell)(v + \delta)$ for $\delta \in \mathbb{R}_{>0}$ (time step).

A *run* ϱ of B is an execution sequence that alternates action steps and time steps, that is:

$$\varrho = q_0 \sigma_0 q_1 \sigma_1 q_2 \ldots, \text{ such that } q_i \in \mathrm{Q}, \ q_i \xrightarrow{\sigma_i} q_{i+1}, \text{ and } i \in \mathbb{Z}_{>0}, \sigma_i \in \mathcal{A} \cup \mathbb{R}_{>0}.$$

We say that a state (ℓ, v) is *reachable* if there is an execution sequence from the initial configuration (ℓ_0, v_0) leading to (ℓ, v). In this paper, we always assume components with *well formed guards*, that is, transitions $\ell \xrightarrow{a,g,r} \ell'$ satisfy $g(v) \Rightarrow \mathsf{tpc}(\ell)(v) \wedge \mathsf{tpc}(\ell')(v[r])$ for any $v \in \mathcal{V}$. This ensures that the reachable states always satisfy the time progress conditions, i.e. if (ℓ, v) is reachable then we have $\mathsf{tpc}(\ell)(v)$. Consequently, the action step of Definition 2 can be simplified as:

$$(\ell, v) \xrightarrow{a} (\ell', v[r]) \text{ if } \ell \xrightarrow{a,g,r} \ell' \text{ and } g(v)$$

Notice that the set of reachable states is in general infinite, but it can be partitioned into a finite number of symbolic states [30, 7, 20]. A symbolic state is defined by a pair (ℓ, ζ) where, ℓ is a location of B, and ζ is a zone, i.e. a set of clock valuations defined by a clock constraint (as defined in Definition 1). Efficient algorithms for computing symbolic states and operations on zones are fully described in [7]. Given symbolic states $\{(\ell_j, \zeta_j)\}_{j \in J}$ of B, the predicate *Reach*(B) characterizing the reachable states can be expressed as:

$$\mathit{Reach}(B) = \bigvee_{j \in J} \mathsf{at}(\ell_j) \wedge \zeta_j,$$

where $\mathsf{at}(\ell_j)$ is true on states whose location is ℓ_j, and clock constraint ζ_j is straightforwardly applied to clock valuation functions of states.

We define the predicate *Enabled*(a) characterizing states (ℓ, v) at which an action a is enabled, i.e. such that $(\ell, v) \xrightarrow{a} (\ell', v')$ for some (ℓ', v'). It can be formally written as:

$$\mathit{Enabled}(a) = \bigvee_{\ell \in \mathcal{L}} \mathsf{at}(\ell) \wedge \mathit{guard}(a, \ell).$$

A state (ℓ, v) is said *urgent* if time cannot progress from (ℓ, v), that is, there is no $\delta \in \mathbb{R}_{>0}$ such that $(\ell, v) \xrightarrow{\delta} (\ell, v')$. Urgent states are characterized by the predicate:

$$Urgent(B) = \bigvee_{\ell \in \mathcal{L}} \mathsf{at}(\ell) \wedge \mathit{urg}(\ell) \tag{1}$$

where $\mathit{urg}(\ell)$ is a clock constraint characterizing the valuations from which time cannot progress with respect to the time progress condition of ℓ, that is, it is defined by $\mathit{urg}(\ell) = \bigvee_{i=1}^{m} (x_i \geq ct_i)$ if

$\mathsf{tpc}(\ell) = \bigwedge_{i=1}^{m} x_i \leq ct_i$. Notice that due to well-formed guards, an urgent reachable state satisfies also (1) if inequalities $x_i \geq ct_i$ on clocks are replaced by equalities $x_i = ct_i$ in the expression of $urg(\ell)$.

Following [31], we require that components or systems execute forever. This is referred to as *the requirement of progress* which can be divided split into discrete and time progress captured respectively by the notion of deadlocks and timelocks.

▶ **Definition 3** (Deadlock and action-time-lock)**.** We say that a state (ℓ, v) of a component B is *deadlock* if, with respect to its semantics, no action can be executed from (ℓ, v) and from its successors, that is:

$$Deadlock(B) = \neg\Big(\exists a \in \mathcal{A}\ .\ (\ell, v) \xrightarrow{a} (\ell', v')\ \vee \exists \delta > 0\ .\ (\ell, v) \xrightarrow{\delta} (\ell, v+\delta) \xrightarrow{a} (\ell', v')\Big).$$

Deadlock states are characterized by the following predicate:

$$Deadlock(B) = \neg\Big(\bigvee_{a \in \mathcal{A}} \swarrow \big(Enabled(a) \wedge \mathsf{tpc}(\ell)\big)\Big).$$

Because of well-formed guards this could be simplified into:

$$Deadlock(B) = \bigwedge_{a \in \mathcal{A}} \neg\big(\swarrow Enabled(a)\big).$$

A deadlock (ℓ, v) is called an *action-time-lock* when no interaction can execute nor time can progress from (ℓ, v), that is:

$$ActionTimeLock(B) = \neg\Big(\exists a \in \mathcal{A}\ .\ (\ell, v) \xrightarrow{a} (\ell', v')\ \vee\ \exists \delta > 0\ .\ (\ell, v) \xrightarrow{\delta} (\ell, v+\delta)\Big).$$

Action-time-lock states are characterized by the following predicate:

$$ActionTimeLock(B) = \Big(\bigwedge_{a \in \mathcal{A}} \neg Enabled(a)\Big)\ \wedge\ \Big(\bigvee_{\ell \in \mathcal{L}} \mathsf{at}(\ell) \wedge urg(\ell)\Big).$$

Deadlocks are situations from which a component is stuck at a given location without being able to progress by executing an action, which must be avoided in reactive systems. Action-time-locks are modeling errors and consist in deadlocks from which time cannot progress.

We denote by $\mathsf{time}(\varrho, i)$ the total elapsed time until point i, that is, $\sum_{j<i} \sigma_j$ such as $\sigma_j \in \mathbb{R}_{>0}$. In the same way, $\mathsf{time}(\varrho)$ represents the total elapsed time during ϱ, and is defined to be the limit of $\mathsf{time}(\varrho, i)$ if the sequence converges and ∞ otherwise.

▶ **Definition 4** (Zeno runs and Timelocks)**.** Let ϱ be an infinite run such that $\mathsf{time}(\varrho) \neq \infty$. Such a run violates the time progress requirements (only a finite number of events can occur in a finite amount of time), and is called *zeno*. Given a component B, a state of B is *timelock* if all infinite runs starting form that state are zeno.

In [31], it was shown that the class of timed automata in which 1 time unit is elapsed in every structural loop, also known as *strongly non-zeno*, is non zeno. An interesting property of this class is that it preserves non-zenoness under composability. Thus, checking the requirements of progress of a given system will boil down to checking its deadlock freedom.

In our framework, components communicate by means of *multiparty interactions.* A multiparty interaction is a rendez-vous synchronization between actions of a fixed subset of components. It takes place only if all the participants agree to execute the corresponding actions. Given n

components B_i, $i = 1, \ldots, n$, with disjoint sets of actions $\mathcal{A}_i$, an interaction is a subset of actions $\alpha \subseteq \cup_{1 \le i \le n} \mathcal{A}_i$ containing at most one action per component, i.e. $\alpha \cap \mathcal{A}_i$ is either empty or a singleton $\{a_i\}$. That is, an interaction α can be put in the form $\{a_i\}_{i \in I}$ with $I \subseteq \{1, \ldots, n\}$ and $a_i \in \mathcal{A}_i$ for all $i \in I$. We denote by $\text{part}(\alpha)$, the set of components *participating* in α, that is, $\text{part}(\alpha) = \{B_i\}_{i \in I}$. We say that two interactions α and β are conflicting, denoted by $\alpha \# \beta$, $\text{part}(\alpha) \cap \text{part}(\beta) \neq \emptyset$.

The semantics of a composition is interpreted at runtime by evaluating enabled interactions based on current states of the system components. In a composition of n components $B_{i \in \{1, \ldots, n\}}$ synchronizing through the interaction set γ, denoted by $\gamma(B_1, \ldots, B_n)$, an action a_i can execute only as part of an interaction α such that $a_i \in \alpha$, that is, along with the execution of all other actions $a_j \in \alpha$, which corresponds to the usual notion of multiparty interaction.

▶ **Definition 5** (Standard Semantics of a Composition). Given a set of components $\{B_1, \ldots, B_n\}$ and an interaction set γ. The (standard) semantics of the composition $S = \gamma(B_1, \ldots, B_n)$ w.r.t the set of interactions γ, is the LTS $(\mathrm{Q}_g, q_0, \gamma \cup \mathbb{R}_{>0}, \rightarrow_\gamma)$ where:

- $\mathrm{Q}_g = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ is the set of global states, where $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ and $\mathcal{X} = \bigcup_{i=1}^{n} \mathcal{X}_i$. We write a state $q = (\ell, v)$ where $\ell = (\ell_1, \ldots, \ell_n) \in \mathcal{L}$ is a global location and $v \in \mathcal{V}(\mathcal{X})$ is a global clock valuation. $q_0 = (\ell_0, v_0)$ is the initial state, where $\ell_0 = (\ell_0^1, \ldots, \ell_0^n)$ and v_0 is the clock valuation assigning 0 to all clocks.
- $\rightarrow_\gamma \subseteq \mathrm{Q} \times (\gamma \cup \mathbb{R}_{>0}) \times \mathrm{Q}$ is the set of labeled transitions defined by the rules:
  - $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$ for $\alpha = \{a_i\}_{i \in I} \in \gamma$, if $\forall i \in I\ (\ell_i, v_i) \xrightarrow{a_i} (\ell'_i, v'_i)$ and $\forall i \notin I\ (\ell_i, v_i) = (\ell'_i, v'_i)$.
  - $(\ell, v) \xrightarrow{\delta}_\gamma (\ell, v + \delta)$ for $\delta \in \mathbb{R}_{>0}$ if $\forall i \in \{1, \ldots, n\}\ \mathsf{tpc}_i(\ell_i)(v_i + \delta)$ where v_i denotes the restriction of v to clocks $\mathcal{X}_i$ of B_i.

To simplify notations, predicates defined on individual components B_i are straightforwardly interpreted on states (ℓ, v) of a composition $S = \gamma(B_1, \ldots, B_n)$ by considering the projection (ℓ_i, v_i) of (ℓ, v) on B_i, which is such that $\ell = (\ell_1, \ldots, \ell_n)$ and v_i is restriction of v to clocks $\mathcal{X}_i$ of B_i. For instance, $\mathsf{at}(\ell_i)$ evaluates to true on (ℓ, v) iff $\ell \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_{i-1} \times \{\ell_i\} \times \mathcal{L}_{i+1} \times \ldots \times \mathcal{L}_n$. Similarly, clock constraints of components B_i are applied to clock valuation functions v of the composition by restricting v to clocks $\mathcal{X}_i$ of B_i. This allows to write the predicate *Enabled*(α) characterizing states (ℓ, v) from which an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$ can be executed, i.e., such that $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$, as:

$$\begin{aligned} \mathit{Enabled}(\alpha) &= \bigwedge_{i \in I} \mathit{Enabled}(a_i) \\ &= \bigwedge_{i \in I} \bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \wedge \mathit{guard}(a_i, \ell_i) \\ &= \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \ldots, \ell_n)}} \mathsf{at}(\ell) \wedge \bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \mathit{guard}(a_i, \ell_i). \end{aligned}$$

Notice that the above formulation of *Enabled*(α) corresponds to locations enumeration of all components participating in interaction α. In practice, we rather consider only a subset of locations $\mathcal{L}_\alpha \subseteq \mathcal{L}$, from which the execution of α is possible. This corresponds to $\prod_{i \in I} |\mathcal{L}_{a_i}|$ possible configuration, where $\mathcal{L}_{a_i} \subseteq \mathcal{L}_i$ is a subset of locations from which there exists a transition labeled by action $a_i \in \alpha$, and $|\mathcal{L}_{a_i}|$ denotes the cardinality of $\mathcal{L}_{a_i}$, which is reasonably small in pratical examples but can be (at the worst case) equal to $\prod_{i \in I} |\mathcal{L}|$.

The definitions of run, reachable states, deadlocks and action-time-locks of components are also trivially extended to composition of components. Deadlocks of a composition $S = \gamma(B_1, \ldots, B_n)$

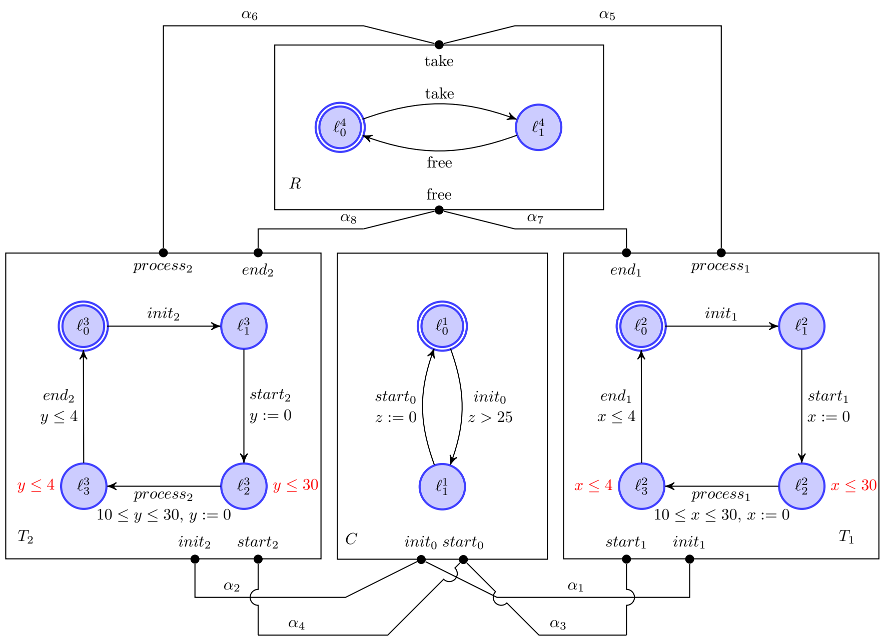


Figure 1 Task Manager.

can be characterized as follows:

$$Deadlock(S) = \bigvee_{\ell=(\ell_1,\ldots\ell_n)\in\mathcal{L}} \mathsf{at}(\ell) \wedge \left[\bigwedge_{\alpha\in\gamma} \neg \swarrow \left(Enabled(\alpha) \wedge \bigwedge_{1\leq i\leq n} \mathsf{tpc}_i(\ell_i)\right)\right], \tag{2}$$

and action-time-locks by:

$$ActionTimeLock(S) = \left(\bigwedge_{\alpha\in\gamma} \neg Enabled(\alpha)\right) \wedge \left(\bigvee_{1\leq i\leq n} \bigvee_{\ell_i\in\mathcal{L}_i} \mathsf{at}(\ell_i) \wedge urg(\ell_i)\right).$$

▶ **Example 6** (Running Example). Let us consider as a running example the composition of four components C, T_1, T_2, and R of Figure 1. Component C represents a controller that initializes then releases tasks T_1 and T_2. Tasks use the shared resource R during their execution. To implement such behavior, we consider the following interactions between C, R, and T_1: $\alpha_1 = \{init_0, init_1\}$, $\alpha_3 = \{start_0, start_1\}$, $\alpha_5 = \{take, process_1\}$, $\alpha_7 = \{free, end_1\}$, and similar interactions α_2, α_4, α_6, α_8 for task T_2, as shown by connections on Figure 1. The controller is responsible for firing the execution of each task. First, it non-deterministically initializes one of the two tasks, i.e. executes α_1 or α_2, and then releases it through interaction α_3 or α_4. Tasks perform their processing independently of the controller, after being granted an access to the shared resource (α_5 or α_6). When finished, a task releases the resource (interactions α_7 or α_8) and goes back to its initial location. An example of execution sequence of the system of Figure 1 is given below, in which valuations v of clocks x, y, and z are represented as a tuples $(v(x), v(y), v(z))$:

$$((\ell_0^1, \ell_0^2, \ell_0^3, \ell_0^4), (0,0,0)) \xrightarrow{26}_\gamma ((\ell_0^1, \ell_0^2, \ell_0^3, \ell_0^4), (26,26,26)) \xrightarrow{\alpha_1}_\gamma ((\ell_1^1, \ell_1^2, \ell_0^3, \ell_0^4), (26,26,26)) \xrightarrow{\alpha_3}_\gamma$$
$$((\ell_0^1, \ell_2^2, \ell_0^3, \ell_0^4), (0,26,0)) \xrightarrow{10}_\gamma ((\ell_0^1, \ell_2^2, \ell_0^3, \ell_0^4), (10,36,10)) \xrightarrow{\alpha_5}_\gamma ((\ell_0^1, \ell_3^2, \ell_0^3, \ell_1^4), (0,36,10)) \xrightarrow{2}_\gamma$$
$$((\ell_0^1, \ell_3^2, \ell_0^3, \ell_1^4), (2,38,12)) \xrightarrow{\alpha_2}_\gamma ((\ell_1^1, \ell_3^2, \ell_1^3, \ell_1^4), (2,38,12))$$

3 Local Planning of Interactions

In the previous Section, we presented a timed automata model for representing timed systems with multiparty interactions. The semantics of such model is based on the notion of *global states*, that is, interactions executions are based not only on the state of participating components but on the states of all components of the system. Conversely, a distributed system can be seen as a collection of loosely coupled components that communicate with a scheduling layer [23, 29] which, based on a partial view of the system, is responsible of taking decisions for interactions executions and their effective execution dates. Additionally, high-level coordination primitives, such as multiparty synchronizations (interactions) are rarely built-in primitives of distributed platforms. Hence, their implementation on a distributed platform requires synchronization protocols responsible for realizing synchronizations using simpler primitives such as point-to-point messages passing as explained in [29]. This is classically implemented using one or more additional coordination component(s) observing the system state and deciding on interactions execution, which adds on a communication overhead not reflected by the semantics of Section 2.

This motivates the introduction of the *local planning semantics*. This semantics differs from the standard semantics of timed automata in two main aspects: *(i)* interactions execution is based only on partial state of the system, that is, based only on the state of components participating in the considered interaction. Thus, it allows to decide locally without monitoring the entire system. *(ii)* it distinguishes between the execution decision of an interaction (its *planning*), and the execution itself. This distinction allows us to impose a delay between the planning of an interaction and its execution. The latter is constrained by the (maximal) communication latency induced by the execution platform, which is a parameter of the semantics. It is correct in the sense that it refines (it is included in) the semantics of Section 2. However, being based on local states, planning decisions are too permissive and may introduce deadlocks when they are not compatible with the global state of the system.

3.1 Definition of the LPS

Let $S = \gamma(B_1, \cdots, B_n)$ be a composition of components $B_1, \ldots, B_n$ with disjoint set of locations, actions and clocks. We define the predicate $\mathit{Plannable}(\alpha, \delta)$ characterizing states (ℓ, v) from which an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$ is enabled in $\delta \in \mathbb{R}_{\geq 0}$ units of time (if time progresses by δ units of time), that is, such that $\mathit{Enabled}(\alpha)$ evaluates to true on state $(\ell, v + \delta)$. It is characterized by:

$$\mathit{Plannable}(\alpha, \delta) = \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \cdots, \ell_n)}} \mathsf{at}(\ell) \wedge \bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \big(\mathit{guard}(a_i, \ell_i) + \delta\big) \tag{3}$$

Notice that for an interaction α the predicate $\mathit{Plannable}(\alpha, \delta)$ depends only on states of components of $\mathrm{part}(\alpha)$, which motivates the following property.

▶ **Property 7.** *Let (ℓ, v) be a state of the composition S. For any interactions $\alpha, \beta \in \gamma$ such that, $(\ell, v) \xrightarrow{\beta}_{\gamma} (\ell', v')$ and $\mathrm{part}(\alpha) \cap \mathrm{part}(\beta) = \emptyset$, if $\mathit{Plannable}(\alpha, \delta)$ holds at state (ℓ, v) then it still holds at state (ℓ', v').*

This property derives directly from the fact that executing an interaction β does not change the states of components participating in an interaction α, provided that α and β have disjoint sets of participating components, and thus, $\mathit{Plannable}(\alpha, \delta)$ is not affected by the execution of β in this case. In the following, we say that two interactions α and β *conflicts* when they have common participating components, that is, when $\mathrm{part}(\alpha) \cap \mathrm{part}(\beta) \neq \emptyset$, and we write $\alpha \# \beta$. We denote by $\mathit{conf}(\alpha)$ the set of interactions conflicting with α, that is, $\mathit{conf}(\alpha) = \{\beta \in \gamma \mid \alpha \# \beta\}$.

▶ **Property 8.** *Let (ℓ, v) and $(\ell, v + \delta')$, with $\delta' \in \mathbb{R}_{>0}$ be two states of the composition S. For an interaction $\alpha \in \gamma$, if Plannable(α, δ) holds at state (ℓ, v) then Plannable$(\alpha, \delta - \delta')$ also holds at any state $(\ell, v + \delta')$ such that $\delta' \leq \delta$.*

This property can be found directly by expressing the predicate 3 on state $(\ell, v + \delta')$.

As previously explained, due to communication latencies induced by execution platforms we assume that interactions cannot be planned in δ units of time if $\delta < h_{\min}$, where $h_{\min} \in \mathbb{Z}_{\geq 0}$ is a parameter representing the minimal *planning horizon*, which should represent the upper bound communication latencies. Notice that for the sake of simplicity, we consider a global parameter $h_{\min}$ but we could also assume different parameters for each interaction. Additionally, we also consider upper bounds planning horizons $h_{\max} : \gamma \to \mathbb{Z}_{\geq 0} \cup \{+\infty\}$ for each interaction such that for any $\alpha \in \gamma$ we have $h_{\max}(\alpha) \geq h_{\min}$. We denote by $h^{\infty}_{\max}$ the upper planning horizon assigning infinity to every $h_{\max}(\alpha)$. A direct consequence of introducing the planning horizons is that every interaction α can be planned only using a horizon δ satisfying $h_{\min} \leq \delta \leq h_{\max}(\alpha)$, meaning that every component $B \in \text{part}(\alpha)$ will be blocked for a duration between $[h_{\min}, h_{\max}(\alpha)]$. Observe that while $h_{\min}$ represents the worst case estimation of the communication delays for a given platform, the parameters $h_{\max}(\alpha)$ will be used later to find a strategy that avoids deadlocks by restricting the amount of time components can be blocked for.

For an interaction α we define the predicate *Plannable*(α) characterizing states from which α can be planned in a delay respecting the planning horizons $h_{\min}$ and $h_{\max}(\alpha)$, that is:

$$\textit{Plannable}(\alpha) \Leftrightarrow \exists \delta \in \mathbb{R}_{\geq 0} \;.\; h_{\min} \leq \delta \leq h_{\max}(\alpha) \;\wedge\; \textit{Plannable}(\alpha, \delta),$$

It can be written equivalently as follows:

$$\textit{Plannable}(\alpha) = \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \cdots, \ell_n)}} \mathsf{at}(\ell) \wedge \swarrow_{h_{\min}}^{h_{\max}(\alpha)} \Big(\bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \textit{guard}(a_i, \ell_i) \Big) \tag{4}$$

▶ **Definition 9** (Plan). A plan π is a function $\pi : \gamma \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ defining relative times for executing interactions. An interaction α is planned to execute in $\pi(\alpha)$ time units only if $\pi(\alpha) < +\infty$. Plans satisfy that for any two interactions $\alpha \neq \beta$ such that $\pi(\alpha) < +\infty$ and $\pi(\beta) < +\infty$, then the interactions α and β are not conflicting (i.e. $\neg(\alpha \# \beta)$).

We denote by π_0 the plan assigning $+\infty$ to every interaction of γ, that is, $\forall \alpha \in \gamma, \pi_0(\alpha) = +\infty$. For a plan π, we consider its minimum value $\min(\pi) = \min \{\pi(\alpha) | \alpha \in \gamma\}$. We also denote by $\textit{conf}(\pi)$ the set of interactions conflicting with the plan π, i.e. $\textit{conf}(\pi) = \{\alpha \mid \exists \beta \# \alpha. \pi(\beta) < +\infty\}$, and $\text{part}(\pi)$ the set of components participating in interactions planned by π, i.e. $\text{part}(\pi) = \{B_i \mid \exists \alpha \;.\; \pi(\alpha) < +\infty \wedge B_i \in \text{part}(\alpha)\}$. Notice that since π stores relative times, whenever time progresses by δ, the value $\pi(\alpha)$ assigned by π to an interaction α should be decreased by δ until it reaches 0, meaning that α has to execute. We write $\pi - \delta$ to describe the progress of time over the plan, that is, $(\pi - \delta)(\alpha) = \pi(\alpha) - \delta$ for interactions α such that $\pi(\alpha) < +\infty$. Similarly, $\pi[\alpha \mapsto \delta]$ assigns relative time δ to α, $\alpha \notin \textit{conf}(\pi)$, into existing plan π, i.e. $(\pi[\alpha \mapsto \delta])(\beta) = \delta$ for $\beta = \alpha$, $(\pi[\alpha \mapsto \delta])(\beta) = \pi(\beta)$ otherwise.

▶ **Definition 10** (Local Planning Semantics). Given a set of components $\{B_1, \cdots, B_n\}$ and an interaction set γ, we define the local planning semantics (LPS) of the composition $\gamma(B_1, \cdots, B_n)$, as the LTS $(\mathrm{Q}_p, q_{p_0}, \sum_p, \rightsquigarrow_\gamma)$ where:

- $\mathrm{Q}_p = \mathcal{L} \times \mathcal{V}(\mathcal{X}) \times \Pi$, where $\mathcal{L}$ is the set of global locations, $\mathcal{V}(\mathcal{X})$ is the set of global clock valuation, and Π is the set of plans.
- $\sum_p = \gamma \cup \mathbb{R}_{>0} \cup (\gamma \times \mathbb{R}_{\geq 0})$, where $(\gamma \times \mathbb{R}_{\geq 0})$ defines the action of planning interactions of γ and their relative times.

- $\rightsquigarrow_\gamma \subseteq \mathrm{Q}_p \times \sum_p \times \mathrm{Q}_p$ is the set of labeled transitions defined by the rules:
  - $(\ell, v, \pi) \overset{(\alpha,\delta)}{\rightsquigarrow}_\gamma (\ell, v, \pi[\alpha \mapsto \delta])$ for $\alpha \in \gamma$, $h_{\min} \leq \delta \leq h_{\max}(\alpha)$ and $\delta \neq +\infty$ if $\alpha \notin \mathit{conf}(\pi)$ and $\mathit{Plannable}(\alpha, \delta)$ holds on (ℓ, v, π).
  - $(\ell, v, \pi) \overset{\alpha}{\rightsquigarrow}_\gamma (\ell', v', \pi[\alpha \mapsto +\infty])$ for $\alpha \in \gamma$ if $\pi(\alpha) = 0$ and $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$.
  - $(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_\gamma (\ell, v+\delta, \pi-\delta)$ for $\delta \leq \min(\pi)$, $\ell = (\ell_1, \ldots, \ell_n)$, if $\mathsf{tpc}_i(\ell_i)(v+\delta)$ for components $B_i \in \mathrm{part}(\pi)$ and $\mathsf{tpc}_i(\ell_i)(v + \delta + h_{\min})$ for components $B_i \notin \mathrm{part}(\pi)$.

Remark that in the above definition as well as in what follows, predicates defined on states $(\ell, v) \in \mathrm{Q}_g = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ of the standard semantics are straightforwardly interpreted on states $(\ell, v, \pi) \in \mathrm{Q}_p$ considering the projection (ℓ, v) of (ℓ, v, π) on Q_g.

States of the LPS do not include only locations and clock valuations, but also the relative execution times of the planned interactions stored by π. Initially, no interaction is planned, that is, initial states (ℓ_0, v_0, π_0) satisfy $\pi_0 = +\infty$. Planning an interaction α to be executed at a relative time $h_{\min} \leq \delta \leq h_{\max}(\alpha)$ corresponds to the operation $\pi[\alpha \mapsto \delta]$ on the plan, which can only be done if α is not conflicting with the latter, and becomes enabled if time progresses by δ (i.e. if $\mathit{Plannable}(\alpha, \delta)$). On the other hand, time progress not only updates clock values but also the plan by decreasing the relative execution times of the planned interactions. To force the execution of planned interactions when their relative execution times reach 0, time cannot progress more than the closest relative execution times of the interactions (more than $\min(\pi)$). As for the standard semantics, time progress is limited by the time progress conditions of the components, but with the following significant difference: Components $B_i \in \mathrm{part}(\pi)$ participating in planned interactions behave as in the standard semantics, that is, time can progress until their time progress conditions expire. For components $B_i \notin \mathrm{part}(\pi)$, i.e., that are not participating in planned interactions, we take into account the minimal delay $h_{\min}$ needed for planning and then executing an interaction: in components $B_i \notin \mathrm{part}(\pi)$ time can progress only up to $h_{\min}$ time units before their time progress conditions expire. By doing so, we ensure that there always remains enough time to plan interactions involving $B_i \notin \mathrm{part}(\pi)$, if they exist, and execute them before their time progress conditions expire.

▶ **Example 11.** Let us consider the following execution sequence for example of Figure 1 under the LPS with $h_{\min} = 2$ and $h_{\max} = h^{\infty}_{\max}$.

$$\begin{array}{lll}
((\ell^1_0, \ell^2_0, \ell^3_0, \ell^4_0), (0,0,0), +\infty) & \overset{(\alpha_1,26)}{\rightsquigarrow}_\gamma ((\ell^1_0, \ell^2_0, \ell^3_0, \ell^4_0), (0,0,0), \{\alpha_1 \mapsto 26\}) & \overset{26}{\rightsquigarrow}_\gamma \\
((\ell^1_0, \ell^2_0, \ell^3_0, \ell^4_0), (26,26,26), \{\alpha_1 \mapsto 0\}) & \overset{\alpha_1}{\rightsquigarrow}_\gamma ((\ell^1_1, \ell^2_1, \ell^3_0, \ell^4_0), (26,26,26), +\infty) & \overset{(\alpha_3,2)}{\rightsquigarrow}_\gamma \\
((\ell^1_1, \ell^2_1, \ell^3_0, \ell^4_0), (26,26,26), \{\alpha_3 \mapsto 2\}) & \overset{2}{\rightsquigarrow}_\gamma ((\ell^1_1, \ell^2_1, \ell^3_0, \ell^4_0), (28,28,28), \{\alpha_3 \mapsto 0\}) & \overset{\alpha_3}{\rightsquigarrow}_\gamma \\
((\ell^1_0, \ell^2_2, \ell^3_0, \ell^4_0), (0,28,0), +\infty) & \overset{(\alpha_2,26)}{\rightsquigarrow}_\gamma ((\ell^1_0, \ell^2_2, \ell^3_0, \ell^4_0), (0,28,0), \{\alpha_2 \mapsto 26\}) & \overset{26}{\rightsquigarrow}_\gamma \\
((\ell^1_0, \ell^2_2, \ell^3_0, \ell^4_0), (26,54,26), \{\alpha_2 \mapsto 0\}) & \overset{\alpha_2}{\rightsquigarrow}_\gamma ((\ell^1_1, \ell^2_2, \ell^3_1, \ell^4_0), (26,54,26), +\infty) & \overset{(\alpha_4,2)}{\rightsquigarrow}_\gamma \\
((\ell^1_1, \ell^2_2, \ell^3_1, \ell^4_0), (26,54,26), \{\alpha_4 \mapsto 2\}) & \overset{2}{\rightsquigarrow}_\gamma ((\ell^1_1, \ell^2_2, \ell^3_1, \ell^4_0), (28,56,28), \{\alpha_4 \mapsto 0\}) & \overset{\alpha_4}{\rightsquigarrow}_\gamma \\
((\ell^1_0, \ell^2_2, \ell^3_2, \ell^4_0), (28,0,0), +\infty) & \overset{(\alpha_6,30)}{\rightsquigarrow}_\gamma & \\
((\ell^1_0, \ell^2_2, \ell^3_2, \ell^4_0), (28,0,0), \{\alpha_6 \mapsto 30\}) & &
\end{array}$$

This execution sequence represents a path that alternates plan actions, time progress and execution of some interactions, and leads to the action-time-lock state $((\ell^1_0, \ell^2_2, \ell^3_2, \ell^4_0), (0,0,28), \{\alpha_6 \mapsto 30\})$. In fact, the time progress condition $x \leq 30$ in component T_1, imposes the planning of interaction α_7 at the latest h_{min} units of time before it becomes urgent. However, since interaction α_6 was planned in 28 units of time, α_7 cannot be planned since it is conflicting with α_6. This execution sequence shows that a given system action-time-locks under the local planning semantics , even if it is deadlock-free in the standard semantics.

3.2 Properties of the LPS

We use weak simulation to compare models of the standard semantics and the local planning semantics by considering the planning transitions unobservable. As shown in Example 11, the LPS does not preserve the deadlock freedom property of our system. Nevertheless, the following proves weak simulation relations between the two semantics.

▶ **Lemma 12.** *Given a reachable state (ℓ, v, π) of the LPS. If for $\alpha \in \gamma$, $\pi(\alpha) < +\infty \Rightarrow$ $Plannable(\alpha, \pi(\alpha))$.*

▶ **Proposition 13.** *An interaction can execute from a state (ℓ, v, π) in the LPS semantics only if it can execute from (ℓ, v) in the standard semantics, that is:*

$$\forall \alpha \in \gamma.(\ell, v, \pi) \overset{\alpha}{\rightsquigarrow}_{\gamma} (\ell', v', \pi') \Rightarrow (\ell, v) \xrightarrow{\alpha}_{\gamma} (\ell', v').$$

Proposition 13 is a consequence of Lemma 12: an interaction α can execute in the local planning semantics only if $\pi(\alpha) = 0$ (see Definition 9). That is, a state (ℓ, v, π) of the LPS from which α can execute satisfies $Plannable(\alpha, 0)$ or equivalently $Enabled(\alpha)$, which demonstrates that α can execute from (ℓ, v) in the standard semantics.

▶ **Proposition 14.** *Time can progress by δ at a state (ℓ, v, π) in the local planning semantics only if time can progress by δ at (ℓ, v) in the standard semantics, that is:*

$$\forall \delta \in \mathbb{R}_{>0}.(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{\gamma} (\ell', v', \pi') \Rightarrow (\ell, v) \xrightarrow{\delta}_{\gamma} (\ell', v').$$

Proposition 14 is a direct consequence of the definition of time progress in the local planning semantics which is a restriction of the one in the standard semantics.

▶ **Corollary 15.** *If a state (ℓ, v, π) is reachable in the local planning semantics, then the state (ℓ, v) is reachable in the standard semantics.*

Corollary 15 is obtained from Propositions 13 and 14 and the fact that planning transitions (labeled by (α, δ)) affect only the plan π in states (ℓ, v, π) of the LPS.

▶ **Definition 16** (Weak Simulation). A weak simulation over $A = (Q_A, q_{A_0} \sum \cup \{\beta\}, \rightarrow_A)$ and $B = (Q_B, q_{B_0} \sum \cup \{\beta\}, \rightarrow_B)$, where β actions represent silent (unobservable) action, is a relation $R \subseteq Q_A \times Q_B$ such that:

- $\forall (q, r) \in R, a \in \sum .q \xrightarrow{a}_A q' \implies \exists r' : (q', r') \in R \wedge r \xrightarrow{\beta^* a \beta^*}_B r'$ and,
- $\forall (q, r) \in R : q \xrightarrow{\beta}_A q' \implies \exists r' : (q', r') \in R \wedge r \xrightarrow{\beta^*} r'$.

B simulates A, denoted by $A \sqsubseteq_R B$, means that B can do everything A does.

The definition of weak simulation is based on the unobservability of $\beta-$transitions. In our case, $\beta-$transitions corresponds to planning transitions. Let LTS_g and LTS_p be respectively the underlying labeled transition system of the standard semantics and the local planning semantics respectively.

▶ **Corollary 17.** *$LTS_p \sqsubseteq_R LTS_g$ with $R = \{((q, \pi); q) \in Q_p \times Q_g\}$.*

Corollary 17 corresponds to a notion of correctness of the local planning semantics: any execution in the LPS corresponds to an execution in the standard semantics. In addition, if interactions are allowed to be planned with relative execution times of 0 (i.e. $h_{\min} = 0$) then timeless planning of interactions becomes possible. Thus, the planning semantics simulates the standards semantics in that case.

▶ **Corollary 18.** *$LTS_g \sqsubseteq_{R'} LTS_p$ with $R' = \{(q;(q,\pi)) \in \mathrm{Q}_g \times \mathrm{Q}_p \mid h_{\min} = 0\}$.*

However, this is no longer true in general if $h_{\min} > 0$ which means that not all execution sequences of the standard semantics are preserved by the local planning semantics.

▶ **Corollary 19.** *If LTS_g is zeno runs free then LTS_p is too.*

Corollary 19 states that the LPS does not introduce any zenoness behavior if the standard semantics is free from the latter. It is a direct consequence of Corollary 17 and the fact that it is not possible to have infinite sequences of planning transitions without interaction execution (γ is finite and planning times are bounded).

▶ **Proposition 20.** *If every reachable state of LTS_g is not a deadlock, then a reachable state of LTS_p is not deadlock if and only if it is not an action-time-lock.*

Proof of Propostion 20. We prove Proposition 20 by contradiction. Let us assume that the system under the standard (resp. local planning) semantics is deadlock free (resp. action-time-lock-free). Let (ℓ, v, π) be a reachable deadlock state of the LPS. We have:

$$\nexists \sigma \in \gamma \cup (\gamma \times \mathbb{R}_{\geq 0}), \exists \delta.\ (\ell, v, \pi) \overset{\sigma}{\rightsquigarrow}_{\gamma} (\ell', v', \pi') \vee (\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{\gamma} (\ell, v+\delta, \pi-\delta) \overset{\sigma}{\rightsquigarrow}_{\gamma} (\ell', v', \pi')$$

We denote by $wait(\ell, v, \pi)$ the set of allowed waiting times at state (ℓ, v, π), that is:

$$wait(\ell, v, \pi) = \{0\} \cup \{\delta \in \mathbb{R}_{>0} | (\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{\gamma} (\ell, v+\delta, \pi-\delta)\}$$

We also put $\max(wait(\ell, v, \pi))$ to denote the maximal waiting time at state (ℓ, v, π). Notice that $\max(wait(\ell, v, \pi))$ may not be defined in some cases. In fact, we are not interested in its actual existence but rather in the fact that it is bounded ($< +\infty$) or not.

▶ **Lemma 21.** *Let (ℓ, v, π) be a reachable state of the local planning semantics. For $k \in \mathbb{R}_{\geq 0}$, such that $k = \max(wait(\ell, v, \pi))$, we have the following properties:*

P1 *If $k < +\infty$ then $(\ell, v, \pi) \overset{k}{\rightsquigarrow}_{\gamma} (\ell, v+k, \pi-k) \wedge wait(\ell, v+k, \pi-k) = \{0\}$*

P2 *If $\pi \neq \pi_0$ then $k \leq \min(\pi)$*

We distinguish 2 cases:

Case 1: no interaction is planned (i.e. $\pi = \pi_0$)

By definition of the LPS, it is clear that for $\pi = \pi_0$, there is no interaction to execute from (ℓ, v, π) or any of its successor $(\ell, v+\delta, \pi-\delta)$.

1. $wait(\ell, v, \pi) = \{0\}$:
   This means that time progress is not allowed at state (ℓ, v, π). We also have $\nexists \sigma \in (\gamma \times \mathbb{R}_{\geq 0}).(\ell, v, \pi) \overset{\sigma}{\rightsquigarrow}_{\gamma} (\ell', v', \pi')$ (deadlock assumption). We can conclude that (ℓ, v, π) is a reachable action-time-lock state, which contradicts the assumption that the system under the local planning semantics is action-time-lock-free.
2. $wait(\ell, v, \pi) \neq \{0\}$:
   a. $\max(wait(\ell, v, \pi)) = +\infty$:

   ▶ **Lemma 22.** *Let (ℓ, v, π) be a reachable state of the local planning semantics . If $\forall \delta \in \mathbb{R}_{>0}$. $(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{\gamma} (\ell, v+\delta, \pi-\delta) \wedge \neg Plannable(\alpha)$ at (ℓ, v, π), then we have $\neg Enabled(\alpha)$ at $(\ell, v+\delta, \pi-\delta)$ with $\delta \geq h_{\min}$.*

By P1 of Lemma 21 we can deduce that $\exists \delta \geq h_{\min}$ such that $(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_\gamma (\ell, v + \delta, \pi - \delta)$. We also have from the deadlock assumption and Lemma 22: $\bigwedge_{\alpha \in \gamma} \neg Enabled(\alpha)$. Finally, since the state $(\ell, v + \delta, \pi - \delta)$ is reachable in the standard semantics, and by evaluating the deadlock characterization 2 on state $(\ell, v + \delta, \pi - \delta)$, we can conclude that the system under the standard semantics deadlocks, which contradicts the assumption of deadlock freedom of the system under the standard semantics.

b. $\max(wait(\ell, v, \pi)) < +\infty$:
Considering that $k = \max(wait(\ell, v, \pi))$, then we have by P1 of Lemma 21: $(\ell, v, \pi) \overset{k}{\rightsquigarrow}_\gamma (\ell, v + k, \pi - k) \wedge wait(\ell, v + k, \pi - k) = \{0\}$. Using the deadlock assumption we have: $\bigwedge_{\alpha \in \gamma} \neg Plannable(\alpha)$ at state $(\ell, v + k, \pi - k)$. Since the system cannot progress beyond this state ($wait(\ell, v + k, \pi - k) = \{0\}$), we can conclude that $(\ell, v + k, \pi - k)$ is a reachable action-time-lock state, which contradicts the assumption that the system under the local planning semantics is action-time-lock-free.

Case 2: at least an interaction is planned (i.e. $\pi \neq \pi_0$)

Considering that $k = \max(wait(\ell, v, \pi))$, since $\pi \neq +\infty$, we have by 2 of Lemma 21: $k < +\infty \wedge k \leq \min \pi$. Using the deadlock assumption we can infer that $k < \min \pi$, since no execution is possible from (ℓ, v, π) or any of its successors. This means that $(\ell, v+k, \pi-k)$ is a reachable action-time-lock state, which contradicts the assumption that the system under the LPS is action-time-lock-free. ◀

4 Enforcing Deadlock-Free Planning

As explained in previous section, the local planning semantics is based on local conditions for planning interactions and may exhibit deadlocks even when the system is deadlock-free with the standard semantics. Such deadlocks are partly due to the fact that planning an interaction may block, in addition to the participating components, extra components whose timing constraints are not considered by these local conditions. In this section, we investigate simple execution strategies that only restrict the horizon used for planning interactions with upper bounds. By reducing the period of time during which components are blocked, they tend to remove deadlocks from the reachable states. In what follows, we consider a composition of components $S = \gamma(B_1, \cdots, B_n)$ such that it is deadlock-free in the standard semantics.

▶ **Corollary 23** (Sufficient Condition for Deadlock-freedom). *If a reachable state of the planning semantics is not an action-time-lock then it is not deadlock.*

Corollary 23 is a direct consequence of Proposition 20. It affirms that for systems that are initially deadlock-free under the standard semantics, it is sufficient to prove action-time-lock-freedom of the LPS to prove its deadlock-freedom.

▶ **Proposition 24.** *A reachable state (ℓ, v, π) of the local planning semantics is an action-time-lock if and only if:*

$$\pi > 0 \;\wedge \bigwedge_{\alpha \notin conf(\pi)} \neg Plannable(\alpha) \;\wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \;\wedge (urg(\ell_i) + h_{\min}).$$

The above proposition derives directly from the definition of action-time-locks on a state of the local planning semantics. As shown in Example 11, the local planning semantics may introduce deadlocks. The source of deadlocks is twofold: *(i)* due to communication delays, consecutive execution in a component are separated by at least $h_{\min}$ units of time which may be incompatible with its timings constraints, and *(ii)* conditions for planning interactions are too permissive as they only take into account timing constraints of participating components whereas they may block additional components, namely the ones participating in conflicting interactions. In the rest of the paper, we study how to generate planning strategies for preserving deadlock-freedom by restricting the planning transitions of the LPS so that deadlock states become unreachable. Such a strategy may not exist when timing constraints cannot accommodate with the communication delays $h_{\min}$.

From Corollary 23, action-time-lock-freedom is a sufficient condition for deadlock-freedom of the LPS. By Proposition 24, a state (ℓ, v, π) is an action-time-lock in the local planning semantics if and only if no time progress is allowed nor planning or execution of interactions from (ℓ, v, π), that is:

$$\pi > 0 \ \wedge \bigwedge_{\alpha \in \gamma \setminus conf(\pi)} \neg \mathit{Plannable}(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}).$$

The above predicate characterizes the fact that no interaction can be executed or planned, nor time can progress in component $B_i \notin \mathrm{part}(\pi)$. Consequently, we deduce that a necessary condition of action-time-lock is the existence of a component $B_i \notin \mathrm{part}(\pi)$ such that time cannot progress in B_i and B_i cannot be planned in an interaction, that is:

$$\bigwedge_{\alpha \in \gamma(B_i) \setminus conf(\pi)} \Big(\neg \mathit{Plannable}(\alpha) \ \wedge \bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \Big).$$

where $\gamma(B_i)$ denotes the subset of interactions in which B_i participates, that is, $\gamma(B_i) = \{\beta \in \gamma \mid B_i \in \mathrm{part}(\beta)\}$. Notice that the above expression strongly depends on the plan π, which is difficult to characterize in practice. The following theorem proposes sufficient plan-independent condition characterizing action-time-lock states of the LPS .

▶ **Theorem 25.** *Let ϕ be the following predicate:*

$$\bigvee_{1 \le i \le n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \ \vee \bigvee_{\substack{\beta \in conf(\alpha) \\ B_i \notin \mathrm{part}(\beta)}} \widetilde{\mathit{Plannable}(\beta)} \Big) \Big].$$

where $\widetilde{\mathit{Plannable}(\beta)}$ is the predicate defined as follows:

$$\widetilde{\mathit{Plannable}(\beta)} \Leftrightarrow \exists \delta > 0 \ . \ \delta \le h_{\max}(\beta) \ \wedge \ \mathit{Plannable}(\beta, \delta).$$

We prove that a reachable action-time-lock state (ℓ, v, π) satisfies ϕ.

Proof of Theorem 25. A reachable action-time-lock state of the LPS satisfies:

$$\pi > 0 \ \wedge \bigwedge_{\alpha \in \gamma(B_i) \setminus conf(\pi)} \Big(\neg \mathit{Plannable}(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \Big).$$

In order to approximate the above formula, we distinguish two cases:

Case 1: no interaction is planned (i.e. $\pi = \pi_0$)

From $\pi = +\infty$ we deduce directly that there exists an urgent component B_i such that no interaction α involving B_i can be planned, that is:

$$\bigvee_{1 \le i \le n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \neg \mathit{Plannable}(\alpha) \Big]. \tag{1}$$

Case 2: at least an interaction is planned (i.e. $\pi \neq \pi_0$)

In this case, there exists an urgent component $B_i \notin \mathrm{part}(\pi)$ such that no interaction α involving B_i can be planned, either because it conflicts with a planned interaction β $(0 < \pi(\beta) < +\infty)$ or because $\mathit{Plannable}(\alpha)$ is not satisfied, that is $\exists \beta \in \pi, \exists B_i \notin \mathrm{part}(\beta)$ satisfying:

$$(0 < \pi(\beta) < +\infty) \;\wedge \bigwedge_{\alpha \in \gamma(B_i) \setminus \mathit{conf}(\beta)} \neg \mathit{Plannable}(\alpha) \;\wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\beta)}} \mathsf{at}(\ell_i) \;\wedge (\mathit{urg}(\ell_i) + h_{\min}).$$

or equivalently $\exists \beta \in \pi, \exists B_i \notin \mathrm{part}(\beta)$ satisfying:

$$\bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\beta)}} \mathsf{at}(\ell_i) \wedge (\mathit{urg}(\ell_i) + h_{\min}) \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \vee \big(\beta \in \mathit{conf}(\alpha) \wedge (0 < \pi(\beta) < +\infty)\big)\Big).$$

By noticing that we have the following implication between quantifiers $\exists y, \forall x. Q(x, y) \implies \forall x, \exists y. Q(x, y)$, we can deduce that the above condition implies:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \;\wedge (\mathit{urg}(\ell_i) + h_{\min}) \;\wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \;\vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \mathrm{part}(\beta)}} 0 < \pi(\beta) < +\infty \Big) \Big].$$

As $\pi > 0$, and if we consider only reachable action-time-locks, we have $0 < \pi(\beta) \leq h_{\max}(\beta)$, and by Lemma 12 we have $\mathit{Plannable}(\beta, \pi(\beta))$. That is, β satisfies $\mathit{Plannable}(\beta)$ in which the lower bound $h_{\min}$ is replaced by the strict lower bound 0, i.e. $\widetilde{\mathit{Plannable}(\beta)}$. Then, the above expression becomes:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \;\wedge (\mathit{urg}(\ell_i) + h_{\min}) \;\wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \;\vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \mathrm{part}(\beta)}} \widetilde{\mathit{Plannable}(\beta)} \Big) \Big]. \quad (2)$$

By remarking that Expression 1 implies Expression 2, we can conclude that an action-time-lock of the local planning semantics satisfies:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \;\wedge (\mathit{urg}(\ell_i) + h_{\min}) \;\wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \;\vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \mathrm{part}(\beta)}} \widetilde{\mathit{Plannable}(\beta)} \Big) \Big]. \quad ◀$$

Notice that due to the monotony of ϕ on upper bound horizons, we obtain the following lemma:

▶ **Lemma 26.** *If LTS_p is action-time-lock free for the upper bound horizons function $h_{\max}$, then it is action-time-lock free for any upper bound horizon function $h'_{\max} \leq h_{\max}$.*

In order to attest that planning interactions does not introduce deadlocks, we use an SMT solver to check the satisfiability of ϕ. As explained earlier, a given system is deadlock-free under the restricted LPS if $\mathit{Reach}(LTS_p) \wedge \phi$ is unsatisfiable. Since $\mathit{Reach}(LTS_p) \subseteq \mathit{Reach}(LTS_g)$ (Corollary 17), we can verify the above on $\mathit{Reach}(LTS_g)$. Effectively, we do not compute $\mathit{Reach}(LTS_g)$ to avoid the combinatorial explosion problem, inherent to composition of timed automata. In fact, we rather build an over-approximation, $\widetilde{\mathit{Reach}(LTS_g)}$, of the latter, and use it during our verification. Finding a strategy granting action-time-lock-freedom is based on the idea of restricting the upper bound horizon function $h_{\max}$. In fact, since $h_{\min}$ is a parameter that is dependent of the communication latency of a given execution platform, it cannot be tuned. Instead, initially for each interaction $\alpha \in \gamma$, $h_{\max}(\alpha) = +\infty$. Thereafter, due to the monotony of ϕ (Lemma 26) on upper horizons, this parameter will be refined, that is, its maximum will be decreased until finding a function $h_{\max}$ for which $\widetilde{\mathit{Reach}(LTS_g)} \wedge \phi$ is unsatisfiable or until reaching the upper horizon function $h_{\max}^{h_{\min}}$ for which $h_{\max}(\alpha) = h_{\min}$ for every $\alpha \in \gamma$ and such that $\widetilde{\mathit{Reach}(LTS_g)} \wedge \phi$ is satisfiable.

5 Planning Semantics as Real-Time Controller Synthesis

In Section 5, we presented a method that provides execution strategies by restricting the upper bounds planning horizon for each interaction. This strategy aims to preserve the deadlock-freedom property of a given system under the local planning semantics without imposing further scheduling constraints. This approach relies on the verification of a given expression on over-approximations of the reachable states of the initial semantics. Thus, it may give false-positive results due to *(i)* the nature of expression to check (sufficient condition) and *(ii)* the over-approximation of the reachable states of the LPS using over-approximations of the reachable states of the standard semantics (Corollary 17).

In such cases, an alternative is to tackle the problem as a real-time controller synthesis problem. Real-time controller synthesis is a common method used to extract an execution strategy from formal specifications satisfying certain properties. Usually, these properties express the reachability (resp. non-reachability) of a set of winning states (resp. bad states). In case of planning interactions with bounded horizons, the idea is to restrict the transition relation so that all the remaining behaviors do not lead to states where a component is urgent and no possible execution including this component may occur. This can be formalized as a reachability game for a timed game automaton [12], where the main idea consists in trying to find an execution strategy guaranteeing that a given set of namely *bad states* of the system are never reached.

In order to apply this approach, it is required to encode the planning of interactions and their effects on the system, that is, *(i)* encode interactions planning as synchronizations between components, *(ii)* reserve the components of the planned interactions until their chosen execution date, i.e, keep track of the planned interactions and their execution dates, and *(iii)* characterize the set of bad states. Thereafter, tools such as UPPAAL-Tiga [6] can be used to find an execution strategy of the planning semantics avoiding the set of bad states, that is, deadlock states. Expressing the planning problem as a real-time controller synthesis problem is not an easy task. Hereinafter, we discuss the different issues met during the formalization process and provide suggestions for solving them.

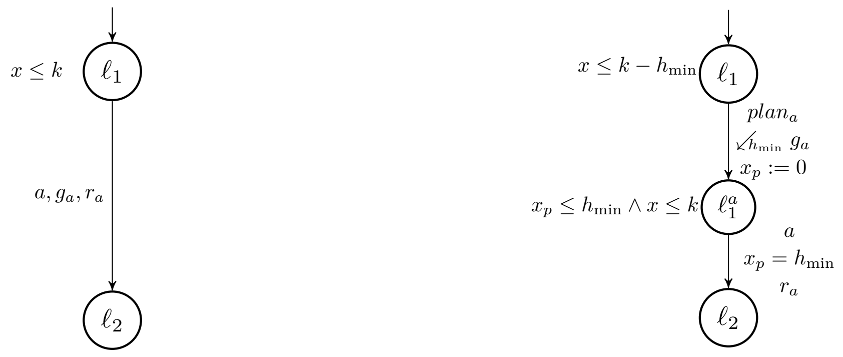
5.1 Planning Zones

From expression 4, we can see that the clocks values for planning an interaction α are calculated at a global level, that is, by applying the $\swarrow_{h_{\min}}^{h_{\max}(\alpha)}$ on the conjunction of its participating actions timing constraints. Notice that for a timing constraint $g = g_1 \wedge g_2$, we have:

$$\swarrow_{h_{\min}}^{h_{\min}} g = \swarrow_{h_{\min}}^{h_{\min}} (g_1 \wedge g_2) \implies \swarrow_{h_{\min}}^{h_{\min}} g_1 \wedge \swarrow_{h_{\min}}^{h_{\min}} g_2 \tag{5}$$

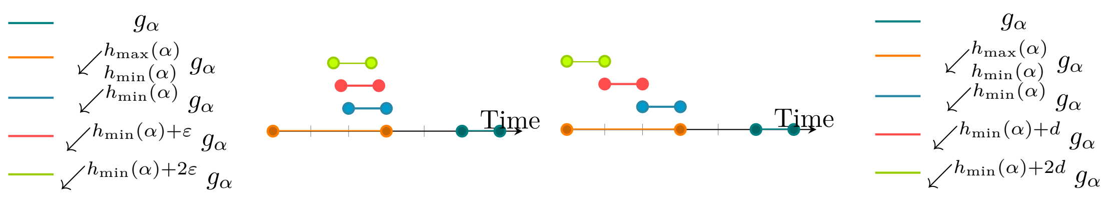
The above expression bears out the fact that planning states must be encoded on the composition of the system model and not on individual components. Particularly, expression 5 points out the fact that encoding the planning on transitions of individual components will induce additional behavior ($\swarrow_{h_{\min}}^{h_{\min}} (g_1 \wedge g_2) \implies \swarrow_{h_{\min}}^{h_{\min}} g_1 \wedge \swarrow_{h_{\min}}^{h_{\min}} g_2$). This represents the first drawback of this method since building the composition may be tedious especially for big scale systems. Therefore, a simple solution to avoid computing the composition is to consider models with interactions having timing constraints on up to one of their participating actions, that is, given an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$, we have $g_\alpha = \mathit{true}$ or $g_\alpha = g_{a_i}$, with $g_{a_j} = \mathit{true}$ for $j \in I, j \neq i$. In fact, considering interactions including up to one action with timing constraints, will allow to encode the planning on individual components that, additionally to the defined synchronizations (interactions), will also synchronize their planning actions.

The idea is to split each transition of the initial model into two transitions: *(1)* a planning transition, followed by *(2)* an execution transition after the plan transition being performed.



(a) Part of a timed automaton. **(b)** Planning encoding.

Figure 2 Planning as a Timed Automaton.


g_α
$\swarrow^{h_{\max}(\alpha)}_{h_{\min}(\alpha)}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)+\varepsilon}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)+2\varepsilon}\ g_\alpha$
Time
Time
g_α
$\swarrow^{h_{\max}(\alpha)}_{h_{\min}(\alpha)}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)+d}\ g_\alpha$
$\swarrow^{h_{\min}(\alpha)+2d}\ g_\alpha$

An interaction guard g_α and its planning intervals. Discretized planning intervals for g_α.

Figure 3 Discretizing Planning Horizons for Interaction.

For an interaction $\alpha \in \gamma$, the choice of the planning horizon, that is, the duration for which components participating in α will be blocked for until their execution, will be encoded on the execution transition of the component whose action $a_i \in \alpha$ and $g_\alpha = g_{a_i}$. Otherwise, if $g_\alpha = true$ this choice is made arbitrarily. Consequently, this component will be equipped with a clock x_p that will be used to track the planning dates. Finally, time progress conditions must also be translated to enforce planning at the latest $h_{\min}$ units of time before their expiry. Figure 2 depicts an overview of such transformation for $\delta = h_{\min}$ horizon:

5.2 Infinite Planning Transitions

Effectively, in order to encode the planning in timed automata, horizons values must be integer. Moreover, due to the dense time nature of the planning intervals (relative planning date for each interactionα are in $[h_{\min}, h_{\max}(\alpha)]$), we end up with an infinity of plan transitions, especially when not restricted upper bound planning horizons, i.e., $h_{\max} = h^{\infty}_{\max}$. Consequently, the first thing to do is to restrict for each interaction $\alpha \in \gamma$ the upper bound planning horizon $h_{\max}(\alpha)$.Thereafter, we propose to discretize the planning horizons in order to obtain finite values in $\mathbb{Z}_{>0}$ (Figure 3). In what follows, we denote by $Disc : \gamma \longrightarrow \mathcal{D}$ the discretized horizon function defining for each interaction its respective discretized planning horizons $\mathcal{D} \subset \mathbb{Z}_{>0}$.

▶ **Definition 27** (Planning Timed Automaton). Given n timed components $\mathcal{B}_i = (\mathcal{L}_i, \ell_0^i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{X}_i, \mathcal{I}_i)$ synchronizing through the interaction set γ such that, for each interaction $\alpha \in \gamma$, the guard of α is equal to the guard of one of its included actions. We define the corresponding planning model as the composition of the n timed automata $\mathcal{B}_i^p = (\mathcal{L}_i^p, \ell_0, \mathcal{A}_i \cup \mathcal{P}_i, \mathcal{T}_i^p, \mathcal{X}_i \cup \{x_i^p\}, \mathcal{I}_i^p)$, w.r.t the interaction set $\gamma \cup \mathcal{P}$, where:

- $\mathcal{P}_i = \cup_{a \in \mathcal{A}_i} p_a$ is the set of *Planning Actions*
- $\mathcal{P} = \{p_\alpha = \{p_{a_i}\}_{i \in I} | \alpha \in \gamma \wedge \alpha = \{a_i\}_{i \in I}\}$ is the set of *Planning Interactions*
- x_i^p is a *Tracking Clock* for interactions execution in each component
- $\mathcal{L}_i^p = (\mathcal{L}_i \cup \mathcal{L}_{i_p})$ is the set of control locations, where $\mathcal{L}_{i_p}$ is the set of locations following planning actions
- $\mathcal{T}_i^p$ is such that for each $(\ell_i, a_i, g_i, r_i, \ell_i') \in \mathcal{T}_i$, $a_i \in \alpha$ and for each $\delta \in Disc(\alpha)$:
  - if $g_\alpha \neq true$ we have:

    $$\text{Planning transitions: } \begin{cases} \ell_i \xrightarrow{p_{a_i}, true, \emptyset} \ell_{a_i}, & \text{if } g = true \\ \ell_i \xrightarrow{p_{a_i}, \swarrow^{\delta} g_i, r(x_i^p)} \ell_{a_i}^\delta, & \text{otherwise} \end{cases}$$

    $$\text{Execution transitions: } \begin{cases} \ell_{a_i} \xrightarrow{a, true, r_i} \ell_i', & \text{if } g = true \\ \ell_{a_i}^\delta \xrightarrow{a, g_a \wedge x_i^p = \delta, r_i} \ell_i', & \text{otherwise} \end{cases}$$

    where $\ell_a, \ell_{a_i}^\delta \in \mathcal{L}_{i_p}$.

  - if $g_\alpha = true$, we choose one action $b \in \alpha$:

    $$\text{Planning transitions: } \begin{cases} \ell_i \xrightarrow{p_{a_i}, true, \emptyset} \ell_{a_i}, & \text{if } a \neq b \\ \ell_i \xrightarrow{p_{a_i}, true, r(x_i^p)} \ell_{a_i}^\delta, & \text{otherwise} \end{cases}$$

    $$\text{Execution transitions: } \begin{cases} \ell_{a_i} \xrightarrow{a_i, true, r_i} \ell_i', & \text{if } a \neq b \\ \ell_{a_i}^\delta \xrightarrow{a_i, g_i \wedge x_i^p = \delta, r_i} \ell_i', & \text{otherwise} \end{cases}$$

- $\mathcal{I}_i^p$ is the set of *Location Invariants* , such that $\forall \ell_i^p \in \mathcal{L}_i^p$, we have:

  $$\mathcal{I}_i^p(\ell_i^p) = \begin{cases} \mathsf{tpc}(\ell_i) - h_{\min}, & \text{if } \ell_i^p = \ell_i \in \mathcal{L}_i \\ x_i^p \leq \delta \wedge \mathsf{tpc}(\ell_i), & \text{if } \ell_i^p = \ell_{a_i}^\delta \in \mathcal{L}_{i_p} \text{ such that } \ell_i \in \mathcal{L}_i \wedge \ell_i \xrightarrow{p_{a_i}} \ell_{a_i}^\delta, \end{cases}$$

For a composition $\gamma(B_1, \cdots, B_n)$, let $LTS_{p'} = (\mathrm{Q}_{p'}, \gamma' \cup \mathbb{R}_{>0}, \longrightarrow_{\gamma'})$, where $\gamma' = \gamma \cup \mathcal{P}$, be the corresponding labeled transition system of its planning model under the standard semantics.

► **Theorem 28.** *$LTS_{p'} \sqsubseteq_{R'} LTS_g$ where R' is the relation defined as follows: For $q^p = (\ell^p, v^p) \in \mathrm{Q}_{p'}$ and $q^g = (\ell^g, v^g) \in \mathrm{Q}_g$, such that $(q^p, q^g) \in R'$, we have:*

- *$\ell^p = (\ell_1^p, \cdots, \ell_n^p)$, $\ell_g = (\ell_1^g, \cdots, \ell_n^g)$:*

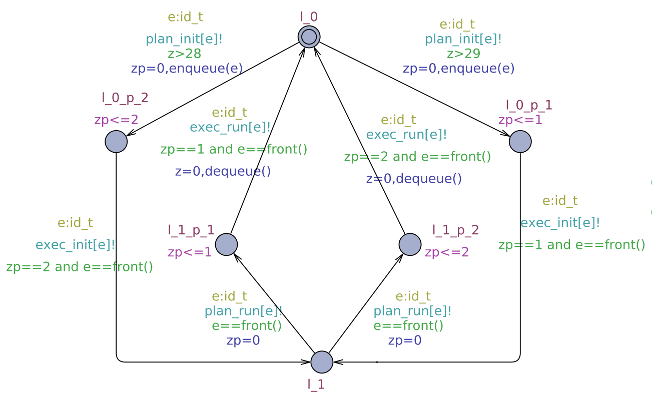
  $$\forall i \in \{1, \cdots, n\},\ \ell_i^g = \begin{cases} \ell_i^p, & \text{if } \ell_i^p \in \mathcal{L}_i, \\ \ell_i, & \text{if } \ell_i^p \in \mathcal{L}_{i_p} \text{ with } \ell_i \xrightarrow{a,g,r} \ell_i^p \in \mathcal{T}_i^p \wedge \ell_i \in \mathcal{L}_i, \end{cases}$$

  Notice that for the case where $\ell_i^p \in \mathcal{L}_{i_p}$, ℓ_i is unique by construction of the planning model.
- *$v^g = equ(v^p)$, where $equ(v^p)$ is the projection of v^p on clocks of v^g*

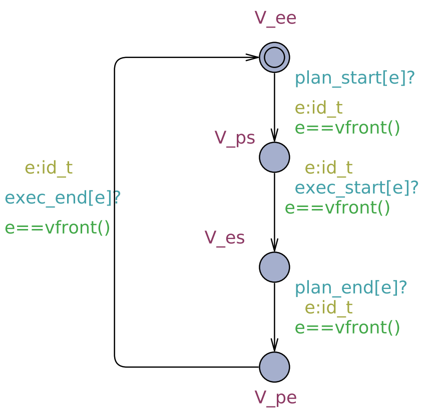
Proof of Theorem 28. To prove that $LTS_{p'} \sqsubseteq_{R'} LTS_g$, we need to prove that:

1. $\forall (q^p, q^g) \in R', \sigma \in \gamma \cup \mathbb{R}_{>0}$ such that $q^p \xrightarrow{\sigma}_{\gamma'} q'^p \Rightarrow \exists q'^g.(q'^p, q'^g) \in R' \wedge q^g \xrightarrow{\sigma}_{\gamma} q'^g$
2. $\forall (q^p, q^g) \in R', p_\alpha \in \mathcal{P}$ such that $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p \Rightarrow (q'^p, q^g) \in R'$

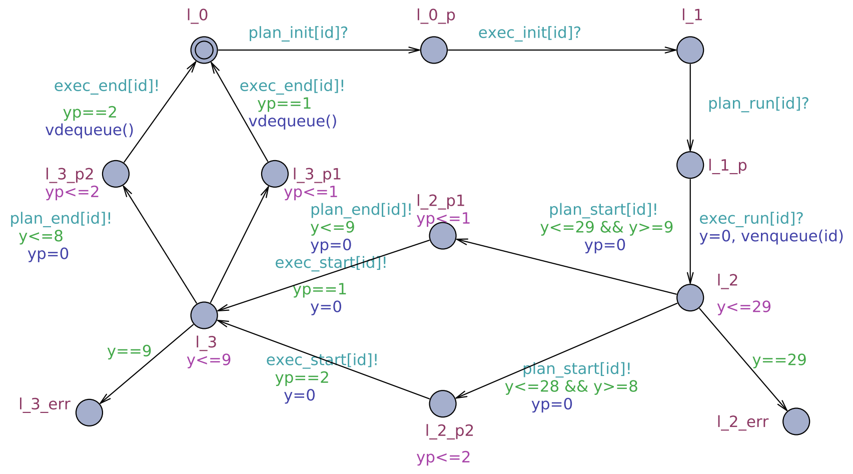
1. a. Suppose that $(q^p, q^g) \in R', \sigma = \alpha \in \gamma$ and $q^p \xrightarrow{\alpha}_{\gamma'} q'^p$ with $q'^p = ((\ell'^p_1, \cdots, \ell'^p_n), v'^p)$. We have: $q^p \xrightarrow{\alpha}_{\gamma'} q'^p \Rightarrow g_\alpha$ is *true*, and for $\alpha = \{a_i\}_{i \in \mathcal{I}}$, by construction of the planning automaton, we have: $\ell_i^g \xrightarrow{a_i, g_i, r_i} \ell'^g_i$ such that $\ell'^g_i = \ell'^p_i$. Moreover, since the same clocks are reset by the execution of α in both models, we deduce that $v'^g = equ(v'^p)$. By remarking that the state of components not participating in α remains the same, we conclude that $\exists q'^g$ such that $q^g \xrightarrow{\alpha}_{\gamma} q'^g \wedge (q'^p, q'^g) \in R'$.



(a) Controller Component.



(b) Variable Component.



(c) Task Component.

Figure 4 Planning Automata for the Task Manager Example.

b. Suppose that $(q^p, q^g) \in R', \sigma \in \mathbb{R}_{>0}$ and $q^p \xrightarrow{\sigma}_{\gamma'} q'^p$. For $q_i^p = (\ell_i^p, v_i^p)$, we define $\mathcal{I}_g$ the set of indexes such that $\ell_i^p \in \mathcal{L}_i$, and $\mathcal{I}_p$ the set of indexes such that $\ell_i^p \in \mathcal{L}_{p_i}$.
- $\forall i \in \mathcal{I}_g.\ell_i^p = \ell_i^g \wedge q_i^p \xrightarrow{\sigma} q'^p_i \Rightarrow q_i^g \xrightarrow{\sigma} q'^g_i$. This implication is a direct result of the planning model definition since: $\sigma \le \mathcal{I}(\ell_i^p) \le \mathsf{tpc}(\ell_i^g) - h_{\min}$.
- $\forall i \in \mathcal{I}_p.\ell_i^g = \ell_i$ such that $\ell_i^p \in \mathcal{L}_{i_p}$ with $\ell_i \xrightarrow{a,g,r} \ell_i^p \in \mathcal{T}_i^p \wedge \ell_i \in \mathcal{L}_i$. Thus $q_i^p \xrightarrow{\sigma} q'^p_i \Rightarrow q_i^g \xrightarrow{\sigma} q'^g_i$, since $\mathcal{I}(\ell_i^p) \implies \mathsf{tpc}(\ell_i^g)$.

We conclude that $\exists q'^g$ such that $q^g \xrightarrow{\sigma}_{\gamma} q'^g \wedge (q'^p, q'^g) \in R'$.

2. Suppose that $(q^p, q^g) \in R'$ and $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p$, with $p_\alpha \in \mathcal{P}$ and $q'^p = ((\ell'^p_1, \cdots, \ell'^p_n), v'^p)$. We have: $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p \Rightarrow$ for $\alpha = \{a_i\}_{i \in \mathcal{I}}$ $\ell_i^g = \ell_i^p \wedge \ell_i^g \xrightarrow{p_{a_i}} \ell'^p_i$. Moreover, since planning actions reset only the clocks x_i^p for tracking execution time, we can deduce that $(q'^p, q^g) \in R'$. ◀

Once interactions planning encoded, one last thing to do is to add the set of bad states to each planning automaton (if needed) and find a strategy to avoid those states. Figure 4 depicts the corresponding planning automata for example of Figure 1 with respect to Definition 27. Locations suffixed by p, correspond to locations following planning actions, whereas locations ending with *err* define the bad states, that is, states with urgent time progress condition(s) and no possible execution removing the urgency. In this example, for each interaction $\alpha \in \gamma$, we chose $\mathcal{D}(\alpha) = \{1, 2\}$. Notice that for this example, we consider that all actions are controllable actions

```
State:  ( Controller.l_1_p_1 Task(0).l_0 Task(1).l_1_p Task(2).l_3_p2
Task(3).l_0 Task(4).l_0 Task(5).l_0 Task(6).l_0 Task(7).l_0 Task(8).l_0
Task(9).l_0 Task(10).l_0 Task(11).l_0 Task(12).l_0 Task(13).l_0 Task(14).l_0
Task(15).l_0 Task(16).l_0 Task(17).l_0 Task(18).l_0 Task(19).l_0 Var.V_pe
) vlist[0]=2 vlist[1]=0 vlist[2]=0 vlist[3]=0 vlist[4]=0 vlist[5]=0
vlist[6]=0 vlist[7]=0 vlist[8]=0 vlist[9]=0 vlist[10]=0 vlist[11]=0
vlist[12]=0 vlist[13]=0 vlist[14]=0 vlist[15]=0 vlist[16]=0 vlist[17]=0
vlist[18]=0 vlist[19]=0 vlen=1 Controller.list[0]=1 Controller.list[1]=0
Controller.list[2]=0 Controller.list[3]=0 Controller.list[4]=0
Controller.list[5]=0 Controller.list[6]=0 Controller.list[7]=0
Controller.list[8]=0 Controller.list[9]=0 Controller.list[10]=0
Controller.list[11]=0 Controller.list[12]=0 Controller.list[13]=0
Controller.list[14]=0 Controller.list[15]=0 Controller.list[16]=0
Controller.list[17]=0 Controller.list[18]=0 Controller.list[19]=0
Controller.len=1
When you are in (Controller.zp==1 && Task(2).yp<=2), take transition
Controller.l_1_p_1-> Controller.l_0 { zp == 1 && 1 == front(), exec_run[1]!,
z := 0, dequeue() } Task(1).l_1_p->Task(1).l_2 { 1, exec_run[id]?, y := 0,
venqueue(id) } When you are in (Task(2).yp==2 && Controller.zp<=1), take
transition Task(2).l_3_p2-> Task(2).l_0 { yp == 2, exec_end[id]!, vdequeue()
} Var.V_pe->Var.V_ee { 2 == vfront(), exec_end[2]?, 1 }
```

Figure 5 Sample of the Output Strategy from UPPAAL-Tiga.

since it is a closed system in the sense that there is no interaction with the environment.

We performed the verification on the Task Manager examples with 20 tasks. The winning condition being a safety condition: avoid all *"err"* locations. This was translated into the following property:

control: A[ ] forall (i : int[0,N-1]) not (Task(i).l_2_err or Task(i).l_3_err) (6)

The property of interest was successfully verified. Additionally, we were also able to synthesize all wining actions of all states using the command line of UPPAAL-Tiga. A sample of the resulting output is provided below Figures 5. Notice that the average execution time[1] for verifying Property 6 is 0.1141 seconds (0.6534 seconds when requesting the generation of a strategy).

5.3 Discussion

In this section, we explained how the problem of planning interactions can be formalized into a real-time controller synthesis approach. However, this approach has some drawbacks. In order to encode planning of interactions in components as timed automata, this approach restricts its scope to discretized horizon values which results in having less control over the planning dates of interactions, and leads in case of a high number of discretized values, to an explosion in the number of planning transitions. Unfortunately, we do not have an immediate solution for this problem. In fact, it is user dependent since one user may just want to have a ASAP execution for a given interaction, for instance because the components involved in this interaction are often

[1] The experiments have been conducted on a HP machine with Ubuntu 16.04, an Intel® Core™i5-4300U processor of frequency 1.90GHz×4, and 7.7GiB memory.

requested, and in that case the practice will be to always plan with $h_{\min}$. In other cases, the user may want to plan an interaction with flexible amount time. Additionally, this approach considers only a class of systems where interactions have timing constraints on up to one of their participating components action. Otherwise, the planning should be encoded on the composition, which represents a tedious work because of the state space explosion problem. Nevertheless, this approach differs form the usual scheduler synthesis approach since it is not performed on the regular semantics of timed automata. Particularly, here we are interested in avoiding bad states of the planning semantics (states that verify the expression of Theorem 25). Consequently, unless finding an automatic general method for generating such complex expression in the query language accepted by such tools, and without ignoring that finding a strategy avoiding those states may be hard in term of computational complexity, our real-time controller synthesis approach seems more straightforward and much simpler but it comes with some feasibility restriction.

6 Implementation and Experiments

6.1 Implementation

For our experiments, we used the BIP framework [5] as a modeling language to define systems and their synchronizations. BIP (Behavior, Interaction, Priority), is a component-based framework with a rigorous semantics that allows to model systems as a set of atomic components coordinating their behaviors through multiparty interactions. The BIP framework provides a rich set of tools that allows to model, verify and execute systems. The BIP toolbox is structured in different categories (see Figure 6):

1. This category includes *translation of various language or modeling paradigm* that allows the automatic generation of BIP models as well as the front-end of the BIP compiler.
2. The middle-end of the BIP compiler consists of several modules that allows model transformation (from BIP to BIP) and performance optimization (flattening and distributed real-time filter). Particularly, the distributed real-time filter provides an intermediate model transformation (Send/Receive transformation [29]) that aims to reduce the gap between high level models and their actual implementations. Additionally, in association with the RTD-Finder tool it provides analyses allowing performance evaluation [15] as well as the actual analyses for the approach presented in Section 4. Note that the identity filter is the default filter that given a BIP model return the same BIP model.
3. The BIP back-end consists of code generator that generates the actual C++ corresponding to the actual BIP model yield by the middle-end. The engine based code generator produces simulation code that incorporates the BIP simulation engine. The Distributed code generator generates form Send/Receive models C++ code for distributed platform.
4. Verification of safety properties or properties allowing to tune a given system in order to obtain better performances are achieved using the RTD-Finder [27] and the SBIP tools [26].

The proposed method has been implemented in the distributed real-time middle-end filter of the BIP compiler. It aiming to generate information that could be used by the back-end during the code generation phase. The presented approach requires a substantial knowledge of the system model, since the satisfiability verification of $\widehat{Reach(LTS_g)} \wedge \phi$, needs a deep analysis of the system, in order to generate the predicates used in the latter. The implementation takes as input a BIP model and a horizon file specifying at least the lower bound horizon, that is, $h_{\min}$. Since we do not have a concrete execution platform, the choice of $h_{\min}$ was done relatively to the timing constraints of the verified model, that is, we chose values of $h_{\min}$ that are always smaller than upper bounds of any timing constraints appearing in the components of a given system. This choice was motivated by the fact that deploying a system with timing constraints being of the same order of magnitude than the communication delays of the target platform is unlikely to

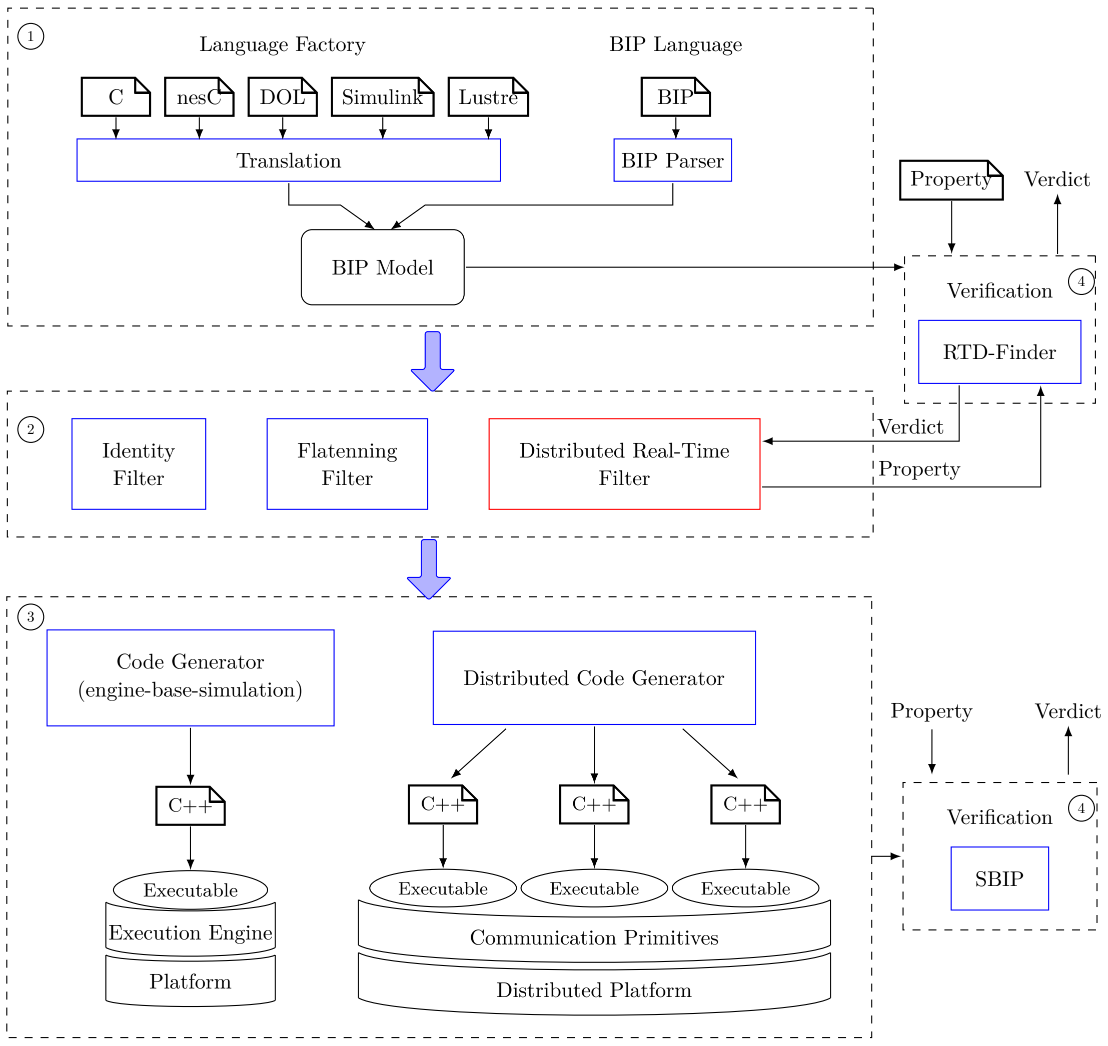


Figure 6 The BIP Toolbox.

happen. First, the front-end of the BIP compiler creates an abstract representation of the latter. Thereafter, the distributed real-time filter performs a model analysis in order to construct the predicates needed in ϕ, while keeping interactions upper horizons as free variables. In order to ease the verification process and remove the back and forth process between the predicates generation and their verification, we fully integrated the generation of the compositional invariants used by the RTD-Finder tool in the middle-end. Finally, a Yices [17] file including system invariants and the predicates approximating action-time-locks, is generated. The Yices solver checks then the satisfiability of $\widetilde{Reach}(LTS_g) \wedge \phi$. If the result is unsatisfiable, then planning does not miss the deadlines expressed by components time progress conditions. Otherwise, if the result is satisfiable, Yices generates a counter-example. Since for each interaction $h_{\min}$ is a fixed value, and due to the monotony of ϕ on $h_{\max}$, the generated counter-example is used to find the maximal value of $h_{\max}$ guaranteeing action-time-lock-freedom, and thus deadlock-freedom of the planning semantics. This part is however done manually, and based on binary search algorithm.

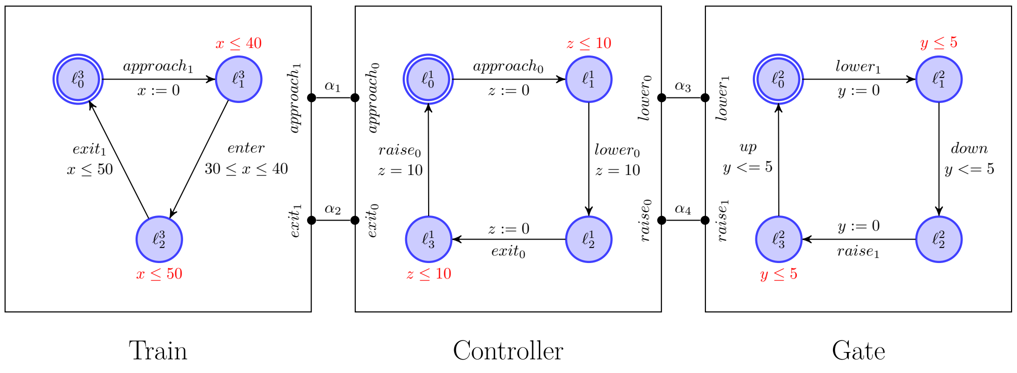


Figure 7 Train Gate Controller.

6.2 Benchmarks

For the experiments, we chose three additional benchmarks:

Train Gate Controller

The train gate controller [4] is a system composed of: a controller, a gate and a train. Figure 7 gives an overview of the system and its interactions: The train informs the controller about his position (w.r.t. to the crossing) through the interactions α_1 (approach) and α_2 (exit). On the other hand, the controller lowers (α_3) and raises (α_4) the gate whenever the train enters, respectively exits. Notice that actions $\{enter\}$ of the train, and $\{up, down\}$ of the gates are considered as singleton interactions.

Firewire

The IEEE 1394 root contention protocol (firewire) [14] is a standard protocol for interconnecting multimedia devices. It describes a serial bus used to transport digitized video and audio signals in a network of multimedia equipments. Among the different protocols used in this system, we put our interest in the leader election protocol called *tree identify protocol*. In this model, we consider two nodes (devices) and their respective channels. In order to elect a leader, each node sends a request via its respective channel asking its neighbor to be *a parent*. Once a neighbor receives a parent request, it either sends an acknowledgment or detects a *contention* in the case where it also sent a parent request. This contention is solved by assigning waiting times before the next send requests. Figure 8 depicts the model for the node component.

Gear Controller

The gear controller system describes the control system responsible for the gear change inside a vehicle. The used model encompasses formal models of the gear controller and its environment. The whole system includes five components: an interface, a controller, a clutch, an engine a gear-box and two global variables. In order to change the gear, the interface sends a signal to the controller. Consequently, the controller interacts with the engine, the clutch and the gear-box to achieve the gear change. The engine is responsible of either regulating the torque or synchronizing the speed. On the other hand, the gear-box sets the gear between some fixed bounds, whereas, the clutch is used whenever the engine is not able to function properly (under difficult driving conditions, for instance). The case study was initially designed by UPPAAL [25] and has been translated to BIP.

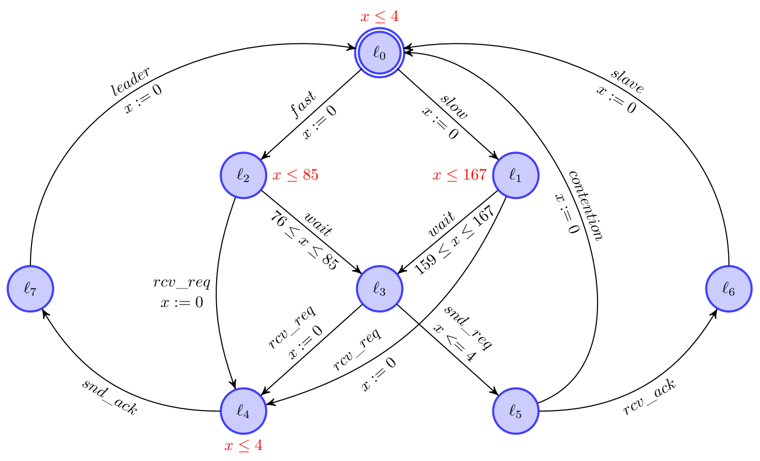


Figure 8 Timed Automaton for a Node.

Table 1 Detailed Results of the Task Manager Experiments.

$h_{\min}$	$h_{\max}(\alpha_1)$, $h_{\max}(\alpha_2)$	$h_{\max}(\alpha_3)$, $h_{\max}(\alpha_4)$	$h_{\max}(\alpha_5)$, $h_{\max}(\alpha_6)$	$h_{\max}(\alpha_7)$, $h_{\max}(\alpha_8)$
4	$+\infty$	$+\infty$	9	$+\infty$
3	$+\infty$	$+\infty$	8	$+\infty$
2	$+\infty$	$+\infty$	7	$+\infty$
1	$+\infty$	$+\infty$	6	$+\infty$

6.3 Results

Table 1 depicts the values $h_{\max}$ for each interaction of the running example, obtained while fixing $h_{\min}$. Notice that the symmetry of the system implies the same $h_{\max}$ for interactions $\alpha_i, \alpha_{i+1}, i \in \{1, 3, 5, 7\}$. By remarking that location ℓ_3^2 (resp. ℓ_3^3) has a time progress condition $x \leq 4$ (resp. $y \leq 4$), and by observing that the clock x is reset on the transition leading to this location, we can conclude that planning the system with $h_{\min} > 4$ will lead to an action-time-lock. Particularly, in Example 11, for $h_{\min} = 2$ interaction α_6 was planned with a horizon $\delta = 8$, and consequently, leads to a action-time-lock state. Our method detects such cases and thus, finds that the maximum horizon for this interaction is 7. Likewise, the $h_{\max}$ for interactions α_2, α_4 and α_8 (resp. α_1, α_3 and α_7) is found to be unbounded ($+\infty$).

Table 2 summarizes the experiments obtained on the benchmarks stated above, where n is the number of components, nb_{tpc} the number of time progress conditions that will be verified against action-time-lock freedom and max $h_{\min}$ the maximum value of $h_{\min}$ for which the system is action-time-lock-free in the planning semantics. Additionally, the column $h_{\max}$ indicates whether a restriction on the upper horizons is required to avoid deadlocks. Finally, t_{exec} gives an overview of the execution time including both the invariants generation and the verification time.

As shown in table 1, the task manager model has a maximal $h_{\min}$ value of 4 TU and requires a restriction on the upper horizons for interactions α_5 and α_6. In the same way, we found that the train gate controller, the firewire and the train gate controller models have respectively maximal $h_{\min}$ value of 4 TU, 5 TU and 130 TU. However, they do not require any restriction on the upper horizons values of their interactions.

Table 2 Experiments Results.

Model	n	nb_{tpc}	$\max h_{\min}$	$h_{\max}$	$t_{exec}(s)$
Task Manager	4	4	4	B	0.11
Train Gate Controller	3	6	4	$+\infty$	0.16
Firewire	4	10	5	$+\infty$	3.03
Gear Controller	5	19	130	$+\infty$	4.65

7 Related Work

Timed automata are high-level representations which are useful for modeling, specifying and analyzing system behavior [4]. They rely on mathematical abstractions such as real-valued clocks, instantaneous executions and communications, which are no longer valid at implementation level. Following model-based design approaches, a valid question is how to derive implementations from timed automata? This problem has already been addressed for centralized execution platforms. More specifically, Abdellatif et al. [1] shows how to take into account execution times and provides sufficient conditions for an implementation to be robust with respect to execution times. [3] and [32] studied the preservation of properties when introducing various sources of delays and digital (discrete) clocks in the implementations, to represent realistic executions on the hardware platform. [21, 22] takes a different approach than [3, 32] by trying to actively counteract the effect of delays in the generated code so as to meet properties.

In the context of distributed platforms, existing implementation frameworks [18, 9, 19, 11] for real-time applications are restricted to time-deterministic systems, which is a strictly less expressive than timed automata as explained in [1]. They also consider much simpler coordination mechanisms than multiparty interactions proposed in this paper. The generation of distributed implementations from components subject to multiparty (nary) interactions has been extensively studied in the untimed context [10, 8], and more recently for timed systems under the assumption of non-decreasing deadlines in [28]. The principle is to transform multiparty interactions into coordination mechanisms using simpler primitives such as asynchronous point-to-point messages, so that they can be mapped directly on communication mechanisms offered by distributed platforms. We contribute to this research field by considering in addition delays between the decision to execute an interaction and its actual execution. They are due to the transmission delays between the component responsible for such a decision and the components involved in the interaction, and may have a huge impact in the satisfaction of timing constraints in real-time systems. Indeed, such delays may introduce behavioral flaws (e.g. deadlocks) when dealing with arbitrary timing constraints (i.e. no restriction to the non-decreasing deadlines case), as shown in [16]. Our contribution consists in *(i)* the introduction of a semantics based on partial states of the system components and that includes a complete formalization of the effect of the delays in this context, and *(ii)* practical means for enforcing system correctness in their presence. This paper is an extension of the work presented in [16]. The semantics proposed in [16] allows to choose arbitrary (i.e. non-negative) delays between decisions and executions, which is not realistic. We improve [16] by restricting such delays with respect to lower bounds representing worst-case estimates of communication delays. We also updated accordingly the underlying semantics by restricting the progress of time as well as the sufficient conditions for system correctness presented in [16]. As explained in [16], they can be used in some cases to derive simple execution strategies achieving correctness. When our method is not applicable (i.e. the sufficient conditions cannot be met), an alternative method could be to use existing frameworks for control synthesis in timed automata. However, as we explained in this paper the problem addressed here cannot be fully expressed in these frameworks [6, 2], and had to be simplified by a discretization step. Moreover, when applicable our method remains faster than this alternative.

8 Conclusion and Future Work

We presented a local planning semantics for scheduling real-time systems in a distributed context. The proposed approach intends to mitigate the effect of communication delays through planning interactions ahead. A sufficient deadlock-freedom condition has been proved, a compositional verification method for checking action-time-lock-freedom was provided, and a simple execution strategy, based on restricting upper bounds horizons planning of interactions, has been presented. Additionally, a formalization of the planning problem as a real-time controller synthesis approach has been provided. This work shows how to express the planning semantics as timed game automata and highlights the encountered issues met during the formalization.

This approach opens a number of directions for future work. In case of action-time-locks of the planning semantics, a first idea consists to study their origins and derive a refinement method for models in order to take into account the communication delays. Another interesting direction is the characterization of the reachable states of the planning semantics. Instead of using an over-approximation of systems reachable states under the standard semantics, a more accurate approach could be to define a method for deriving invariants w.r.t the local planning semantics. Finally, an interesting idea is to investigate how scheduler(s) can benefit from the information provided by the presented method in order to optimize their scheduling policy.

References

1 Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In *Proceedings of the 10th International conference on Embedded software, EMSOFT 2010, Scottsdale, Arizona, USA, October 24-29, 2010*, pages 229–238, 2010. doi:10.1145/1879021.1879052.

2 Karine Altisen, Gregor Gößler, and Joseph Sifakis. Scheduler Modeling Based on the Controller Synthesis Paradigm. *Real-Time Systems*, 23(1-2):55–84, 2002. doi:10.1023/A:1015346419267.

3 Karine Altisen and Stavros Tripakis. Implementation of Timed Automata: An Issue of Semantics or Modeling? In *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, pages 273–288, 2005. doi:10.1007/11603009_21.

4 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

5 Ananda Basu, Laurent Mounier, Marc Poulhiès, Jacques Pulou, and Joseph Sifakis. Using BIP for Modeling and Verification of Networked Systems – A Case Study on TinyOS-based Networks. In *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), 12 - 14 July 2007, Cambridge, MA, USA*, pages 257–260, 2007. doi:10.1109/NCA.2007.52.

6 Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: Time for Playing Games! In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, pages 121–125, 2007. doi:10.1007/978-3-540-73368-3_14.

7 Johan Bengtsson and Wang Yi. On Clock Difference Constraints and Termination in Reachability Analysis of Timed Automata. In *Formal Methods and Software Engineering, 5th International Conference on Formal Engineering Methods, ICFEM 2003, Singapore, November 5-7, 2003, Proceedings*, pages 491–503, 2003. doi:10.1007/978-3-540-39893-6_28.

8 Saddek Bensalem, Marius Bozga, Jean Quilbeuf, and Joseph Sifakis. Optimized distributed implementation of multiparty interactions with observation. In *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions, AGERE! 2012, October 21-22, 2012, Tucson, Arizona, USA*, pages 71–82, 2012. doi:10.1145/2414639.2414649.

9 Gérard Berry and Georges Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992. doi:10.1016/0167-6423(92)90005-V.

10 Borzoo Bonakdarpour, Marius Bozga, and Jean Quilbeuf. Model-based implementation of distributed systems with priorities. *Design Autom. for Emb. Sys.*, 17(2):251–276, 2013. doi:10.1007/s10617-012-9091-0.

11 Sylvain Camier, Damien Chabrol, Vincent David, and Christophe Aussaguès. OASIS formal approach for distributed safety-critical real-time system design. In *ISoLA 2007, Workshop On Leveraging Applications of Formal Methods, Verification and Validation, Poitiers-Futuroscope, France, December 12-14, 2007*, pages 167–178, 2007. URL: http://editions-rnti.fr/?inprocid=1000543.

12 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *CONCUR 2005 - Concurrency Theory, 16th International Conference,*

CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings, pages 66–80, 2005. doi:10.1007/11539452_9.

13 Robert N. Charette. This Car Runs on Code. *IEEE Spectrum*, 2009.

14 Conrado Daws, Marta Z. Kwiatkowska, and Gethin Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *STTT*, 5(2-3):221–236, 2004. doi:10.1007/s10009-003-0118-5.

15 Mahieddine Dellabani, Jacques Combaz, Saddek Bensalem, and Marius Bozga. Knowledge Based Optimization for Distributed Real-Time Systems. In *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, pages 751–756, 2017. doi:10.1109/APSEC.2017.106.

16 Mahieddine Dellabani, Jacques Combaz, Marius Bozga, and Saddek Bensalem. Local Planning of Multiparty Interactions with Bounded Horizons. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*, pages 199–216, 2016. doi:10.1007/978-3-319-48989-6_13.

17 Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006.

18 Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and Verifying Real-Time Systems by Means of the Synchronous Data-Flow Language LUSTRE. *IEEE Trans. Software Eng.*, 18(9):785–793, 1992. doi:10.1109/32.159839.

19 Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

20 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Inf. Comput.*, 111(2):193–244, 1994. doi:10.1006/inco.1994.1045.

21 BaekGyu Kim, Lu Feng, Linh T. X. Phan, Oleg Sokolsky, and Insup Lee. Platform-specific timing verification framework in model-based implementation. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 235–240, 2015. URL: http://dl.acm.org/citation.cfm?id=2755804.

22 BaekGyu Kim, Lu Feng, Oleg Sokolsky, and Insup Lee. Platform-Specific Code Generation from Platform-Independent Timed Models. In *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*, pages 75–86, 2015. doi:10.1109/RTSS.2015.15.

23 Christos Kloukinas and Sergio Yovine. A model-based approach for multiple QoS in scheduling: from models to implementation. *Autom. Softw. Eng.*, 18(1):5–38, 2011. doi:10.1007/s10515-010-0074-8.

24 Hermann Kopetz. An Integrated Architecture for Dependable Embedded Systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 160–161, 2004. doi:10.1109/RELDIS.2004.1353016.

25 Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear Controller. In *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, pages 281–297, 1998. doi:10.1007/BFb0054178.

26 Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Jacques Combaz, Axel Legay, and Saddek Bensalem. SBIP 2.0: Statistical Model Checking Stochastic Real-time Systems. Technical Report TR-2018-5, Verimag Research Report, 2018.

27 Souha Ben Rayana, Marius Bozga, Saddek Bensalem, and Jacques Combaz. RTD-Finder: A Tool for Compositional Verification of Real-Time Component-Based Systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 394–406, 2016. doi:10.1007/978-3-662-49674-9_23.

28 Ahlem Triki. *Distributed Implementations of Timed Component-based Systems. (Implémentations distribuées des systèmes temps-réel à base de composants)*. PhD thesis, Grenoble Alpes University, France, 2015. URL: https://tel.archives-ouvertes.fr/tel-01169720.

29 Ahlem Triki, Borzoo Bonakdarpour, Jacques Combaz, and Saddek Bensalem. Automated Conflict-Free Concurrent Implementation of Timed Component-Based Models. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, pages 359–374, 2015. doi:10.1007/978-3-319-17524-9_25.

30 Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique. (The Formal Analysis of Timed Systems in Practice)*. PhD thesis, Joseph Fourier University, Grenoble, France, 1998. URL: https://tel.archives-ouvertes.fr/tel-00004907.

31 Stavros Tripakis. Verifying Progress in Timed Systems. In *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, pages 299–314, 1999. doi:10.1007/3-540-48778-6_18.

32 Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP Semantics: From Timed Models to Timed Implementations. In *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, pages 296–310, 2004. doi:10.1007/978-3-540-24743-2_20.