A Model for Industrial Real-Time Systems

Md Tawhid Bin Waez[1], Andrzej Wąsowski[2], Juergen Dingel[1], and Karen Rudie[1]

[1] Queen's University, Canada {waez@cs,dingel@cs,karen.rudie@}.queensu.ca
[2] IT University of Copenhagen, Denmark wasowski@itu.dk

Abstract. Introducing automated formal methods for large industrial real-time systems is an important research challenge. We propose timed process automata (TPA) for modeling and analysis of time-critical systems which can be open, hierarchical, and dynamic. The model offers two essential features for large industrial systems: (i) compositional modeling with reusable designs for different contexts, and (ii) an automated state-space reduction technique. Timed process automata model dynamic networks of continuous-time communicating control processes which can activate other processes. We show how to automatically establish safety and reachability properties of TPA by reduction to solving timed games. To mitigate the state-space explosion problem, an automated state-space reduction technique using compositional reasoning and aggressive abstractions is also proposed.

1 Introduction

This paper develops a model for the *automated analysis* of *safety* and *reachability* properties in large industrial *time-critical systems*. To fulfill industrial requirements, we consider time-critical systems that are open (communicate with external components), hierarchical (can be decomposed and recomposed into smaller control systems), and dynamic (the decomposition can change over time). In the paper, we use *real-time systems*, meaning time-critical systems that fulfill all these features. The model also facilitates compositional modeling and reusable designs for different contexts.

An *open system* continuously interacts with an unpredictable environment. A good example of time-critical open systems is a pacemaker, which continuously interacts with a heart, an uncontrolled environment. The pacemaker's performance crucially depends on the exact timing of an action performed either by the system or by the environment. The *theory of timed games* [1,2,3,4] is well-known in the research community for the analysis of time-critical open systems.

A *hierarchical system* is a hierarchical composition of smaller systems. An automotive system, developed by an *original equipment manufacturer (OEM)*, may be used in different models of cars. In this case, the system has a *controller* which helps the system adapt to different *environments* and cars. In other words, the system is an open system, which has two distinguished interacting segments: the controller and the environment. Typically, these systems consist of other smaller systems in a *hierarchical* structure. For instance, a system Actuator can

be a component of a larger system Position, while Position can be a component of another system Brake-by-Wire, and so on. Every component of a system has a specific set of tasks; for example, system Brake-by-Wire may use its component Position to perform some desired tasks in interaction with the environment, and Brake-by-Wire may also indirectly—through using Position—use its sub-component Actuator to perform some desired tasks in interaction with the environment.

A *dynamic system* is a hierarchical system whose components may change over time. Many hierarchical systems have dynamic characteristics, which are activating components only when needed. Dynamic behaviors are an important feature when resource constraints (such as limited memory) do not allow one to keep all the components active at the same time. Sometimes dynamic behaviors are inherent to the system. For example, we applied timed game theory in an industrial project to construct a fault-tolerant framework for a hierarchical open system that has a scheduler, a set of tasks, and a set of subtasks; only the scheduler is active in the initial system-state; subtasks are activated by their parent tasks, and the top level tasks are activated by their scheduler; thus the scheduler controls tasks, and a task controls its subtasks; due to the termination or the initialization of tasks (or subtasks) the structures of the processes may change; thus the system is a dynamic open system [5].

Timed automata (TA) [6,7] are desirable for the development of real-time systems because TA can model and analyze both discrete-time controllable behaviors of the system and continuous-time uncontrollable behaviors of the environment. Timed automata and their more than 80 variants [8] are mostly studied for the development of embedded systems, where behaviors of the components are known and the number of the components is static. As a result, modeling techniques, automated analyses, and other key issues of TA are typically addressed for *static closed systems*. The application domain of TA is growing [8]. In our two projects with General Motors (GM), we used different TA-based analyses to investigate the fault-tolerance of real-time systems, which are part of many large-scale safety-critical systems. During our industrial projects, we observed that continuous-time formal methods of TA may provide the most accurate analysis; however, TA are not suited for industrial real-time systems mainly because of poor *scalability*. Moreover, we found that TA have no structured support for modeling real-time systems, which may lead to cumbersome design details in a large-scale real-time system having several control hierarchies. The paper extends TA to achieve better modeling support and scalability for automated analysis of real-time systems.

We propose *timed process automata (TPA)*, a variant of TA, for the development of industrial real-time systems. The proposed variant provides compositional modeling (with reusable designs for different contexts) and automated analysis—a system needs to be modeled and analyzed using TPA only once when copies of it are used as independent systems or multiple components of a larger system or components of different larger systems or a combination of all previous scenarios. The contributions of this paper include:

1. Timed process automata, the first model that provides compositional modeling with reusable designs for dynamic hierarchical open time-critical systems.

2. Definition of a formal semantics for TPA.
3. An automated analysis for safety and reachability properties of TPA.
4. The first automated state-space reduction technique for time-critical systems, which can be dynamic, hierarchical, and open.

The rest of the paper can be divided into seven sections:

Section 2 Describes the motivation for the work. The motivation is based on the experience achieved from a couple of automotive industrial projects.
Section 3 Provides the required background to understand the paper.
Section 4 Presents the syntax (Sect. 4.1) and the semantics (Sect. 4.2) of TPA, which use start actions, finish actions, final locations, and channels to facilitate compositional modeling to reuse designs without manual alterations.
Section 5 Presents an automated analysis technique—based on timed games—for TPA. The analysis model of a timed process automaton T is constructed by composing a finite number of *timed I/O automata* (TIOA) [9,2,4], a variant of TA, to mimic the execution of T. The analysis model is constructed using an automated technique that allows the designer to avoid manual alteration techniques for different compositions. Other than the automated construction, the constructed analysis models essentially are TIOA models, whose state spaces are too large to analyze industrial real-time systems.
Section 6 Develops an automated state-space reduction technique that converts each callee process into a small automaton having only two locations and two edges, irrespective of the size of the callee. The technique uses structured construction of TPA, compositional reasoning, aggressive abstractions, and fewer synchronizations to ensure smaller state space.
Section 7 Discusses related work. It also classifies TPA depending on the classification of TA variants presented in a previous work [8].
Section 8 Concludes the paper.

2 Motivation

The first goal of the paper is to develop a real-time model, where a designer will not need to readjust a design for different compositions. The second and main goal is to allow automated analysis of the model for industrial systems.

Figure 1 presents an abstract Brake-by-Wire system modeled using TIOA, and the system is developed by an OEM. The model has seven automata representing different copies of only three elements: one copy of the *main thread* of Brake-by-Wire (the top automaton), two copies of the main thread of Position (the two automata in the middle), and four copies of Actuator system (the four automata in the bottom). Each Position system contains two *children* (Actuator systems) and its main thread that schedules the children, communicates with its parent (the main thread of Brake-by-Wire), and performs some other functions, which cannot be performed by the children. Similarly, the Brake-by-Wire system contains two children (Position systems) and its main thread that schedules the children and performs some other functions, which cannot be performed by the children.

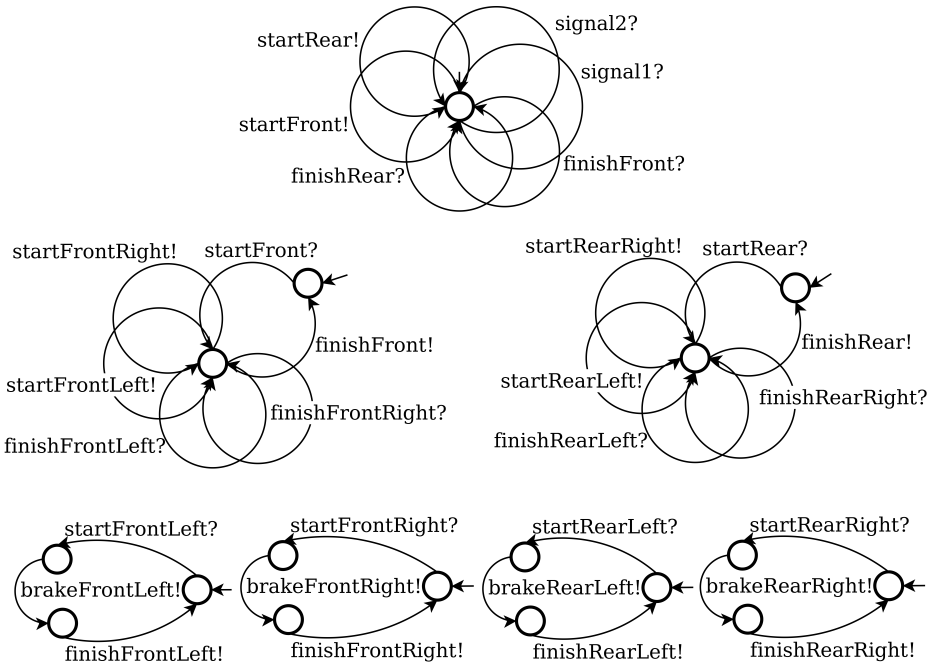

Fig. 1. An abstract Brake-by-Wire system modeled using standard TIOA

In this model, the main thread of Brake-by-Wire is the *root*, which does not have a parent. However, in the future a car manufacturer may include this Brake-by-Wire system in a car and then the main thread of Brake-by-Wire will no longer be the root. Then a central control system may be able to start the main thread of Brake-by-Wire. To analyze the new complex system, a designer will need to manually alter the model again by including *start* and *finish* actions (in the top automaton of Fig. 1). Let us assume a complex system contains N Break-by-Wire systems; to analyze this complex system, a designer will need to manually construct at least $N \times 7$ automata with a proportionally growing alphabet! Existing TA-based modeling techniques do not support compositional modeling with reusable designs for different contexts; that is, a design may need to be altered manually in every composition. All these ad hoc alterations may make a large industrial design incomprehensible and error-prone. Figure 2 contains the same Brake-by-Wire system of Fig. 1 modeled by using TPA. Timed process automata always model a system only once. For example, Fig. 2 presents only three TPA, which are equivalent to the seven automata of Fig. 1. Moreover, the number of copies and the root status of Break-by-Wire system has no impact on the new design.

To the best of our knowledge, no automated state-space reduction technique has been developed for the analysis of real-time systems. During our two projects with GM, we noticed that even a (practically) very small real-time system may have a state space too large for automated formal analysis because of hierarchy, dynamic behaviors, and time calculations. We overcame the scalability problem in one of the projects—construction of a fault-tolerance framework [5]—by developing a manual state-space reduction technique that applies aggressive

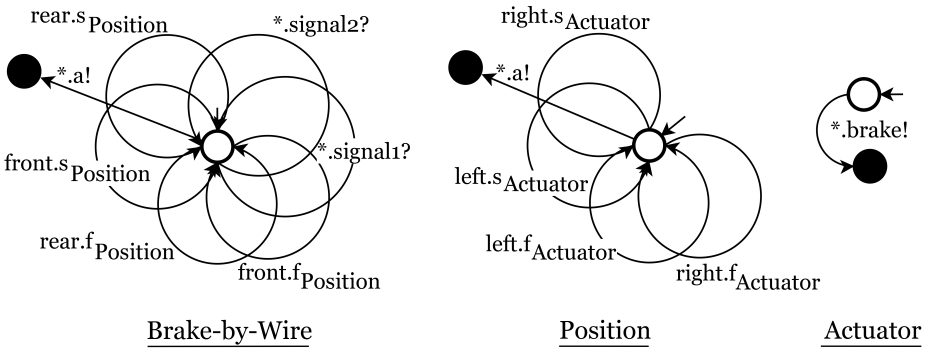

Fig. 2. The same Brake-by-Wire system of Fig. 1 is modeled using TPA

abstractions and uses fewer synchronizations. Applying this manual technique to a design of an industrial system is a challenging task. Moreover, the technique may not work for every real-time systems. A generalized automated reduction technique, therefore, is needed for analysis of large real-time systems, which is provided in this paper by presenting an automated reduction technique for TPA.

3 Background

The semantic construction of TA is expressed using semantics objects called *timed transition systems (TTS)* [10,4,7]. A *timed I/O automaton* [9,2,4] is a timed automaton which has an input alphabet along with a regular output alphabet. The controller plays controllable output transitions and the environment plays uncontrollable input transitions; thus TIOA are a natural model for timed games. Two TIOA are *composable* with each other if they don't have a common output action. The *composition of two well-formed TIOA* forms a larger timed I/O automaton [2,4]. The section defines TTS, TIOA, composition of TIOA, and all other terms required to understand the remaining paper.

Definition 1 *[10,4,7] A* timed transition system *is a tuple* $\mathcal{T} = (St, s_0, \Sigma, \dashrightarrow)$, *where* St *is an infinite set of* states, $s_0 \in St$ *is the* initial state, Σ *is an alphabet, and* $\dashrightarrow : St \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times St$ *is a* transition relation.

We use $d \in \mathbb{R}_{\geq 0}$ to denote delay. A TTS satisfies *time determinism* (i.e., whenever $s \overset{d}{\dashrightarrow} s'$ and $s \overset{d}{\dashrightarrow} s''$ then $s' = s''$ for all $s \in S$), *time reflexivity* (i.e., $s \overset{0}{\dashrightarrow} s$ for all $s \in S$), and *time additivity* (i.e., for all $s, s'' \in S$ and all $d_1, d_2 \in \mathbb{R}_{\geq 0}$ we have $s \overset{d_1+d_2}{\dashrightarrow} s''$ iff there exists an s' such that $s \overset{d_1}{\dashrightarrow} s'$ and $s' \overset{d_2}{\dashrightarrow} s''$). A *run* ρ of a TTS $\mathcal{T}$ from a state $s_1 \in St$ is a sequence $s_1 \overset{a_1}{\dashrightarrow} s_2 \overset{a_2}{\dashrightarrow} s_3 \cdots \overset{a_n}{\dashrightarrow} s_{n+1}$ such that for all $1 \leq m \leq n : s_m \overset{a_m}{\dashrightarrow} s_{m+1}$ with $a_m \in \Sigma \cup \mathbb{R}_{\geq 0}$. A state s is *reachable* in a transition system $\mathcal{T}$ if and only if there is a run $s_0 \overset{a_0}{\dashrightarrow} s_1 \overset{a_1}{\dashrightarrow} s_2 \cdots \overset{a_{n-1}}{\dashrightarrow} s_n$, where $s = s_n$. *Timed I/O transition systems (TIOTS)* are TTS with input and output modalities on transitions. Timed I/O transition systems are used to define semantics of TIOA.

A *clock* is a non-negative real variable. A *constraint* $\delta \in \mathcal{C}(X, V)$ over a set of clocks X and over a set of non-negative finitely bounded integer variables V is generated by the grammar $\delta ::= x_m \prec q \mid k \prec \alpha \mid x_m - x_n \prec q \mid \mathit{true} \mid \Phi \wedge \Phi$, where $q \in \mathbb{Q}_{\geq 0}$, $\alpha \in \mathbb{Z}_{\geq 0}$, $\{x_m, x_n\} \subseteq X$, $k \in V$ and $\prec \in \{<, \leq, >, \geq\}$. Consequently, the set of *clock constraints* $\mathcal{C}(X)$ is the set of constraints $\mathcal{C}(X, V)$, where $V = \emptyset$. Let $\Psi(V)$ be the set of assignments over the set of variables V.

Definition 2 *[9,2,4,7] A* timed I/O automaton *is a tuple $\mathcal{A} = (L, l_0, X, V, A, E, I)$, where L is a finite set of locations, $l_0 \in L$ is the initial location, X is a finite set of clocks, V is a finite set of non-negative finitely bounded integer variables, $A = A_\mathrm{i} \oplus A_\mathrm{o}$ is a finite set of actions, partitioned into input actions A_i and output actions A_o, $E \subseteq L \times A \times \Phi(X, V) \times \Psi(V) \times 2^X \times L$ is a set of edges, and $I : L \to \mathcal{C}(X)$ is a total mapping from locations to invariants.*

A *clock valuation* over X is a mapping $\mathbb{R}^X_{\geq 0} : X \to \mathbb{R}_{\geq 0}$. Given a clock valuation v and $d \in \mathbb{R}_{\geq 0}$, we write $v + d$ for the clock valuation in which for each clock $x \in X$ we have $(v + d)(x) = v(x) + d$. For $\lambda \subseteq X$, we write $v[x \mapsto 0]_{x \in \lambda}$ for a clock valuation agreeing with v on clocks in $X \setminus \lambda$, and giving 0 for clocks in λ. For $\phi \in \Phi(X, N)$ and $v \in \mathbb{R}^X_{\geq 0}$, we write $v, N \models \phi$ if v and N satisfy ϕ. Let $e = (l, a, \phi, \theta, \lambda, l')$ be an edge, then l is the source location, a is the action label, and l' is the target location of e; the constraint ϕ has to be satisfied during the traversal of e; the set of clocks $\lambda \in 2^X$ are reset to 0 and the set of non-negative finitely bounded integer variables are updated to θ whenever e is traversed.

Definition 3 *[2,4] Two timed I/O automata $\mathcal{A}^m = (L^m, l^m_0, X^m, N^m, A^m, E^m, I^m)$ and $\mathcal{A}^n = (L^n, l^n_0, X^n, N^n, A^n, E^n, I^n)$ are* composable *with each other when $A^m_o \cap A^n_o = \emptyset$, $X^m \cap X^n = \emptyset$, and $N^m \cap N^n = \emptyset$; when composable, their* composition *is a TIOA $\mathcal{A} = \mathcal{A}^m \| \mathcal{A}^n = (L^m \times L^n, (l^m_0, l^n_0), X^m \cup X^n, N^m \cup N^n, A, E, I)$, where $A = A_\mathrm{i} \cup A_\mathrm{o}$ with $A_\mathrm{o} = A^m_\mathrm{o} \cup A^n_\mathrm{o}$ and $A_\mathrm{i} = (A^m_\mathrm{i} \cup A^n_\mathrm{i}) \setminus A_\mathrm{o}$. The set of edges E contains:*

- $((l^m, l^n), a, \phi^m \wedge \phi^n, \lambda^m \cup \lambda^n, \theta^m \cup \theta^n, (l'^m, l'^n)) \in E$ *for each* $(l^m, a, \phi^m, \theta^m, \lambda^m, l'^m) \in E^m$ *and* $(l^n, a, \phi^n, \theta^n, \lambda^n, l'^n) \in E^n$ *if* $a \in \{A^m_\mathrm{i} \cap A^n_\mathrm{o}\} \cup \{A^m_\mathrm{o} \cap A^n_\mathrm{i}\}$
- $((l^m, l^n), a, \phi^m, \lambda^m, \theta^m, (l'^m, l^n)) \in E$ *for each* $(l^m, a, \phi^m, \lambda^m, \theta^m, l'^m) \in E^m$ *if* $a \notin A^n$
- $((l^m, l^n), a, \phi^n, \lambda^n, \theta^n, (l^m, l'^n)) \in E$ *for each* $(l^n, a, \phi^n, \lambda^n, \theta^n, l'^n) \in E^n$ *if* $a \notin A^m$

and the set of invariants I is constructed as follows: $I(l^m, l^n) = I^m(l^m) \wedge I^n(l^n)$

4 Processes

Timed process automata model processes, where each process is a real-time system. Every process hierarchically contains its active callee processes. Thus the control of a process is hierarchically shared with its active callee processes. The main thread of a process can activate callee processes via communication channels. An active process can receive any input in any state. An active callee process can deactivate itself in any state of the main thread of its caller process. An activated callee process dies within its worst-case execution time. This section presents the syntax and the semantics of TPA.

4.1 Timed Process Automata

Timed process automata are a variant of TIOA. Unlike a timed I/O automaton, a timed process automaton has a finite set of *start actions* A_s, a finite set of *finish actions* A_f, a final location l_f, and a finite set of *channels* C.

The set of *actions* $A = A_\mathrm{i} \oplus A_\mathrm{o} \oplus A_\mathrm{s} \oplus A_\mathrm{f}$ of a timed process automaton is a disjoint union of finite sets of input actions A_i, output actions A_o, start actions A_s, and finish actions A_f. For every set of actions A, there exists a bijective mapping between its start actions A_s and finish actions A_f in such a way that for each start action $\mathrm{s}_N \in A_\mathrm{s}$ there is exactly one finish action $\mathrm{f}_N \in A_\mathrm{f}$, and vice versa. These actions can be used for starting and terminating processes associated with N. We use s and f with the name N (of another timed process automaton) as a subscript index (e.g., s_N and f_N) to denote a start action and a finish action, respectively. We use the same subscript to indicate *paired* actions. We write a to denote an action in general. Processes synchronize via instantaneous channels. Each TPA uses the same designated symbols for its *public channel* ($*$) and *caller channel* ($\triangle$). We use c to denote a channel in general.

Definition 4 *A* timed process automaton *is a tuple* $T = (L, l_0, X, A, C, E, I, l_\mathrm{f})$*, where* L *is a finite set of locations,* $l_0 \in L$ *is the initial location,* X *is a finite set of clocks,* $A = A_\mathrm{i} \oplus A_\mathrm{o} \oplus A_\mathrm{s} \oplus A_\mathrm{f}$ *is a finite set of actions as described above,* C *is a finite set of channels,* $E \subseteq (L \times A \times C \setminus \{\triangle, *\} \times \Phi(X) \times 2^X \times L) \cup (L \times (A_\mathrm{i} \cup A_\mathrm{o}) \times \{\triangle, *\} \times \Phi(X) \times 2^X \times L)$ *is a set of edges,* $I : L \to \Phi(X)$ *is a total mapping from locations to invariants, and* $l_\mathrm{f} \in L$ *is a designated final location which does not have any outgoing edges to other locations and has the invariant* $I(l_\mathrm{f}) = \mathit{true}$*.*

Figure 2 presents TPA Brake-by-Wire, Position, and Actuator. In the figure, initial locations have a dangling incoming edge, final locations are filled with black, and TPA names are underlined. The final location l_f of a TPA may be unreachable from the initial location (and then l_f is not shown in the figure).

4.2 Process Executions

Every instance of a timed process automaton is a *process*. Two processes of the same timed process automaton represent two different copies of the same system. Every process has a unique *process identifier*. A *process* is a tuple $P = (\mathsf{id}(P), \mathsf{tpa}(P), \mathsf{channel}(P))$, where $\mathsf{id}(P)$[3] is the process identifier, timed process automaton $\mathsf{tpa}(P)$ defines the execution logic, and *caller channel* $\mathsf{channel}(P)$ is the private channel to communicate with the caller and the other processes which are started via the same channel. A process Q is a *callee* of P if P is the caller of Q. We use $\perp$ to denote the caller channel of the root process. Every process P of $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_\mathrm{f})$ has its own copy $P.c$ of channel $c \in C$. We write $P.c.a$ meaning that action a is performed via channel $P.c$.

At the same time, no two processes of the same timed process automaton can have the same caller channel. A process P, therefore, may have at most $|C| \times |A_\mathrm{s}|$

[3] To avoid clutter, we abuse notation by writing P instead of $\mathsf{id}(P)$.

active callee processes. For example, an instance of automaton Brake-by-Wire of Fig. 2 can activate at most two instances of automaton Position of Fig. 2 at the same time via two different channels front and rear, where the instance of Brake-by-Wire is the caller process of the two instances of Position, which are the callee processes of the instance of Brake-by-Wire. A *subprocess* is a callee or a callee of a subprocess, recursively. For example, every instance of Brake-by-Wire has six subprocesses: two instances of Position and four instances of automaton Actuator of Fig. 2. Every process hierarchically contains all of its subprocesses. Two processes are *siblings* if they have the same caller channel. The caller can use separate channels to differentiate control over different callees, even if they are processes of the same automaton.

A process P starts a process Q of an automaton $\mathsf{tpa}(Q)$ via channel $P.c$ by traversing an edge $e_1 = (\_, \mathrm{s}_{\mathsf{tpa}(Q)}, c, \_, \_, \_)$ labeled by a start action $\mathrm{s}_{\mathsf{tpa}(Q)}$ if there exists no active process of $\mathsf{tpa}(\mathrm{Q})$ with caller channel $P.c$; dually, P traverses an edge $e_2 = (\_, \mathrm{j}_{\mathsf{tpa}(Q)}, c, \_, \_, \_)$ labeled by the paired finish action $\mathrm{j}_{\mathsf{tpa}(Q)}$ whenever Q reaches its final state. No edge labeled by $\mathrm{j}_{\mathsf{tpa}(Q)}$ will ever be traversed if $\mathsf{tpa}(Q)$ is a *non-terminating timed process automaton*. Correspondingly, note that existing processes may start different processes of $\mathsf{tpa}(Q)$—but always with different process identifiers. However, only P listens to finish action $\mathrm{j}_{\mathsf{tpa}(Q)}$ via channel $\mathsf{channel}(Q)$. Process P traverses an edge $e = (\_, a, c, \_, \_, \_)$ when P receives (respectively, sends) an input (resp., output) a in channel $P.c$. Process P communicates with its callee Q via $\mathsf{channel}(Q)$ and with the environment via channel $P.*$.

We formalize the above mechanics of execution by first giving the semantics of the main thread of the process, ignoring its subprocesses in Def. 5 and then giving the semantics of the entire process in Def. 6. The standalone semantics of a process are essentially the same semantics as a standard TIOA [7,9,2,4]. The main difference is that states are decorated with process identifiers and edges with channel names to distinguish different instances of the same TPA in Def. 6. Also the caller channel $\triangle$ is instantiated for an actual parent process. The technical reason for this will become apparent in Def. 6.

Definition 5 *The* standalone semantics $\mathcal{S}[\![P]\!]$ *of a process* $P = (P, \mathsf{tpa}(P), \mathsf{channel}(P))$ *are a TIOTS* $\mathcal{S}[\![P]\!] = (L \times \mathbb{R}^X_{\geq 0} \times P, (l_0, \mathbf{0}, P), A^P, \twoheadrightarrow)$[4]*, where* $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_\mathrm{f})$, $\mathbf{0}$ *is a function mapping every clock to zero and* $\twoheadrightarrow \subseteq (L \times \mathbb{R}^X_{\geq 0} \times \{P\}) \times (A^P \cup \mathbb{R}_{\geq 0}) \times (L \times \mathbb{R}^X_{\geq 0} \times \{P\})$ *is the transition relation generated by the following rules:*

Action *For each clock valuation* $v \in \mathbb{R}^X_{\geq 0}$ *and each edge* $(l, a, c, \phi, \lambda, l') \in E$ *such that* $v \models \phi$, $v' = v[x \mapsto 0]_{x \in \lambda}$, *and* $v' \models I(l')$ *we have* $(l, v, P) \xrightarrow{P.c.a} (l', v', P)$ *if* $c \neq \triangle$, *otherwise* $(l, v, P) \xrightarrow{\mathsf{channel}(P).a} (l', v', P)$

Delay *For each clock valuation* $v \in \mathbb{R}^X_{\geq 0}$ *and for each delay* $d \in \mathbb{R}_{\geq 0}$ *such that* $(v + d) \models I(l)$ *we have* $(l, v, P) \overset{d}{\twoheadrightarrow} (l, v + d, P)$.

The transition system induced by the standalone semantics of a process is time deterministic, time reflexive, and time additive.

[4] A^P is the set of actions where action names are constructed using regular expression $(P\text{“.”}C \mid \mathsf{channel}(P))\text{“.”}A$.

Ground timed process automata are TPA that cannot perform a start or finish action ($A_{\mathrm{s}} \cup A_{\mathrm{f}} = \emptyset$). Automaton Actuator in Fig. 2, for instance, is a ground timed process automaton. *Compound timed process automata* are TPA that can perform a start or finish action ($A_{\mathrm{s}} \cup A_{\mathrm{f}} \neq \emptyset$). For example, Brake-by-Wire and Position in Fig. 2 are compound TPA. A *well-formed channel* cannot be used by two processes sharing an output action. Processes of a *well-formed timed process automaton* have only well-formed channels. Non-recursive TPA are defined inductively using the following rules: (i) every ground timed process automaton is a non-recursive timed process automaton, and (ii) a compound timed process automaton which performs only those start and finish actions whose subscripts are the names of some other existing non-recursive TPA is a non-recursive timed process automaton. All three TPA in Fig. 2, for example, are non-recursive TPA. A process of a non-recursive timed process automaton hierarchically contains only a finite number of subprocesses. The caller may activate an idle process, iteratively. Thus a process may activate a subprocess an arbitrary number of times. In this paper, we are only concerned with non-recursive well-formed TPA.

A *standalone final state* of a process P is (l_{f}, v, P), where v is any clock valuation. We use st^P, st^P_{i}, c^P, and st^P_{f} to denote a standalone state, the standalone initial state, the set of channels, and a standalone final state of process P, respectively. We say that a process P is A'-enabled for a channel $P.c$ if for every reachable standalone state st^P we have $st^P \xrightarrow{P.c.a}\!\!\!\!\rightarrow st'^P$ for some standalone state st'^P for each action $a \in A'$. We require that each process P is A_{i}-enabled (input enabled) for all channels of P, and A_{f}-enabled (finish enabled) for all channels of P other than channels $P.\triangle$ and $P.*$ to reflect the phenomenon that inputs from the environment and the deaths of callees are independent events, beyond the control of a process. We present the semantics of a process in the following:

Definition 6 *The* global operational semantics $\mathcal{G}[\![P]\!]$ *(semantics $[\![P]\!]$ for short) of a process $P = (P, \mathsf{tpa}(P), \bot)$ are a TIOTS $\mathcal{G}[\![P]\!] = (2^S, s_0, \mathbb{P} \times \mathbb{C} \times \mathbb{A}, \rightarrow)$, where S is the set of all the standalone states of all the processes in the universe, $\mathsf{tpa}(P) = (L, l_0, X, A, E, I, l_{\mathrm{f}})$, $s_0 = \{st^P_0\}$ is the initial state, $\mathbb{P}$ is the set of all the processes in the universe, $\mathbb{C}$ is the set of all the channels in the universe, $\mathbb{A}$ is the set of all the actions in the universe, and $\rightarrow \subseteq 2^S \times (\mathbb{P} \times \mathbb{C} \times \mathbb{A} \cup \mathbb{R}_{\geq 0}) \times 2^S$ is the transition relation generated by the following rules:*

$$\frac{st^Q \xrightarrow{Q.c.s_T}\!\!\!\!\rightarrow st'^Q \text{ and } c \notin \{\triangle, *\} \qquad \{st^W \in s \mid \mathsf{channel}(W) = Q.c \text{ and } \mathsf{tpa}(W) = T\} = \emptyset \qquad st^Q \in s \qquad (R, T, Q.c) \text{ is a freshly started process}}{s \xrightarrow{Q.c.s_T} s \setminus \{st^Q\} \cup \{st^R_0, st'^Q\}} \textsc{Start}$$

$$\frac{st^R_{\mathrm{f}}, st^Q \in s \text{ and } \mathsf{channel}(R) = Q.c \qquad \{st^U \in s \mid \mathsf{channel}(U) \in C^R\} = \emptyset \qquad st^Q \xrightarrow{Q.c.\mathrm{j}_{\mathsf{tpa}(R)}}\!\!\!\!\rightarrow st'^Q}{s \xrightarrow{Q.c.\mathrm{j}_{\mathsf{tpa}(R)}} s \setminus \{st^R_{\mathrm{f}}, st^Q\} \cup \{st'^Q\}} \textsc{Finish}$$

$$\frac{s' = \{st'^Q \mid st^Q \xrightarrow{d}\!\!\!\!\rightarrow st'^Q \text{ and } st^Q \in s \text{ and } (st^Q \neq st^Q_{\mathrm{f}} \text{ or } |s| = 1)\} \qquad |s| = |s'|}{s \xrightarrow{d} s'} \textsc{Delay}$$

$$\frac{a \notin \bigcup_{st^Q \in s} A_{\mathrm{o}}^{\mathsf{tpa}(Q)} \qquad s' = \{st^Q \in s \mid st^Q \overset{Q.*.a}{\longrightarrow\!\!\!\!\!\rightarrow} st'^Q\}}{s \xrightarrow{a} s \setminus s' \cup \{st'^Q \mid st^Q \overset{Q.*.a}{\longrightarrow\!\!\!\!\!\rightarrow} st'^Q \text{ and } st^Q \in s\}} \textsc{Input}$$

$$\frac{\begin{array}{c} st^Q \overset{W.c.a}{\longrightarrow\!\!\!\!\!\rightarrow} st'^Q \text{ and } a \in A_{\mathrm{o}}^Q \text{ and } st^Q \in s \\ s' = \{st^R \in s \mid st^R \overset{W.c.a}{\longrightarrow\!\!\!\!\!\rightarrow} st'^R \text{ and } W.c \text{ is a channel}\} \end{array}}{s \xrightarrow{Q.c.a} s \setminus s' \cup \{st'^R \mid st^R \overset{W.c.a}{\longrightarrow\!\!\!\!\!\rightarrow} st'^R \text{ and } st^R \in s\}} \textsc{Output}$$

A *global state* is a set which holds standalone states of all active processes. The Start rule states that the initial standalone state of a freshly started callee is added to the global state whenever the corresponding start action is performed by its caller. The rule also states that no two active processes can have the same timed process automaton and the same caller channel. The Finish rule prescribes that the standalone-final state of a callee is removed from the global state and the caller executes the corresponding finish action whenever that callee is in the standalone-final state and no standalone state of its subprocesses is in global state. Thus the rule defines *global-final state* (*final state* for short) of a process: a process is in its the final state when the process is in its final location and the process has no active subprocess. The Delay rule declares that globally a process can delay if that process and all of its active subprocesses can delay in their respective standalone semantics. Every subprocess is a part of the root process and thus if a subprocess is performing an action (or not idle) then the root process is also not idle. The rule also says that a process cannot delay if that process or any of its subprocess is in its global final state. That means a process finishes as soon as it reaches its final state. The Input rule states that a process receives an input from the environment via channel $id.*$. Rule Output declares a process send an output via channel $id.c$ to others who share $id.c$. It follows from the properties of the standalone semantics that the transition system induced by Def. 6 is time deterministic, time reflexive, and time additive. The process semantics, therefore, defines a well-formed TIOTS. This allows us to use TA as a basis for analyzing TPA. A *local run* of the main thread of a process P is a standalone run of P for which there exists a global run of P such that every transition of that standalone run occurs in that global run. The *local behavior* of the main thread of P consists of all of its local runs.

5 Analysis

We are interested in *safety* and *reachability* properties of real-time systems. This section explains how such analyses can be performed using the theory of timed games. A standard timed I/O automaton can be viewed as a concurrent two-player timed game, in which the players decide both which action to play, and when to play it. The input player represents the environment, and the output player represents the system itself. Similarly, the main thread of a process acts as a concurrent two-player timed game: the environment plays input transitions and finish transitions, and the main thread of the process plays output transitions and

start transitions. Let's consider interactions of a process defined in the previous section. A process controls its output and start transitions. After starting a callee, the main thread of the caller knows that the paired finish action will arrive within the worst-case execution time of the associated callee. However, the main thread does not have any control on the exact arrival time of a finish action. Finish transitions along with input transitions are uncontrollable. The environment of the main thread of a process consists of all the connected processes (such as caller, siblings, and subprocesses) and all unconnected entities.

A global state of a process is safe if and only if all of the standalone states which it contains are safe locations. A *safety property* asserts that the system remains inside a set of global-safe states regardless of what the environment does. We are interested in *Safety Property I: Given a process P and a set of unsafe locations L_U of P, can the controller avoid L_U in P regardless of what the environment does?* A global state of a process is a target state if and only if at least one of its standalone states contains a target location. A *reachability property* asserts that the system reaches any of the global-target states regardless of what the environment does. We are interested in *Reachability Property I: Given a process P and a set of target locations L_T of P, can the controller reach a location of L_T in P regardless of what the environment does?*

The monolithic analysis constructs a static network of automata to represent all possible global executions by mimicking the hierarchical call tree of the analyzed process. It simulates a process execution by changing states of pre-allocated TIOA which fall into two groups: a *root automaton* to simulate the local behaviors of the main thread of the root process and a finite set of *standalone automata* to simulate the local behaviors of the main threads of the subprocesses.

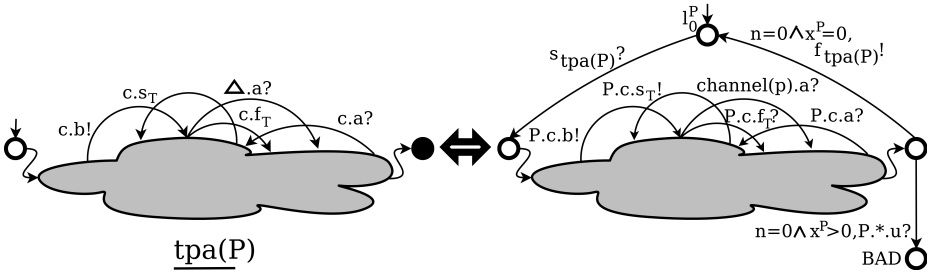

Fig. 3. A generalized view of standalone automata construction

Standalone Automata We construct a standalone automaton for each subprocess to simulate the main thread of that process. To construct a standalone automaton, we prefix the timed process automaton with a simulated start action and suffix it with a simulated finish action. We use non-negative finitely bounded integer variables[5] in standalone automata to count the number of active callees, in order

[5] The use of non-negative finitely bounded integer variables can be avoided if a more cumbersome encoding is used.

to detect termination. We rename actions (e.g., a) of processes uniformly to encode channel names (e.g., $P.c$) in action names (e.g., $P.c.a$) of standalone automata; because standard TIOA do not support private channels. A standalone automaton includes all the locations and slightly altered edges of the corresponding timed process automaton. Moreover, each standalone automaton has two additional locations: a new initial location l_0^{id} to receive (resp., send) a start (resp., finish) message from (resp., to) the caller, and a new unsafe location BAD to prevent the automaton from waiting in final states instead of finishing. Every start (resp., finish) increments (resp., decrements) a counter variable n. The automaton represents finishing of the process in the final location when $n = 0$. Formally, the standalone automaton of process P is $\mathsf{standalone}(P) = (L \cup \{l_0^{\mathrm{P}}, BAD\}, l_0^{\mathrm{P}}, X \cup \{x^{\mathrm{P}}\}, \{n\}, A^{\mathrm{P}}, E^{\mathrm{P}}, I^{\mathrm{P}})$, where $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_{\mathrm{s}})$, l_0^{P} and BAD are two newly added locations, x^{P} is a newly added clock, n is a non-negative finitely bounded integer variable with the initial value 0, $A_{\mathrm{o}}^{\mathrm{P}} = A'_{\mathrm{o}} \cup A'_{\mathrm{s}} \cup \{\mathsf{channel}(P).\mathrm{f}_{\mathsf{tpa}(P)}\}$ and $A_{\mathrm{i}}^{\mathrm{P}} = A'_{\mathrm{i}} \cup A'_{\mathrm{f}} \cup \{\mathsf{channel}(P).\mathrm{s}_{\mathsf{tpa}(P)}, P.*.u\}$ such that $A'_m = \{\mathsf{channel}(P).a \mid a \in A_m\} \cup \{P.c.a \mid a \in A_m \text{ and } c \in C \setminus \{\triangle\}\}$ where $m \in \{\mathrm{o}, \mathrm{s}, \mathrm{i}, \mathrm{f}\}$ and newly added actions are $\mathsf{channel}(P).\mathrm{s}_{\mathsf{tpa}(P)}$, $\mathsf{channel}(P).\mathrm{f}_{\mathsf{tpa}(P)}$, and $P.*.u$. The set of edges E^{P} contains

- Converted edges that do not communicate via caller channel $\triangle$:
  - An edge $(l, P.c.a, \phi, \xi, \lambda \cup \lambda', l') \in E^{\mathrm{P}}$ for each edge $(l, a, c, \phi, \lambda, l') \in E$, where $c \in C \setminus \{\triangle\}$, the integer assignment is empty $\xi = \emptyset$ when $a \in A_{\mathrm{o}} \cup A_{\mathrm{i}}$, $\xi = \{n--\}$ when $a \in A_{\mathrm{f}}$, and $\xi = \{n++\}$ when $a \in A_{\mathrm{s}}$
- Converted edges that communicate via caller channel $\triangle$:
  - An edge $(l, \mathsf{channel}(P).a, \phi, \emptyset, \lambda \cup \lambda', l') \in E^{\mathrm{P}}$ for each edge $(l, a, \triangle, \phi, \lambda, l') \in E$
- Additional new edges that simulate activation and deactivation:
  - Three more edges $(l_0^{\mathrm{P}}, \mathsf{channel}(P).\mathrm{s}_{\mathsf{tpa}(P)}, \emptyset, \emptyset, X, l_0), (l_{\mathrm{f}}, \mathsf{channel}(P).\mathrm{f}_{\mathsf{tpa}(P)}, n = 0 \wedge x^P = 0, \emptyset, \emptyset, l_0^{\mathrm{P}}), (l_{\mathrm{f}}, P.*.u, n = 0 \wedge x^P > 0, \emptyset, \emptyset, BAD)$ are in E^{P}

$\lambda' = \emptyset$ when $l' \neq l_{\mathrm{f}}$, otherwise $\lambda' = \{x^{\mathrm{P}}\}$. The invariant function I^{P} maps each location $l \in L$ to $I(l)$ and maps each location $l \in \{l_0^{\mathrm{P}}, BAD\}$ to $true$. The standalone semantics of automaton $\mathsf{tpa}(P)$ and the semantics of standalone automaton $\mathsf{standalone}(P)$ are essentially the same.

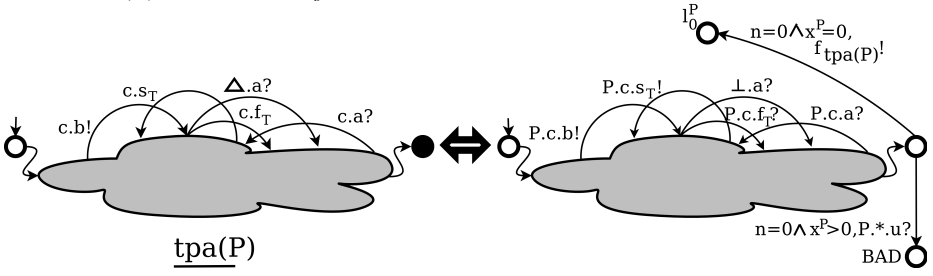

Fig. 4. A generalized view of root automata construction

Root Automata To analyze a timed process automaton $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_{\mathrm{f}})$, we construct the root automaton $\mathsf{root}(\mathrm{P})$ of process P. Standalone automaton $\mathsf{standalone}(P)$ is slightly different from $\mathsf{root}(\mathrm{P})$. The differences are: (i) the caller channel is always $\perp$, (ii) the initial location of root automaton $\mathsf{root}(\mathrm{P})$ is the

location l_0, which is also the initial location of $\mathsf{tpa}(P)$, and (iii) root automaton does not have edge $(l_0^{\mathrm{P}}, \bot.\mathrm{s}_{\mathsf{tpa}(P)}, \emptyset, \emptyset, X, l_0)$, which simulates activation of P.

Monolithic Analysis Model The monolithic analysis model of a ground timed processes automaton (such as Actuator) is its root automaton. We construct the monolithic analysis model of automaton $\mathsf{tpa}(P)$ in the following iterative manner:

First Step: We construct the root automaton $\mathsf{root}(\mathrm{P})$.
Iterative Step: We construct a standalone automaton for each triple (Q, s_T, c), where Q is process for which we have constructed a standalone automaton or the root automaton, $\mathsf{tpa}(Q) = (L, l_0, X, A, C, E, I, l_{\mathrm{f}})$, $c \in C \setminus \{\triangle, *\}$, $\mathrm{s}_T \in A_{\mathrm{s}}$, and $(\_, \mathrm{s}_T, c, \_, \_, \_) \in E$.

Figures 3–4 present a generalized view of the standalone and root automata constructions (a technical report [11] presents monolithic analysis models of processes of TPA Actuator, Position, and Brake-by-Wire). The monolithic analysis model constructs a parallel composition of all the TIOA constructed above. The construction is finite, and the composition is a timed I/O automaton, because we consider only non-recursive well-formed TPA. The created composition is timed-bisimilar to the global semantics (modulo hiding the special actions and renaming the others). Executions of this composition, when projected on the original alphabet, are identical to the executions of the global semantics. Thus the composition has the same properties. We convert Safety Property I to *Safety Property II: Given a process P and a set of unsafe locations L_U of P, can the controller avoid L_U and all the BAD locations in the analysis model regardless of what the environment does?* We also convert Reachability Property I to *Reachability Property II: Given a process P and a set of target locations L_T of P, can the controller reach a location of L_T in the analysis model avoiding all the BAD locations regardless of what the environment does?* Avoiding all the newly added *BAD* locations in the analysis model ensures that each caller process performs the corresponding finish action as soon as the callee finishes. Therefore, if a Safety Property I (resp., Reachability Property I) holds for a process then its corresponding Safety Property II (resp., Reachability Property II) also holds, and vice versa.

6 State-Space Reduction

We introduce an automated state-space reduction technique for TPA to counteract state-space explosion. The technique relies on compositional reasoning, aggressive abstractions, and reducing process synchronizations. In the monolithic analysis of Sect. 5, a callee can be represented by an arbitrary number of standalone automata and each of these automata can be arbitrarily large. The compositional reasoning replaces hierarchical trees of standalone automata representing subprocesses with simple abstractions (Fig. 5)—so called *duration automata*.

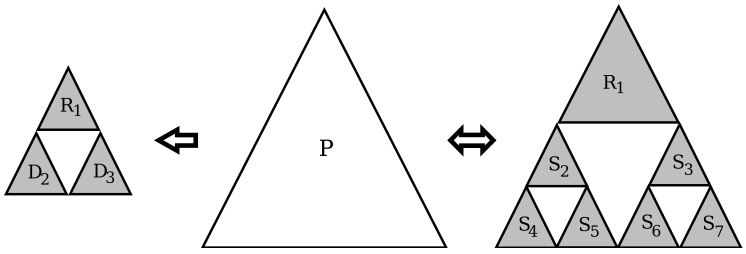

Fig. 5. A compositional (sound) analysis model on the left and a monolithic (sound and complete) analysis model on the right of automaton Brake-by-Wire, where P is a process of the automaton, R_1 is the root automaton, S_1–S_7 are standalone automata, and D_1–D_2 are duration automata

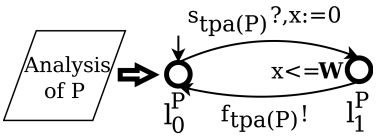

Fig. 6. A generalized view of duration automata construction

Duration Automata A duration automaton (Fig. 6) is timed I/O automaton with only two locations: the initial location (l_0^P) and the active location (l_1^P). A duration automaton of an analyzed process abstracts all the information of global executions of the process other than its *worst-case execution time (WCET)*. It can capture safety and reachability properties of interest. The *minimal-time safe reachability* of a target location is the *minimal-time reachability* [12,13] for which the controller has a winning strategy to reach that target location by avoiding unsafe states. We assume that the WCET $\mathbf{W}$ of a process P is the minimal-time safe reachability time to reach location l_0^{P} of automaton $\mathsf{root}(P)$ in the analysis model of P. This is a known technique to limit the WCET of a controller [14,15]. The WCET of P is unknown ($\mathbf{W} = \infty$) when there is no winning strategy for the minimal-time safe reachability to reach location l_0^{P} of $\mathsf{root}(P)$. The duration automaton of process P is $\mathsf{duration}(P) = (\{l_0^{\mathrm{P}}, l_1^{\mathrm{P}}\}, l_0^{\mathrm{P}}, \{x^{\mathrm{P}}\}, \emptyset, A^{\mathrm{P}}, E^{\mathrm{P}}, I^{\mathrm{P}})$, where $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_{\mathrm{f}})$, $A_{\mathrm{i}}^{\mathrm{P}} = \{\mathsf{channel}(P).\mathrm{s}_{\mathsf{tpa}(P)}\}$, $A_{\mathrm{o}}^{\mathrm{P}} = \{\mathsf{channel}(P).\mathrm{j}_{\mathsf{tpa}(P)}\}$, the set of edges $E^{\mathrm{P}} = \{(l_0^{\mathrm{P}}, \mathsf{channel}(P).\mathrm{s}_{\mathsf{tpa}(P)}, \emptyset, \emptyset, \{x^{\mathrm{P}}\}, l_1^{\mathrm{P}}), (l_1^{\mathrm{P}}, \mathsf{channel}(P).\mathrm{j}_{\mathsf{tpa}(P)}, \emptyset, \emptyset, \emptyset, l_0^{\mathrm{P}})\}$, invariant I^{P} maps location l_0^{P} to *true*, and I^{P} maps location l_1^{P} to $x^{\mathrm{P}} \leq \mathbf{W}$.

Compositional Analysis Model We construct the compositional analysis model in a bottom-up manner: analysis of a compound process is performed only after analyzing all its callees. Like the monolithic analysis, the compositional analysis model of a ground timed process automaton $\mathsf{tpa}(Q)$ (such as Actuator) is a root automaton of process Q. That TIOA is analyzed to construct a duration automaton of Q. For a compound process P, we analyze automaton $\mathsf{root}(P)$ in the context of the duration automata of its callees (instead of the entire hierarchical structure of subprocesses). We construct the compositional analysis model of a timed process automaton $\mathsf{tpa}(P)$ in the following manner:

First Step: We construct the root automaton root(P).
Second Step: We construct a duration automaton for each triple (P, s_T, c), where $\mathsf{tpa}(P) = (L, l_0, X, A, C, E, I, l_{\mathrm{f}})$, $c \in C \setminus \{\triangle, *\}$, $\mathrm{s}_T \in A_{\mathrm{s}}$, and $(\_, \mathrm{s}_T, c, \_, \_, \_) \in E$.

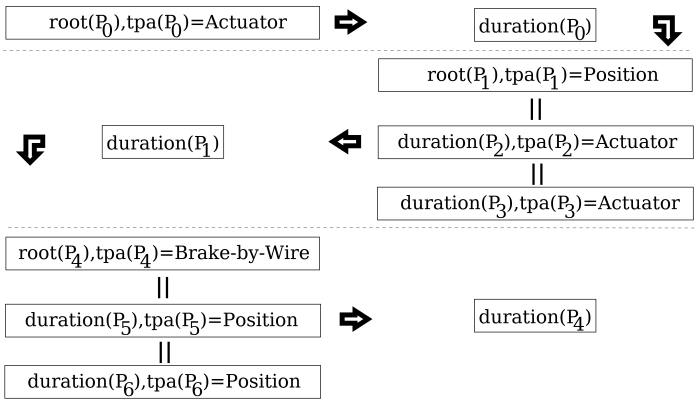

Fig. 7. Steps of the compositional analysis of automaton Brake-by-Wire

Figure 7 presents the compositional analysis procedure of Brake-by-Wire (the detailed models are presented in [11]). The compositional model construction procedure terminates, and the composition of all the above TIOA is a timed I/O automaton, because we consider only non-recursive well-formed TPA. The duration automaton of a process can capture safety properties: if a process has a winning strategy for a safety game, then all locations of its duration automaton are considered safe; otherwise, the active location (l_1^{id}) of the duration automaton is added to the set of unsafe locations L_U. Now this duration automaton can be used as a sound context to analyze the caller automaton for safety. A safety property holds for a compound process when the main thread of the process preserves the property locally and allows the activation of a callee only if that callee also preserves the property. Duration automata can also capture reachability properties: if a process has a winning strategy for a reachability game then the active location (l_1^{id}) of the duration automaton is added to the set of target locations L_T; otherwise, no target location is specified for this callee. This duration automaton can be used as a sound context to analyze the caller automaton for reachability. A reachability property holds for a compound process when the main thread of the process can reach the target locally or can activate a callee where the property holds. Like the monolithic analysis, the compositional analysis is performed for Safety Property II and Reachability Property II. The compositional analysis is sound: if a safety or reachability property holds in compositional analysis then it holds in the global semantics. A duration automaton does not contain any input and output actions of its process. Hence, the root automaton in a compositional model does not synchronize with the input and output actions of its callees—instead the automaton synchronizes for those actions with the environment. The duration automaton was created under the assumption that inputs are uncontrollable, so ignoring synchronization with inputs is sound. Similarly, it is sound to open the inputs of the root automaton from a callee, as they will be treated as open actions, so will be analyzed in a more "hostile" environment than before the abstraction. Therefore, if a property holds in the compositional analysis then it also holds for the monolithic analysis.

Scalability In all the steps of Fig. 7, the largest composition contains only three automata, and except for the root automaton all are tiny duration automata. Monolithic analysis model of Brake-by-Wire is a composition of seven automata (see [11]). A duration automaton always has a small constant size (modulo the size of the WCET constant), and so its state space is very simple (actually the discrete state space is independent of the input model). We applied our approach to examples like the one presented in our previous work [5]. First, we modeled that problem with standard TIOA using shared variables. The timed games solver Uppaal Tiga [16] produced a large winning strategy (290 MB) for a safety objective for a configuration (C1 of [5])—a combination of different system parameters—in the TIOA model within 94 seconds[6]. After that, we modeled the same system with TPA, and applied the state-space reduction technique. The same solver for the same configuration produced a much smaller winning strategy (100 KB) for the same objective in our compositional model within 0.3 seconds. Experiments for different configurations for the same system (of [5]) revealed that speed up of two orders of magnitude is possible with the compositional technique, while maintaining enough precision to obtain useful strategies for realistic scheduling problems. The size of composition in the monolithic analysis is exponential in the depth of the hierarchy, due to a product construction (it is also linear in the multiplication of sizes of all included standalone automata). In the compositional analysis, the depth of the hierarchy is constant (only two layers) and we only take a product of one root automaton with several constant size duration automata; this explains why the practically obtained speed ups are so dramatic. The efficiency gains are primarily due to the coarse abstraction of safety and reachability properties of an arbitrarily large callee into a tiny duration automaton. Abstraction and compositional reasoning together might provide similar speed ups for TIOA [5]; however, the restrictions that TPA impose on models allow one to automate the procedure.

7 Related Work

Classical TA [6,7] and timed I/O automata [2,4] have explicit modeling support only for static non-hierarchical structures. In 2011, we identified and classified eighty variants of TA into eleven classes in a survey [8] and there may be many more. Timed process automata fall in the class of *TA with resources* [8] because of their ability to model dynamic behaviors, which is required when resource constraints do not permit one to activate all the components at the same time. More precisely, the model is a direct generalization of *task automata* [17], *dynamic networks of TA* [18], and *callable timed automata* [19]. These three variants model only closed systems, while TPA can model both closed and open systems. Task automata model only two layers (a scheduler and its tasks) of hierarchy, while TPA, dynamic networks of TA [18], and callable timed automata are able to model any numbers of hierarchies. Unlike TPA, none of them supports

[6] All the analyses were performed by Uppaal Tiga-0.17 on a PC with an Intel Core i3 CPU at 2.4 GHz, 4 GB of RAM, and running 64-bit Windows 7.

private communication, provides compositional modeling with reusable designs for different contexts, or supports automated state-space reduction technique.

Dynamic networks of continuous-time automata have also been studied in the context of hybrid automata [20,21]. These works model physical environments using differential equations, which restrict the kinds of environments that can be described. In practice, large differential equations make analyses unmanageable, or can only give statistical guarantees [21]. These works focus on system dynamics, and do not support private communication. Timed process automata can be considered as a member of the class of *TA with succinctness* [8] because they hide many design details from the designers to achieve succinctness (like *TA variants with urgency* [22,23,8]). Timed process automata are also timed game automata [1,3,2,4] because the new variant uses timed games for analysis.

8 Conclusion

We have presented timed process automata that captures dynamic activation and deactivation of continuous-time control processes and private communication among the active processes. We have provided a safety and reachability analysis technique for non-recursive well-formed timed process automata. We have also designed an abstraction- and compositional reasoning-based state-space reduction technique for automated analysis of large industrial systems. Our analysis techniques can be applied in practice using any standard timed games solver such as Uppaal Tiga [16] and Synthia [24].

To the best of our knowledge, no prior work on dynamic network of timed automata considered private communication or open systems. This is also the first work that provides two important features for industrial time-critical dynamic open systems development: (i) compositional modeling with reusable designs for different contexts and (ii) automated state-space reduction technique.

It would be interesting to consider a model transformation from subset of *real-time π-calculus* [25,26] to TPA. This transformation might enable controllability analysis of π-calculus for open systems. The converse reduction from TPA to real-time π-calculus could also give several advantages: understanding TPA semantics in terms of the well-established π-calculus formalism, access to tools developed for real-time π-calculus [25], which might permit the analysis of recursive processes; it would also give a familiar automata-like syntax to π-calculus formalisms. It would also be relevant to minimize the number of subprocesses in controller synthesis. One may consider synthesis under this objective in the future, possibly by reduction to *priced/weighted timed automata* [27,28].

References

1. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: STACS. (1995)
2. Alfaro, L.d., Henzinger, T.A., Stoelinga, M.: Timed interfaces. EMSOFT (2002)

3. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: CONCUR. (2003)
4. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC. (2010)
5. Waez, M.T.B., Wąsowski, A., Dingel, J., Rudie, K.: Synthesis of a reconfiguration service for mixed-criticality multi-core systems an experience report (to appear). In: FACS. (2014)
6. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: ICALP. (1990)
7. Alur, R., Dill, D.L.: A theory of timed automata. TCS **126** (1994)
8. Waez, M.T.B., Dingel, J., Rudie, K.: A survey of timed automata for the development of real-time systems. CSR (2013)
9. Kaynar, D.K., Lynch, N.A., Segala, R., Vaandrager, F.W.: The Theory of Timed I/O Automata. (2006)
10. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed transition systems. In: REX Workshop. (1992)
11. Waez, M.T.B., Wąsowski, A., Dingel, J., Rudie, K.: A model for industrial real-time systems. Technical Report 2014-622, Queen's University, ON (2014)
12. Brihaye, T., Henzinger, T.A., Prabhu, V.S., Raskin, J.F.: Minimum-time reachability in timed games. In: ICALP. (2007)
13. Jurdziński, M., Trivedi, A.: Reachability-time games on timed automata. In: ICALP. (2007)
14. Cassez, F.: Timed games for computing WCET for pipelined processors with caches. In: ACSD. (2011)
15. Gustavsson, A., Ermedahl, A., Lisper, B., Pettersson, P.: Towards WCET analysis of multicore architectures using UPPAAL. In: WCET. (2010)
16. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Didier, L.: UPPAAL-Tiga: time for playing games! In: CAV. (2007)
17. Fersman, E., Krčál, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation (2007)
18. Campana, S., Spalazzi, L., Spegni, F.: Dynamic networks of timed automata for collaborative systems: A network monitoring case study. In: ISCTS. (2010)
19. Boudjadar, A., Vaandrager, F., Bodeveix, J.P., Filali, M.: Extending UPPAAL for the modeling and verification of dynamic real-time systems. In: FSE. (2013)
20. Göllü, A., Varaiya, P.: A dynamic network of hybrid automata. In: AIS. (1994)
21. David, A., Larsen, K.G., Legay, A., Poulsen, D.B.: Statistical model checking of dynamic networks of stochastic hybrid automata. In: AVoCS. (2013)
22. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: COMPOS. (1998)
23. Barbuti, R., Tesei, L.: Timed automata with urgent transitions. Acta Informatica (2004)
24. Ehlers, R., Mattmüller, R., Peter, H.J.: Synthia: Verification and synthesis for timed automata. In: CAV. (2011)
25. Posse, E., Dingel, J.: Theory and implementation of a real-time extension to the π-calculus. In: FMOODS/FORTE. (2010)
26. Barakat, K., Kowalewski, S., Noll, T.: A native approach to modeling timed behavior in the pi-calculus. In Margaria, T., Qiu, Z., Yang, H., eds.: TASE. (2012)
27. Alur, R., Torre, S.L., Pappas, G.J.: Optimal paths in weighted timed automata. In: HSCC. (2001)
28. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: HSCC. (2001)