Experiments in the use of τ-simulations for the components-verification of real-time systems

Françoise Bellegarde, Jacques Julliand, Hassan Mountassir, Emilie Oudot

LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté
FRE CNRS 2661
16, route de Gray,
25030 Besançon Cedex
Ph:(33) 3 81 66 66 51, Fax:(33) 3 81 66 64 50
{bellegar,julliand,mountass,oudot}@lifc.univ-fcomte.fr

ABSTRACT

We present a verification framework exploiting τ-simulations as a way to preserve local linear properties checked on the components of real-time systems. Therefore, we consider a component-based modeling of real-time systems. Their properties are expressed in a timed logic, MITL (Metric Interval Temporal Logic).
For component-based models, traditional verification techniques are generally applied to the complete composed model, even if some properties only concern some components of the system, if not only one. We show that it is possible to check such local linear properties (safety as well as liveness) on the components they concern, and then to ensure their preservation using τ-simulation relations. We show the interest of the method by applying it on two real-time systems examples and by comparing the results with traditional verification techniques.

Keywords

τ-simulation, integration of components, timed systems, preservation of linear-time properties

1. MOTIVATIONS

Component-based modeling is a modeling method that receives more and more attention. In particular, timed systems are often modeled this way. First, it consists in decomposing the system into a set of sub-systems, called components. The complete model is obtained by putting all the components together, thanks to some parallel composition operator. To ensure their correctness, such models have global requirements to meet, i.e., requirements about the behaviour of the complete model, as well as so-called local requirements. Local requirements are properties concerning the components (or subsets of components) of the system.
Model-checking is a verification method that can be used in order to verify these properties on the model. For both kinds of properties, the verification is performed on the global model. However, this method is generally not applicable on large-sized systems, due to the exponential blow-up of the state space.
We propose a verification method for local linear properties of real-time systems modeled in a compositional framework, by taking advantage of the modeling process. We propose to model systems incrementally, by integration of components: instead of building once the complete model, components (or assembling of components) are integrated step by step to the others, and local properties are checked on these components (or assembling of components) before integration. Model-checking is still applicable since the size of each component / assembling of components is small enough.
To ensure that locally established properties continue to hold after integration, we propose to use timed τ-simulations, i.e., simulation relations extended to handle timing aspects and internal activity of the models. Indeed, it is known that in the untimed case, classic τ-simulations preserve linear safety properties. Divergence-sensitive and stability-respecting τ-simulations [12] also handle the preservation of linear liveness properties. For that matter, this last kind of τ-simulation is already used in other incremental modeling processes, such as the refinement of B event systems [1], which preserves LTL properties [9].
In [5], we defined a timed τ-simulation adapted for timed systems and showed that it preserves safety (un)timed properties. Moreover, we showed that it is well-adapted with integration of components made with the classic parallel composition operator $||$. That is, given two components A and B, A τ-simulates $A||B$ and B τ-simulates $A||B$. In terms of preservation of properties, this means that linear safety properties of A and B are preserved *for free* when A and B are merged together. The timed τ-simulation is also compatible with the operator $||$: given a third component C, if B τ-simulates A, then $B||C$ τ-simulates $A||C$. Compatibility allows to benefit of the compositionnality property:

Fifth International Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2006) November 10-11

given components A, B, C and D, if B τ-simulates A and D τ-simulates C, then $B||D$ τ-simulates $A||C$. Thus, this first timed τ-simulation has nice properties w.r.t. parallel composition. However, as it only preserves safety properties, we extended it into a divergence-sensitive and stability-respecting (DS) timed τ-simulation to preserve a larger spectrum of properties, such as liveness or bounded liveness properties. We proved in [5] this new relation preserves all properties which can be expressed by the linear-time logic MITL [4].

In this paper, we aim at showing the interest of the verification method we propose by applying it to the verification of the local properties of two timed systems. In both examples, we compare the application of our method with the classic verification method, consisting in verifying directly all the properties on the complete model. The first example is a production cell, made up of at least seven components. We focus in particular on the local properties of one of these components, and perform the verification locally on this component before verifying the preservation thanks to the timed τ-simulations. This example shows that the cost of this kind of verification (local verification and preservation checking) is lower than the cost of the classic verification method, in terms of computation times. The second example is a well-known protocol, the CSMA/CD protocol, composed of a medium and at least two senders. With this example, we show that one of the main properties of the protocol can be checked by only considering the smallest possible number of senders, i.e. two, instead of verifying it with a greater number of senders. Moreover, in both examples, we identify cases where the classic verification method is not applicable while the method we propose is.

The paper is organized as follows. In section 2, we present the model we use for modeling timed systems – timed automata [3], the logical formalism MITL used to formulate their properties and the parallel composition || used for the integration of components. In section 3, we recall the definitions and results of [5], where we defined timed τ-simulations and gave its properties w.r.t. the preservation of MITL properties and the compatibility with parallel composition. Section 4 shows the interest in practice of the verification method we propose, by applying it to two examples of timed systems and comparing the results with the classic verification method. Finally, section 5 sums up the results presented in the paper and plans the future works.

Related works. Numerous works have been devoted to the study of timed simulation relations and their preservation ability. A time-abstracting simulation has been studied in [13], where an algorithm to check the relation is proposed. However, timed properties are not preserved by this relation. A timed simulation was defined in [17]. The authors showed that the problem of verifying the existence of this timed simulation is EXPTIME. But, internal activity is not handled by this relation. The closest notion of simulation, in comparison with our timed τ-simulation, is the timed ready simulation of [14]. Internal activity is also considered. As our (DS) timed τ-simulation, this relation also handles the preservation of safety properties, but does not preserve other kind of properties, such that liveness properties. To our knowledge, there is no previous definition or use of simulation relations handling internal activity as well as timing constraints, and preserving safety and, especially, liveness properties.

2. MODELING TIMED SYSTEMS

Timed automata [3] are amongst the most studied formal models for timed systems. They are classical finite automata with real-valued variables called clocks modeling the time elapsing.

2.1 Clock valuations and clocks constraints

Let X be a set of clocks. A clock valuation over X is a function $v : X \to R^+$ assigning to each clock in X a real value. For $\delta \in \mathbb{R}^+$, the valuation $v + \delta$ is obtained by adding δ to the value of each clock. Given $Y \subseteq X$, and a valuation v over X, the dimension-restricting projection of v on Y [18], written $v|_Y$, is a new valuation containing only the values in v concerning the clocks in Y. The *reset operation* on v of the clocks in Y, written $[Y := 0]v$, creates a valuation obtained from v by setting to zero all clocks in Y, and leaving the values of other clocks ($\in X \backslash Y$) unchanged. A clock constraint over X is a predicate of the form

$$g ::= x \sim c \mid g \wedge g \mid \texttt{true} \text{ where } x \in X, c \in \mathbb{N} \text{ and } \sim \in \{<, \leq, =, \geq, >\}$$

The set of all clock constraints over X is called $\mathcal{C}(X)$. We say that a valuation v satisfies a constraint $x \sim c$, written $v \in x \sim c$, if $v(x) \sim c$. Note that a clock constraint over X defines a X-polyhedron. The *reset operation* defined on valuations can be extended straightforwardly to polyhedra. The *backward diagonal projection* of a polyhedron g defines a new polyhedron $\swarrow g$ s.t. $v' \in \swarrow g$ if $\exists \delta \in \mathbb{R}^+ \cdot v' + \delta \in g$.

2.2 Timed automata (TA)

Syntax. Let *Props* be a set of atomic propositions. A timed automaton A over *Props* is a tuple $\langle Q, q_0, \texttt{Labels}, X, T, \texttt{Invar}, L\rangle$ where Q is a finite set of locations, q_0 is the initial location, *Labels* is a finite set of names of actions and X is a finite set of clocks. $\texttt{Invar} : Q \to \mathcal{C}(X)$ is a function associating to each location a clock constraint called its invariant. The invariant of a location defines the time progress condition for this location. $L : Q \to 2^{Props}$ is the labelling function for the locations. $T \subseteq Q \times \mathcal{C}(X) \times \texttt{Labels} \times 2^X \times Q$ is a finite set of transitions. Each transition can reset clocks and is equiped with a clock constraint called its guard, defining *when* the transition can be taken. We write a transition e as a tuple (q, g, a, λ, q') where q and q' are respectively the source and target location of the transition, g is its guard, a its label and λ the set of clocks to be reset by the transition. In the rest of the paper, we use the notations $\texttt{source}(e)$ and $\texttt{target}(e)$ for q and q', $\texttt{guard}(e)$ for g, $\texttt{label}(e)$ for a and $\texttt{reset}(e)$ for λ.

Semantics. The semantics of A is an infinite graph where states[1] are pairs (q, v) composed of a location q of A (the discrete part of the state) and a clock valuation v s.t. $v \in \texttt{Invar}(q)$. The initial state is the pair (q_0, v_0) where v_0 is

[1]In the rest of the paper, we directly call these states, the states of A, instead of the states of the semantic graph of A.

the clock valuation assigning 0 to each clock in X. The transitions of the graph are either discrete transitions or time transitions:

- discrete transitions: given a transition $e = (q, g, a, \lambda, q')$ of A, $(q, v) \xrightarrow{e} (q', v')$ is a discrete transition of the graph if $v \in g$ and $v' \in \texttt{Invar}(q')$. The valuation v' is obtained from v by resetting the clocks in λ. We call (q', v') a discrete successor of (q, v),

- time transitions: $(q, v) \xrightarrow{\delta} (q, v+\delta)$ is a time transition of the graph, for $\delta \in \mathbb{R}^+$, if $v + \delta \in \texttt{Invar}(q)$. We call $(q, v + \delta)$ a time successor of (q, v).

Runs. A run $\rho = (q_0, v_0) \xrightarrow{\delta_0} (q_0, v_1) \xrightarrow{e_0} (q_1, v_2) \xrightarrow{\delta_1} (q_1, v_3) \xrightarrow{\delta_2} (q_1, v_4) \xrightarrow{e_1} (q_2, v_5) \cdots$ of a TA A is a path in its semantic graph. Note that consecutive time transitions are not concatenated. We note $\Gamma(A)$ the set of runs of A. A run ρ is *non-zeno* if time can progress along ρ without upper bound. We write $\texttt{time}(\rho)$ for the total time elapsed during ρ. If $\texttt{time}(\rho) = \infty$ then ρ is called non-zeno.

2.3 Properties of timed systems

MITL (Metric Interval Temporal Logic) [4] is a logical formalism allowing to express linear timed properties. It can be viewed as an extension of the (untimed) linear logic LTL [16], where each temporal operator used in the formulas is constrained by a non singular interval with integer bounds (a singular interval is of the form $[a, a]$, i.e., it is closed and the left and right bounds are equal). MITL formulas are defined inductively by the following grammar:

$$\varphi ::= ap \mid \neg\varphi \mid \phi \vee \psi \mid \phi \; \mathcal{U}_I \; \psi$$

where ap is an atomic proposition and I is a non singular interval with integer bounds. Other classical temporal operators can be defined: $\Diamond_I \varphi = true \; \mathcal{U}_I \varphi$ (eventually φ) and $\Box_I \varphi = \neg\Diamond_I \neg\varphi$ (always φ).

MITL properties are interpreted over runs of a timed automaton. Intuitively, a property $\phi \; \mathcal{U}_I \; \psi$ holds on a run if, when ϕ is met, ϕ holds until ψ is true. Moreover, ψ must hold at a time t within the time interval I following the moment when ϕ was true. An MITL property is satisfied by a timed automaton if it holds on each run of the automaton.

2.4 Integration of components

Consider a timed system composed of a set of components A_1, A_2, $\cdots$, A_n, each one modeled by a TA. Integration of components is a type of incremental modeling, which consists in first considering one component, for instance A_1. Then, the other components are successively added to A_1, until obtaining the complete system. We consider that this integration is achieved by using the classic parallel composition operator $||$. This composition is defined as a synchronized product where the synchronizations are done on actions with the same label, while other actions interleave. Formally, consider two TA $A_i = \langle Q_i, q_{0_i}, \texttt{Labels}_i, X_i, T_i, \texttt{Invar}_i, L_i \rangle$, for $i = 1, 2$, s.t. $X_1 \cap X_2 \neq \emptyset$. The parallel composition of A_1 and A_2, written $A_1||A_2$, creates a new TA which set of clocks is $X_1 \cup X_2$ and which labels are the set $\texttt{Labels}_1 \cup \texttt{Labels}_2$. The set Q of locations consists of pairs (q_1, q_2) composed of a location of each A_i. $\texttt{Invar}((q_1, q_2))$ is defined as $\texttt{Invar}(q_1) \wedge \texttt{Invar}(q_2)$ and $L((q_1, q_2))$ is the set $L(q_1) \cup L(q_2)$. The initial location is the pair (q_{0_1}, q_{0_2}). The set of transitions T is given by the three following rules:

Synchronization: $$\frac{(q_1,g_1,a,\lambda_1,q_1') \in T_1 \;,\; (q_2,g_2,a,\lambda_2,q_2') \in T_2}{((q_1,q_2),g_1 \wedge g_2,a,\lambda_1 \cup \lambda_2,(q_1',q_2')) \in T}$$

Interleaving: $$\frac{(q_1,q_2) \in Q \;,\; (q_1,g_1,a,\lambda_1,q_1') \in T_1 \;,\; a \notin \texttt{Labels}_2}{((q_1,q_2),g_1,a,\lambda_1,(q_1',q_2)) \in T}$$
$$\frac{(q_1,q_2) \in Q \;,\; (q_2,g_2,a,\lambda_2,q_2') \in T_2 \;,\; a \notin \texttt{Labels}_1}{((q_1,q_2),g_2,a,\lambda_2,(q_1,q_2')) \in T}$$

Incremental modeling, and in particular integration of components, is a way to cope with the complexity of the verification when large-sized model are treated. Indeed, it could allow to verify local properties of the components at each step of the modeling, i.e., at each integration of components, instead of performing the verification directly on the complete system. However, this reasoning is valid only if the integration preserves the properties already checked.

3. EXPLOITING SIMULATIONS TO PRESERVE LOCAL PROPERTIES

We showed in [5] that timed τ-simulations are sufficient conditions for the preservation of MITL properties during incremental modeling, and in particular, integration of components. That is, if a component τ-simulates the whole system w.r.t. some τ-simulation relation, then already established properties of the component are preserved after integration in its environment. We defined in [5] two τ-simulation relations: one dealing with safety MITL properties (called a *timed τ-simulation*)[2], and the other to handle all MITL properties (*divergence-sensitive and stability-respecting timed τ-simulation*).

3.1 Timed τ-simulations

Consider a TA A_1 that has to be integrated in an environment E. We note $A_2 = A_1||E$ the result of the integration. Consider also that A_1 and E have a common subset of labels of actions – the synchronized actions –, i.e., $\texttt{labels}_{A_1} \cap \texttt{labels}_E \neq \emptyset$. Let us rename by τ all the labels of the own actions of the environment (the non-common labels of E) and call them τ-actions.
The *timed τ-simulation* $\mathcal{S}$ between the semantic graphs of A_2 and A_1 is characterized by the following points: if A_2 can make an action of A_1 after some amount of time (that is, either a synchronized action or an own action of A_1), then A_1 could also do the same action after the same amount of time (clauses 1 and 2 of Definition 1), and own actions of E (τ-actions) stutter (clause 3).

DEFINITION 1 (TIMED τ-SIMULATION $\mathcal{S}$). *Let* $A_1 = \langle Q_1, q_{0_1}, \texttt{Labels}_1, X_1, T_1, \texttt{Invar}_1, L_1 \rangle$ *and* $A_2 = \langle Q_2, q_{0_2}, \texttt{Labels}_1 \cup \{\tau\}, X_2, T_2, \texttt{Invar}_2, L_2 \rangle$ *be two TA such that* $X_1 \subseteq X_2$. *S_1 and S_2 are the respective set of states of A_1 and A_2. The relation $\mathcal{S}$ is included in $S_2 \times S_1$. We say that* $(q_2, v_2)\mathcal{S}(q_1, v_1)$ *if* $v_2\rfloor_{X_1} = v_1$ *and*

[2]Note that, although *deadlock-freedom* properties can be classified as safety properties, we do not consider them like this, but as a separate class of properties. Thus, this kind of properties are not preserved by the timed τ-simulation.

1. *Strict simulation:*
$(q_2, v_2) \stackrel{e_2}{\rightarrow} (q_2', v_2') \wedge \texttt{label}(e_2) \in \Sigma_1 \Rightarrow \exists (q_1', v_1') \cdot$
$((q_1, v_1) \stackrel{e_1}{\rightarrow} (q_1', v_1') \wedge \texttt{label}(e_1) = \texttt{label}(e_2) \wedge (q_2', v_2') \; \mathcal{S} \; (q_1', v_1'))$.

2. *Time transitions:*
$(q_2, v_2) \stackrel{\delta}{\rightarrow} (q_2, v_2') \Rightarrow$
$\exists (q_1, v_1') \cdot ((q_1, v_1) \stackrel{\delta}{\rightarrow} (q_1, v_1') \wedge (q_2, v_2') \; \mathcal{S} \; (q_1, v_1'))$.

3. *Stuttering:*
$(q_2, v_2) \stackrel{e_2}{\rightarrow} (q_2', v_2') \wedge \texttt{label}(e_2) = \tau \Rightarrow (q_2', v_2') \; \mathcal{S} \; (q_1, v_1)$.

We say that A_1 τ-simulates A_2 w.r.t. $\mathcal{S}$, written $A_2 \preceq A_1$, if $s_{0_2} \; \mathcal{S} \; s_{0_1}$, where s_{0_1} and s_{0_2} are the respective initial states of A_1 and A_2.

This τ-simulation only preserves safety properties. To preserve also liveness properties, it has to be (1) *stability - respecting*: the integration of A_1 in E must not create deadlocks in comparison with A_1 (all the deadlocks appearing in $A_1||E$ must already exist in A_1), and (2) *divergence-sensitive*: the own actions of the environment E must not have the possibility to take the control forever, i.e., $A_1||E$ must not contain non-zeno runs composed of an infinite sequence of successive τ-actions. Such a run is called a non-zeno τ-cycle, and a TA containing such a run is called τ-divergent.

The predicate `free`. To deal with the non-introduction of deadlocks during the integration, we use the predicate `free` introduced in [18]. Informally, $\texttt{free}(q)$ computes the set of valuations of the states with discrete part q that can let some time pass and then take a discrete transition (i.e., non-blocking states):

$$\texttt{free}(q) = \bigcup_{e \in \texttt{out}(q)} \swarrow (\texttt{guard}(e) \cap ([\texttt{reset}(e) := 0]\texttt{Invar}(\texttt{target}(e))))$$

where $\texttt{out}(q)$ is the set of discrete transitions leaving from q.

Non-zeno τ-cycles. Let A be a TA where some labels of actions are renamed by τ. We say that A does not contain any non-zeno τ-cycles (and thus that A is not τ-divergent) if:
$\forall \rho, k \cdot (\rho \in \Gamma(A) \wedge \texttt{time}(\rho) = \infty \wedge k \geq 0 \Rightarrow$

$$\exists k', e \cdot (k' \geq k \wedge (\rho, k') \stackrel{e}{\rightarrow} (\rho, k' + 1) \wedge \texttt{label}(e) \neq \tau)).$$

We restrict the timed τ-simulation of Def. 1 by adding conditions imposing divergence-sensitivity and stability-respect. This Divergence-sensitive and Stability-respecting (DS) timed τ-simulation is defined as follows.

Definition 2 (DS timed τ-simulation $\mathcal{S}_{ds}$). *Consider two TA* $A_1 = \langle \mathrm{Q}_1, \mathrm{q}_{0_1}, \texttt{Labels}_1, \mathrm{X}_1, \mathrm{T}_1, \texttt{Invar}_1, \mathrm{L}_1 \rangle$ *and* $A_2 = \langle \mathrm{Q}_2, \mathrm{q}_{0_2}, \texttt{Labels}_1 \cup \{\tau\}, \mathrm{X}_2, \mathrm{T}_2, \texttt{Invar}_2, \mathrm{L}_2 \rangle$*, such that* $X_1 \subseteq X_2$ *and* A_2 *is not* τ*-divergent.* S_1 *and* S_2 *are the respective set of states of* A_1 *and* A_2*. The relation* $\mathcal{S}_{ds}$ *is included in* $S_2 \times S_1$*. We say that* $(q_2, v_2)\mathcal{S}_{ds}(q_1, v_1)$ *if*

$$(q_2, v_2)\mathcal{S}(q_1, v_1) \text{ and } v_2 \notin \texttt{free}(q_2) \Rightarrow v_1 \notin \texttt{free}(q_1)$$

We say that A_1 τ-simulates A_2 w.r.t. $\mathcal{S}_{ds}$, written $A_2 \preceq_{ds} A_1$, if $s_{0_2} \; \mathcal{S}_{ds} \; s_{0_1}$.

3.2 Properties of timed τ-simulations

We give in this section the main propositions and theorem showing the interest of the timed τ-simulations for the incremental modeling of timed systems. Indeed, the timed τ-simulation is well-adapted to the parallel composition operator $||$ (Proposition 1). Moreover, this operator does not add τ-divergence if the environment in which a component is integrated is not τ-divergent (Proposition 2). Theorem 1 is the main result, showing that the DS timed τ-simulation preserves MITL properties. Proofs can be found in [5].

Proposition 1. *Let A, B, C and D be TA. Then, we have the following:*

1. $A||B \preceq A$,
2. *if* $A \preceq B$ *then* $A||C \preceq B||C$,
3. *if* $A \preceq B$ *and* $C \preceq D$ *then* $A||C \preceq B||D$.

Proposition 2. *Consider two TA A and B s.t. some actions of B are renamed by* τ*. If B is not* τ*-divergent, then neither is* $A||B$.

Theorem 1. *Let* φ *be a MITL property, A and B be TA. If* $A \models \varphi$ *and* $B \preceq_{ds} A$*, then* $B \models \varphi$.

Therefore, in the context of integration of components, the preservation of local safety properties can be obtained for free, since parallel composition is compatible with the timed τ-simulation. The preservation of local linear liveness properties of a component can be ensured only by checking that its integration in an environment does not create deadlocks and that this environment is not τ-divergent.

Remark 1. *TCTL [2] is a branching-time logic which is often used to express timed properties. However, TCTL properties are not preserved by the DS timed* τ*-simulation. Indeed, even in the untimed case, simulation relations can not handle the preservation of branching-time properties. This kind of properties gives the possibility to add existential or universal quantifiers in the formulas. In particular, properties including existential quantifications, such as reachability properties, are not preserved (note that each MITL formula is implicitly preceded by a universal quantification, meaning that the property must hold on all runs). One often need to use bisimulation relations to ensure the preservation of such properties, but they are not appropriate to formalize incremental modeling, and thus, integration of components.*

3.3 Checking the simulations

To check the DS timed τ-simulation, we developped the tool VESTA[3]. The structure of the tool was guided by the structure of the tool OPEN-KRONOS [18], allowing an easy connection to the OPEN-CAESAR platform [11]. It takes as input two TA A_1 and A_2, where A_2 represents the integration of A_1 in an environment E, i.e. $A_1||E$.
The tool performs a joint on-the-fly depth-first search on symbolic graphs representing the two TA (the so-called *simulation graphs* [18]), and checks if $A_2 \preceq_{ds} A_1$. This verification is in $\mathcal{O}((|Z_1| + | \longrightarrow_1 |) \times (|Z_2| + | \longrightarrow_2 |))$, where Z_i and $\longrightarrow_i$, for $i = 1, 2$, are respectively the set of states and

[3]available at: `http://lifc.univ-fcomte.fr/~oudot/VeSTA`

transitions of the simulations graphs of each A_i. If the verification fails, the tool returns a diagnostic. This diagnostic consists in a trace of A_2 containing a state which does not satisfy the relation, and the corresponding trace in A_1. To check if A_2 is τ-divergent, we use the tool PROFOUNDER [19], which can in particular detect non-zeno cycles. Thus, we use it to detect non-zeno τ-cycles first in E, then in A_2 if τ-cycles are detected in E.

4. EXPERIMENTS

We present in this section examples on which we lead experiments to show the interest of timed τ-simulations. For each case study, the verification of the properties was achieved with the tool KRONOS [20]. KRONOS is a verification tool for timed systems which performs TCTL model-checking [2], in particular for component-based models (indeed, KRONOS can compute the parallel composition of TA). TCTL is a logical formalism that allows to express branching-time properties. It can be seen as the timed extension of the untimed logic CTL [8, 10]. To our knowledge, there is no tool performing MITL model-checking. Thus, we focused on linear-time properties that can also be expressed in TCTL to lead the verification with KRONOS. The verification of the simulations was done with VESTA.

4.1 Production Cell

The production cell case study was developed by FZI (the Research Center for Information Technologies, in Karlsruhe) as part of the Korso project. The goal was to study the impact of the use of formal methods when treating industrial applications. Thus, this case study was treated in about thirty different formalisms. We treat it with timed automata, as it was in [7].

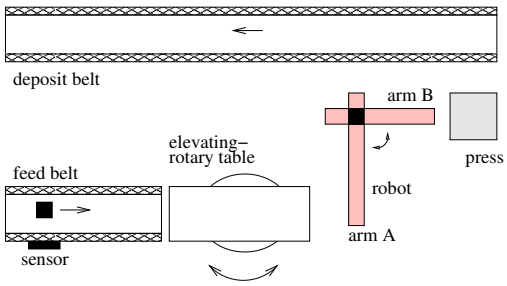

Figure 1: The Production Cell

Modeling. The cell consists of six devices: a feed-belt and a deposit-belt, from which pieces to be treated arrive and are evacuated, a sensor detecting the arrival of the pieces, an elevating rotary table, a two-arms robot and a press (see Fig. 1). The sensor, on the feed-belt, detects when a piece passes in front of it and sends a signal to the robot to inform that a piece is going to be available. When the piece arrives at the end of the feed-belt, it is transfered to the table which goes up while turning until being in a position where the piece can be taken by the arm A of the robot. The robot turns 90° so that the arm A can put down the piece on the press, where it is processed and then transported by the arm B to the deposit belt.
In the following, we focus on local properties concerning the robot. Thus, let us give details about its behaviour: when a piece is available on the table, the robot picks it up and moves to the press so that its arm B is in front of the press. If there is a piece on the press, the arm B takes it, otherwise the robot goes on turning to place the arm A in front of the press to put its piece down. Next, if the arm B is full, the robot goes to the deposit belt to unload its piece (and then goes back to the table), else it goes to an intermediary position, called wait position. From this wait position, the robot can either go back to the table to pick a new piece up or to the press to get a processed piece.
The cell is modeled by at least seven components, one for each device, and one or several pieces. Each component is modeled by a TA. These TA can be found in [7]. The complete model is obtained by making the parallel composition of all these components. The timing constraints of the plant are shown in Fig. 2. We give in Fig. 3 the size, in terms of number of states and transitions, of the simulation graphs of each component.

Device	Description	Time
Robot	moves to press	5
Robot	turns 90°	15
Robot	moves to deposit belt	5
Robot	from deposit belt to table	25
Robot	from deposit belt to wait pos.	22
Robot	from press to wait pos.	17
Robot	from wait pos. to table	3
Robot	from wait pos. to press	2
Robot	at wait pos.	2
Feed Belt	piece moves to sensor	3
Feed Belt	piece moves to table	1
Table	raises and turns	2
Table	returns and turns	2
Press	presses a piece	22-25
Press	ready for a new piece	18-20
Deposit Belt	evacuates a piece	4

Figure 2: Timing constraints for the production cell

Component	Robot	Press	Feed belt	Dep. Belt
States/Trans.	39/40	7/7	6/6	4/4
Component	Table	Sensor	Piece	Complete Model
States/Trans.	6/6	2/2	7/7	1655/2395

Figure 3: Size of the simulation graphs of each component of the production cell

Verification. To ensure that the modeling is correct, there are several properties to check. We focus on the local properties of the components, and in particular on the local properties concerning the robot. Here is a non-exhaustive list of dynamic properties to check on the robot component. Properties 1 and 2 are safety requirements, properties 3 and 4 are liveness requirements and properties 5 to 7 are bounded-response requirements:

(1) When the robot is in wait position, its two arms are empty,

(2) The robot is not waiting in front of the table if the arm A is full,

(3) If there is a piece on arm B, the robot will eventually go to the deposit belt,

(4) If there is a piece on arm A, the robot will eventually go to the press,

(5) When the robot is in front of the deposit belt, then it goes back to the table within 25 time units (t.u.) if there are no pieces on the press,

(6) When the robot is in front of the deposit belt, then it goes to the wait position within 22 t.u. if there is a piece on the press,

(7) When it is in wait position, either the robot goes to the press within 2 t.u. to unload it or it goes back to the table within 3 t.u. to pick a new piece.

The following liveness property concerns the correct interaction between the robot and the press :

(8) If arm A is full then the press will eventually be free.

We used two approaches to verify these properties on the plant. As a first approach, we verified the properties in a classic way, directly on the global model, with only one piece. As a result, we obtained that all the properties hold on this model of the plant. The second approach consisted in verifying the properties locally, i.e., on the robot component for properties 1 to 7, and on the composition $robot||press$ for property 8. Here again, the verification succeeded. Next, to guarantee the preservation of these properties, first when integrating the robot with the press, then when integrating the composition $robot||press$ with the rest of the components, we used the DS timed τ-simulation. We used our tool VESTA to check it for both cases, and obtained as a result that the verification was successful. Thus, properties are preserved.

Fig. 4 gives the results of the comparison of the two approaches in terms of verification times (in seconds). We can see that, even on this small example, the second approach only needs 0,57 sec. of computation time to ensure that the properties hold on the cell, whereas the classic approach consumes 19,58 sec.
Note that, in both approaches, we focused on a global system which contains only one piece. The reason is the following. First, in the case of the second approach, adding other pieces to the global system does not affect the results of the preservation. As the component piece already exists in the global system and that there are no synchronizations between the pieces, no new deadlocks can appear while adding a new piece. Indeed, the system can behave like it did with only one piece, or synchronize with the new piece. In this last case, as the environment of the pieces could synchronize with one piece without introducing deadlocks, then it will synchronize with the new pieces in the same way, thus without introducing deadlocks. On the other hand, in the first approach, adding pieces considerably increase the computation time for the verification of liveness or bounded-response properties. Indeed, even with few pieces, the memory needed to perform classic verification of such properties is too large for the verification to be run to completion.

4.2 CSMA/CD protocol

The CSMA/CD protocol (Carrier Sense, Multiple Access with Collision Detection protocol) [15] is used in broadcast networks with a single channel, to which many stations try to access. The protocol solves the problem of how to assign the use of the channel to one of the stations and allows to detect collisions when two stations try to send data simultaneously.

Modeling. The protocol works as follows: a station tries to send a data. If the channel is busy, it waits some amount of time and then retries to send. Otherwise, it begins to send its data. If two or more stations begin to send a data simultaneously, and thus a collision arises, these stations detect the collision and wait some amount of time to begin retransmitting the data.
At least two TA are used to model the protocol: one per station, called sender, and a medium. These TA are shown in Fig. 5. Transitions cd_i represent a collision detection by the i^{th} sender, and cd_M models this detection by the medium. The complete model of the protocol can contain several senders. For n senders, the synchronizations are the following: for $a \in \{begin, busy, end\}$, there is a synchronized transition $a_i||a_M$[4], for $i \in [1..n]$. Moreover, the cd transitions of each TA are synchronized, i.e., in the parallel composition, there is a transition $cd_M||cd_1||\cdots||cd_n$.

Verification. A main property of the protocol is that *whatever the number of stations, if a collision occurs between two stations i and j, $i \neq j$, both detect it within 26 t.u.* This is a bounded-response property, written in MITL by:

$$\Box(transm_i \wedge transm_j \Rightarrow \Diamond_{\leq 26}(coll\_detected_i \wedge coll\_detected_j)).$$

This property holds in the case of a modeling with only two senders S_1 and S_2. The verification with KRONOS takes less than 0.001 seconds of computation time. We want to ensure that adding other senders does not alter the result of the verification. That is, we want that, with a number $n > 2$ of senders, if S_1 and S_2 transmit their data simultaneously, they still detect the collision. To preserve this bounded-liveness property, we have to check that

$$S_1||S_2||S_3||\cdots||S_n||Medium \preceq_{ds} S_1||S_2||Medium.$$

As the senders S_i are not τ-divergent, and with Proposition 1, we only have to check that the addition of $S_3, \cdots, S_n$ does not add deadlocks to $S_1||S_2||Medium$.
As in the production cell example, there already exist two senders in the system. Thus, when adding a new sender, the synchronizations of this sender with the medium (actions *begin*, *end* and *busy*) can take place as they did with only two senders. Thus, no deadlocks can be introduced by these synchronizations. The main difference with the production cell example is that there exists synchronizations between all the components of the system, i.e., all the senders and the medium (action *cd*). That is, when two senders detect a collision and try to take action *cd* between the locations `transm`$_i$ and `coll_detected`$_i$, all other senders must allow them to detect the collision and thus, must allow them to

[4]Note that, for more simplicity, we extend here the notation || initially defined on timed automata, to transitions. Thus, $a||b$ denotes the transition resulting of the synchronization of transitions a and b.

Property	Global Verification	Local Verification	Preservation checking
Prop. 1	0.01	< 0.001	
Prop. 2	0.01	< 0.001	
Prop. 3	0.98	< 0.001	
Prop. 4	15.79	0.04	0.05
Prop. 5	0.68	< 0.001	
Prop. 6	0.48	< 0.001	
Prop. 7	0.7	< 0.001	
Prop. 8	0.93	0.02	0.46
Total	19.58	0.06	0.51

Figure 4: Production cell : local and global verification times (in seconds)

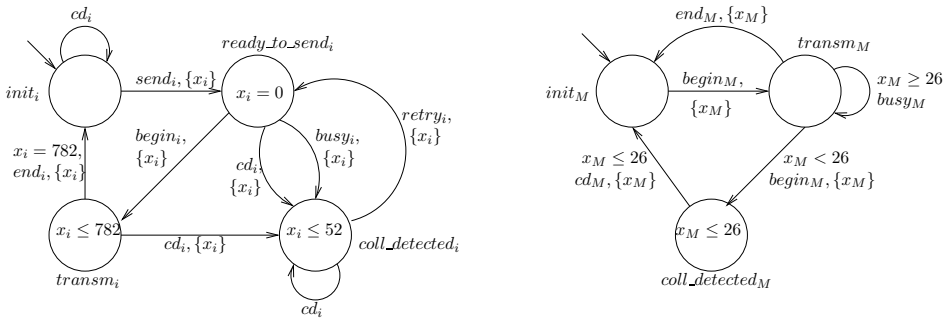

Figure 5: Timed automata for the ith sender and the medium of the CSMA/CD protocol

take this transition. If not, a new deadlock occurs, since the synchronization can not take place, whereas it could be taken with two senders. However, we see that, at each location of the TA of the senders, the transition *cd* appears. This means that whenever a collision must be detected by two senders (i.e., taking transition *cd* between the locations $transm_i$ and $coll\_detected_i$), the other senders also have the possibility to take a transition `cd`, allowing the two first senders to detect the collision. No deadlocks can appear while adding new senders, and thus, the property is preserved whatever the number of senders may be.
We compared our approach with a classic verification approach, and tried to verify this property, for instance, for two senders S_1 and S_2, using the tool KRONOS:

- Up to six senders (S_1 to S_6), the property can be checked successfully. The computation times for the verification changed from less than 0,5 seconds (three senders) to more than 57 minutes (six senders). These computations times take into account the time consumed to make the parallel composition of all the components and the verification time. Note that, in the last case, the verification time takes about 30 seconds, while the construction of the composition takes almost 57 minutes.

- For seven senders or more, the construction of the TA resulting of the parallel composition of all the components takes a considerably long time. For instance, we aborted the construction for seven senders after ten hours of computation without results. Thus, as the composition could not be obtained, it was impossible to perform the verification of the property.

5. CONCLUSION AND FUTURE WORKS

We aimed at treating the problem of the verification of linear-time properties, expressed in MITL, of timed systems. We considered timed systems modeled in a compositional framework, and focused on the verification of local properties of the components.

In [5], we proposed to use timed τ-simulations as a way to verify these properties at a lower cost. The main thesis is that a local linear property can be checked only on the component it concerns, instead of on the complete composed system, and that the preservation of the property when integrating the component in its environment can be checked by means of timed τ-simulation relations. More precisely, we defined two such relations: a timed τ-simulation handling the preservation of linear safety properties, and a divergence-sensitive and stability-respecting timed τ-simulation for the preservation of all MITL properties, in particular liveness properties.

In this paper, we studied the impact in practice of this methodology. We applied it to two timed systems, and compared the results with the ones obtained by a direct verification on the complete system. It turns out that, in terms of computation times, the methodology appears to be more efficient. Moreover, both examples show the interest of the methodology when a system S is susceptible to include an indeterminate number n of components $C_{i,i=1..n}$ of the same "kind", i.e., $S = E||C_1||\cdots||C_n$. In this case, we say that S has n as a parameter. For instance, the parameter of the production cell is the number of pieces on the cell, while in the CSMA/CD protocol, it is the number of senders. In such a parametrized system, the interesting cases are when the verification can be performed with a small fixed number

l of such components, that is $S_l = E||C_1||\cdots||C_l$, implying the preservation of the properties on $S_m = E||C_1||\cdots||C_m$, $\forall m \geq l$, under some simple conditions. As the parallel composition is compatible with the timed τ-simulation, it is enough that the conditions ensure that adding the $C_i s$ does not introduce any deadlocks. For instance, for the production cell example, the condition is that there are no synchronizations between the pieces. Thus, an interesting work is to determine such simple conditions.

Another work direction would be to study the compatibility of the timed τ-simulations with other composition operators, in particular those preserving deadlock-freedom, such as the one presented in [6]. Indeed, this analysis would allow to evaluate the interest of the methodology we propose in a more general framework, i.e., also for systems using composition paradigms which differ from the classic parallel composition one.

6. REFERENCES

[1] J.-R. Abrial. Extending B without changing it (for developing distributed systems). In *1st Conference on the B method*, pages 169–190, Nantes, France, November 1996.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking in Dense Real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[5] F. Bellegarde, J. Julliand, H. Mountassir, and E. Oudot. On the contribution of a τ-simulation in the incremental modeling of timed systems. In *Proceedings of the 2^{nd} International Workshop on Formal Aspects of Component Software (FACS'05)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 97–111, Macao, Macao, October 2005. Elsevier.

[6] S. Bornot, J. Sifakis, and S. Tripakis. Modeling Urgency in Timed Systems. In *COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[7] A. Burns. How to verify a safe real-time system: The application of model-checking and timed automata to the production cell case study. *Real-Time Systems Journal*, 24(2):135–152, 2003.

[8] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[9] C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement Preserves PLTL Properties. In *Proceedings of 3^{rd} International Conference on B and Z Users (ZB'03)*, volume 2651 of *Lecture Notes in Computer Science*, pages 408–420, Turku, Finlande, June 2003. Springer-Verlag.

[10] E. Emerson and J. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the 14^{th} ACM Symp. Theory of Computing (STOC'82)*, pages 169–180, San Francisco, CA, USA, May 1982.

[11] H. Garavel. OPEN/CAESAR: An Open Software Architecture for Verification, Simulation and Testing. In B. Steffen, editor, *Proceedings of 1^{st} International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, Lisboa, Portugal, March 1998.

[12] R. v. Glabbeck. The Linear Time - Branching Time Spectrum II ; The semantics of sequential systems with silent moves. In *Proceedings of 4^{th} international Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Hildesheim, Germany, august 1993. Springer-Verlag.

[13] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36^{th} IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[14] H. Jensen, K. Larsen, and A. Skou. Scaling up Uppaal : Automatic verification of real-time systems using compositionnality and abstraction. In *Proceedings of the 6^{th} international symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*, pages 19–30, London, UK, 2000. Springer-Verlag.

[15] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering, Special Issue on Real-Time Systems*, 18(9):794–804, September 1992.

[16] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18^{th} IEEE Symposium on Foundations Of Computer Science*, pages 46–77, 1977.

[17] S. Tasiran, R. Alur, R. Kurshan, and R. Brayton. Verifying Abstractions of Timed Systems. In *Proceedings of the 7^{th} Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 546–562, Pisa, Italy, 1996.

[18] S. Tripakis. *The analysis of timed systems in practice.* PhD thesis, Universite Joseph Fourier, Grenoble, France, December 1998.

[19] S. Tripakis, S. Yovine, and A. Bouajjani. Checking Timed Büchi Automata Emptiness Efficiently. *Formal Methods in System Design*, 26(3):267–292, May 2005.

[20] S. Yovine. Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.