VeSTA: a Tool to Verify the Correct Integration of a Component in a Composite Timed System[⋆]

Jacques Julliand, Hassan Mountassir, and Emilie Oudot

LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté
16, route de Gray, 25030 Besançon Cedex, France
Ph:+33 (0)3 81 66 66 51, Fax:+33 (0)3 81 66 64 50
{julliand,mountass,oudot}@lifc.univ-fcomte.fr

Abstract. VeSTA is a push-button tool for checking the correct integration of a component in an environment, for component-based timed systems. By correct integration, we mean that the local properties of the component are preserved when this component is merged into an environment. This correctness is checked by means of a so-called divergence-sensitive and stability-respecting timed τ-simulation, ensuring the preservation of all linear timed properties expressed in the logical formalism Mitl (Metric Interval Temporal Logic), as well as strong non-zenoness and deadlock-freedom. The development of the tool was guided by the architecture of the Open-Kronos tool. This allows, as additional feature, an easy connection of the models considered in VeSTA to the Open-Caesar verification platform, and to the Open-Kronos tool.

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

1 Motivations

Model-checking is an attractive automatic verification method to ensure the correctness of models of systems. However, it is well-known that this method has difficulties to handle large-sized models, in particular when treating models involving timing constraints. Component-based modeling is a method often used to model timed systems. First, it consists in decomposing the system into a set of sub-systems, called components. Next, each component is modeled and the interactions between them are specified. The complete model is obtained by putting together all these components with respect to their interactions. With such a modeling, two kinds of properties can be checked to ensure the correctness of the model: global properties concerning the behavior of the complete model, and local properties concerning the behavior of one or some components. For both kind of properties, verification by model-checking is usually performed on the complete model, and thus can become difficult if the size of the model is too large.

[⋆] This work has been partially funded by the ANR-06-SETI-017 TACOS project (Trustworthy Assembling of Components: from requirements to specification). VeSTA is available at the following URL: http://lifc.univ-fcomte.fr/∼oudot/VeSTA.

We propose to use an alternative method for the verification of local properties of the components: integration of components. Integration of components is an incremental development method. It consists, for a local property L of a component C, in checking L only on C. Model-checking is here applicable due to the generally small size of the components. Obviously, L has to be preserved when C is integrated in an environment E. When using the classic parallel composition operator $||$ between components, the preservation of local safety properties of C on $C \| E$ is ensured for free. This is not the case for local liveness properties.

Simulation relations are a way to ensure preservation of properties. They have already been used in the untimed case for this purpose. For instance, [1] defines the refinement of transition systems as a kind of τ-simulation, which ensures the preservation of LTL properties. In the timed case, a time-abstracting simulation is defined in [2], but does not preserve timed properties. Timed simulation is defined in [3], but does not consider the possible internal activity of the systems (internal activity is a main barrier for the preservation of liveness properties). A timed ready simulation is defined in [4], but does not allow to preserve liveness properties. To our knowledge, there is no simulation relation for timed systems, which handles internal activity of the systems, and also preserves liveness properties. Therefore, we defined in [5] a *divergence-sensitive and stability-respecting (DS) timed τ-simulation* for timed components expressed as timed automata [6] and proved it can ensure the preservation of all linear timed properties which can be expressed in the logical formalism MITL [7], thus in particular linear liveness and bounded liveness properties. Strong non-zenoness and deadlock-freedom are also preserved by the relation. That is, if C simulates $C \| E$ with respect to this relation, all linear local timed properties of C are preserved on $C||E$.

The tool VeSTA (**Ve**rification of **S**imulations for **T**imed **A**utomata) was developed to automate the verification of the DS timed τ-simulation. More precisely, VeSTA considers component-based timed systems, developed incrementally by integration of components, where each component is modeled as a timed automaton. It allows to check that local properties of a component (or group of components) of the system are preserved during its integration with other components of this system. The architecture of the tool was inspired by the one of the OPEN-KRONOS tool [8]. Thus, as OPEN-KRONOS, VeSTA benefits of libraries which provide an efficient symbolic representation for networks of timed automata. This choice also allows to connect the models considered in VeSTA to OPEN-KRONOS, and also to the verification platform OPEN-CAESAR[9].

The structure of the paper is the following. In section 2, we recall some background on timed systems, i.e., on the formalisms which are used in VeSTA for the modeling of timed systems and their properties. This section also introduces the divergence-sensitive and stability-respecting timed τ-simulation, and its preservation abilities. Section 3 presents the tool VeSTA: its architecture, the algorithms which are implemented and its graphical user interface. In section 4, we illustrate the interest of VeSTA by using it to verify incrementally properties of a case study concerning a production cell. Section 5 presents some additional features of the tool. Finally, section 6 contains the conclusion and exhibits some future developments for VeSTA.

2 Incremental verification of timed systems

We present here the preliminary notions that we consider concerning component-based timed systems. First, we introduce timed automata which we use to model timed components, and the composition operator we use to assemble these components. Then, we present the simulation relation we defined for timed automata and recall previous results concerning the properties that it preserves during incremental development, and in particular, during integration of components.

2.1 Modeling timed systems

Since their introduction in [6], timed automata are amongst the most studied models for timed systems. They are finite automata with real-valued variables called clocks, to model time elapsing.

Clock valuations and clock constraints. Let X be a set of clocks. A clock valuation over X is a mapping $v : X \to \mathbb{R}^+$, associating to each clock in X a value in $\mathbb{R}^+$. We note $\mathbf{0}$ the valuation assigning the value 0 to each clock in X. Given a clock valuation v and $t \in \mathbb{R}^+$, $v + t$ is the valuation obtained by adding t to the value of each clock in v. Given $Y \subseteq X$, the dimension-restricting projection of v on Y, written $v\rceil_Y$, is the valuation over Y only containing the values in v of clocks in Y. The reset in v of the clocks in Y, written $[Y := 0]v$, is the valuation in which all clocks in Y are reset to zero, while the value of other clocks remains unchanged.
A clock constraint over X is a set of clock valuations over X. The set $\mathcal{C}_{df}(X)$ of diagonal-free clock constraints[1] over X is defined by the following grammar:

$$g ::= x \sim c \mid g \wedge g \mid true$$

where $x \in X$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Diagonal-free clock constraints do not allow comparison between clocks such as $x - y \sim c$. Note that a clock constraint defines a convex X-polyhedron. We note `zero` the X-polyhedron defined by $\bigwedge_{x \in X} v(x) = 0$. The dimension-restricting projection and reset operation can be directly extended to clock constraints. The backward diagonal projection of the X-polyhedron ζ defines a X-polyhedron $\swarrow\zeta$ such that $v' \in \swarrow\zeta$ if $\exists t \in \mathbb{R}^+ \cdot v' + t \in \zeta$. The forward diagonal projection of ζ defines a X-polyhedron $\nearrow\zeta$ such that $v' \in \nearrow\zeta$ if $\exists t \in \mathbb{R}^+ \cdot v' - t \in \zeta$. Given $c \in \mathbb{N}$, the extrapolation of ζ w.r.t. c, written $\mathtt{Approx}_c(\zeta)$, is the smallest polyhedron $\zeta' \supseteq \zeta$ defined intuitively as follows: lower bounds of ζ greater than c are replaced by c, and upper bounds greater than c are ignored. All these operations preserve convexity.

[1] We restrict ourselves to this kind of clocks constraints to ensure the correctness of the construction of the symbolic representation of TA [10].

Timed Automata. Let *Props* be a set of atomic propositions. A timed automaton (TA) over *Props* is a tuple $A = \langle Q, q_0, \Sigma, X, T, \texttt{Invar}, L\rangle$ where Q is a finite set of locations, $q_0 \in Q$ is the initial location, Σ is a finite alphabet of names of actions, X is a finite set of clocks, $T \subseteq Q \times \mathcal{C}_{df}(X) \times \Sigma \times 2^X \times Q$ is a finite set of edges, `Invar` is a function mapping to each location a clock constraint called its invariant and L is a labelling function mapping to each location a set of atomic propositions over *Props*. Each edge is a tuple $e = (q, g, a, r, q')$ where q and q' are the source and target locations, g is a clock constraint defining the guard of the edge, a is the label of the edge and r is the set of clocks to be reset by the edge. We use the notation $\texttt{label}(e)$ to denote the label a of the edge e. Examples of TA can be found in section 4.

Semantics. The semantics of a TA A is an infinite graph $\mathcal{G}(A)$ in which states are pairs (q, v), where q is a location of A and v a clock valuation over the clocks of A, such that $v \in \texttt{Invar}(q)$. The transitions of this graph can be either discrete or time transitions. Consider a state (q, v). Given an edge $e = (q, g, a, r, q')$ of A, $(q, v) \xrightarrow{e} (q', v')$ (where $v' = [r := 0]v$) is a discrete transition in $\mathcal{G}(A)$ if $v \in g$ and $v' \in \texttt{Invar}(q')$. We call (q', v') a discrete successor of (q, v). Time transitions have the form $(q, v) \xrightarrow{t} (q, v + t)$ where $t \in \mathbb{R}^+$ and $v + t \in \texttt{Invar}(q)$. We say that $(q, v + t)$ is a time successor of (q, v).

Symbolic representation. Due to the dense nature of time, the semantic graph of a TA has an infinite number of states. To perform algorithmic analysis for TA, a finite representation of this state space is needed. The symbolic representation currently used is based on the notion of zones, and leads to a symbolic graph called *simulation graph.* A zone (q, ζ) is a set of (semantic) states of a TA, such that they have the same discrete part q and the set of their valuations forms a convex polyhedron ζ. Given a zone $z = (q, \zeta)$, we note $\texttt{disc}(z)$ the discrete part q of z, and $\texttt{poly}(z)$ its polyhedron ζ. The transitions of a simulation graph are labelled by discrete actions (intuitively time elapses inside zones, and thus there are no transitions labelled by time delays). The following operations allow to compute the transitions of a simulation graph: $\texttt{time-succ}(z)$ and $\texttt{time-pred}(z)$ represent respectively the set of time successors and predecessors of some state in z, while $\texttt{disc-succ}(e, z)$ and $\texttt{disc-pred}(e, z)$ represent the set of discrete successors of some state in z, by taking transition e. The operation $\texttt{post}(e, z, c)$[2] computes the successor zone of z by taking transition e, with respect to a constant $c \in \mathbb{N}$ (in general, this constant is the greater constant appearing in the constraints of the TA), while the operation $\texttt{pre}(e, z)$ computes the predecessor zone of z by transition e.

$$\begin{aligned}
\texttt{time-succ}(z) &\overset{def}{=} \{s' \mid \exists s \in z, t \in \mathbb{R}^+\ s \xrightarrow{t} s'\}\\
\texttt{time-pred}(z) &\overset{def}{=} \{s \mid \exists s' \in z, t \in \mathbb{R}^+ \cdot s \xrightarrow{t} s'\}\\
\texttt{disc-succ}(e, z) &\overset{def}{=} \{s' \mid \exists s \in z \cdot s \xrightarrow{e} s'\}\\
\texttt{disc-pred}(e, z) &\overset{def}{=} \{s \mid \exists s' \in z \cdot s \xrightarrow{e} s'\}\\
\texttt{post}(e, z, c) &\overset{def}{=} \texttt{Approx}_c(\texttt{time-succ}(\texttt{disc-succ}(e, z)))\\
\texttt{pre}(e, z) &\overset{def}{=} \texttt{disc-pred}(e, \texttt{time-pred}(z))
\end{aligned}$$

[2] The operation `post` is used to compute the simulation graph. The use of the operator $\texttt{Approx}_c$ in its definition ensures the termination of the construction of the simulation graph. More details can be found in [8].

Consider a TA $A = \langle Q, q_0, \Sigma, X, T, \texttt{Invar}, L\rangle$ and $c \in \mathbb{N}$ a constant greater or equal to the greatest constant appearing in a constraint of A. The simulation graph of A with respect to c, written $SG(A, c)$, is a tuple $\langle Z, z_0, \Sigma, \mathcal{E}\rangle$ where Z is the finite set of states of the graph (i.e., a set of zones) and $z_0 = (q_0, \nearrow\texttt{zero} \cap \texttt{Invar}(q_0))$ is the initial zone. The set $\mathcal{E} \subseteq Z \times T \times Z$ of transitions is defined as follows: given a zone z and an edge $e \in T$, if $z' = \texttt{post}(e, z, c) \neq \emptyset$, then z' is a zone of the graph and $z \xrightarrow{e} z'$ is a transition of the graph.

Classic parallel composition operator for TA. We consider timed systems modeled in a compositional framework. Each component is modeled as a TA, and components are put together with some parallel composition operator. We consider here the classic parallel composition operator for TA. This operator, written $\|$, operates between TA with disjoint sets of clocks. It is defined as a synchronized product where synchronizations are done on actions with identical labels. Other actions interleave and time elapses synchronously between all the components. Formally, let us consider two TA $A_i = \langle Q_i, q_{0_i}, \Sigma_i, X_i, T_i, \texttt{Invar}_i, L_i\rangle$ for $i = 1, 2$, such that $X_1 \cap X_2 = \emptyset$. 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 $\Sigma_1 \cup \Sigma_2$. The set Q of locations consists of pairs (q_1, q_2) where $q_1 \in Q_1$ and $q_2 \in Q_2$. The initial location is the pair (q_{0_1}, q_{0_2}). The invariant of a location (q_1, q_2) is $\texttt{Invar}(q_1) \wedge \texttt{Invar}(q_2)$, and its label is $L(q_1) \cup L(q_2)$. The set T of edges is defined by the following rules:

Interleaving: $$\frac{(q_1,q_2)\in Q\ ,\ (q_1,g_1,a,r_1,q_1') \in T_1\ ,\ a\notin\Sigma_2}{((q_1,q_2),g_1,a,r_1,(q_1',q_2)) \in T}$$

$$\frac{(q_1,q_2)\in Q\ ,\ (q_2,g_2,a,r_2,q_2') \in T_2\ ,\ a\notin\Sigma_1}{((q_1,q_2),g_2,a,r_2,(q_1,q_2')) \in T}$$

Synchronization: $$\frac{(q_1,q_2)\in Q,\ (q_1,g_1,a,r_1,q_1') \in T_1\ ,\ (q_2,g_2,a,r_2,q_2') \in T_2}{((q_1,q_2),g_1\wedge g_2,a,r_1\cup r_2,(q_1',q_2')) \in T}$$

2.2 Simulation relations to preserve properties

Recall that we are interested in developing incrementally component-based timed systems, by integration of components. The major issue when using such a method is to ensure preservation of already checked local properties of a component, when integrating it in an environment. We defined in [5] a divergence-sensitive and stability-respecting (DS) timed τ-simulation, which ensures the preservation of linear timed properties, in particular safety, liveness and bounded-liveness ones.

Consider a component C to be integrated in an environment E, using the parallel composition operator $\|$, where each component is modeled as a timed automaton. This integration leads to a composite automaton $C\|E$, which contains new clocks and new actions comparing to C. New clocks are the clocks of E, and new actions are internal actions of E which do not synchronize with an action of C. In $C||E$, we consider such actions as being non-observable and rename them by τ. The DS timed τ-simulation is defined between the traces of $C||E$ and C and is characterized by (i) if $C||E$ can make an action of C after some amount of time, then C could also make this action after the same amount of time (clauses 1 and 2 of Definition 1), (ii) internal actions of the environment E (called τ) stutter (clause 3 of Definition 1). Note that this definition actually corresponds to the classic notion of τ-simulation, that we extend to handle time. We also add two criteria to the definition of this simulation, namely divergence-sensitivity and stability-respect. Divergence-sensitivity ensures that internal actions τ of E will not take the control forever, and stability-respect guarantees that the integration of C in E will not create new deadlocks.

In order to avoid too many definitions, we remained concise in the presentation of the simulation and focus here directly on its symbolic formal definition, which is the one implemented in the tool VeSTA. More details, as well as the definition at the semantic level, can be found in [5]. However, the following technical points used in Definition 1 must be clarified. The predicate `free`, used in the clause *stability-respect*, was defined in [8]. Informally, given a location q of a timed automaton, $\texttt{free}(q)$ is the set of all valuations (of states with q as discrete part) from which a discrete transition can be taken after some time elapsed. The formal definition is: $\texttt{free}(q) = \bigcup_{e=(q,g,a,r,q')\in T} \swarrow (g \cap ([r := 0]\texttt{Invar}(q')))$.The predicate `src_val`, used in the clause *strict simulation* is defined formally as follows: $\texttt{src\_val}(z, e, z') = \texttt{poly}(\texttt{pre}(e, z') \cap z)$. It represents the valuations of the subset of states in z which lead to states in the zone z' by taking transition e and letting time elapse.

Definition 1 (Symbolic DS timed τ-simulation). *Let $SG_1 = \langle Z_1, z_{0_1}, \Sigma_1, \mathcal{E}_1\rangle$ and $SG_2 = \langle Z_2, z_{0_2}, \Sigma_1 \cup \{\tau\}, \mathcal{E}_2\rangle$ be two simulation graphs, obtained respectively from two TA A_1 and A_2. The symbolic DS timed τ-simulation $\mathcal{Z}_{ds}$ is the greatest binary relation included in $Z_2 \times Z_1$, such that $z_2\mathcal{Z}_{ds}z_1$ if:*

1. *Strict simulation:*
$$z_2 \overset{e_2}{\to} z_2' \wedge \texttt{label}(e_2) \in \Sigma_1 \Rightarrow \exists z_1' \cdot (z_1 \overset{e_1}{\to} z_1' \wedge \texttt{label}(e_1) = \texttt{label}(e_2) \wedge \texttt{src\_val}(z_2, e_2, z_2')\rfloor_{X_1} \subseteq \texttt{src\_val}(z_1, e_1, z_1') \ \wedge \ z_2' \ \mathcal{Z}_{ds} \ z_1').$$
2. *Equality of delays and of common clocks valuations:* $\texttt{poly}(z_2)\rfloor_{X_1} \subseteq \texttt{poly}(z_1)$.
3. *τ-transitions stuttering:* $z_2 \overset{e_2}{\to} z_2' \wedge \texttt{label}(e_2) = \tau \Rightarrow z_2' \ \mathcal{Z}_{ds} \ z_1$.
4. *Stability respect:* $(\texttt{poly}(z_2)\backslash\texttt{free}(\texttt{disc}(z_2)))\rfloor_{X_1} \subseteq \texttt{poly}(z_1)\backslash\texttt{free}(\texttt{disc}(z_1)))$.
5. *Divergence sensitivity: SG_2 does not contain any non-zeno τ-cycles. A non-zeno τ-cycle is a cycle which only contains transitions labelled by τ and in which the total time elapsed goes to infinity (i.e., time diverges).*

We extend this relation to simulation graphs. Consider two simulation graphs SG_1 and SG_2, which initial zones are respectively z_{0_1} and z_{0_2}. We say that SG_1 simulates SG_2 with respect to $\mathcal{Z}_{ds}$, written $SG_2 \preceq_{\mathcal{Z}_{ds}} SG_1$, if $z_{0_2}\mathcal{Z}_{ds}z_{0_1}$.

Preservation abilities. The DS timed τ-simulation preserves all properties which can be expressed with the logic MITL (Metric Interval Temporal Logic) [7], as well as strong non-zenoness and deadlock-freedom. Formal proofs can be found in [5]. MITL is a linear timed logic, which can be viewed as the timed extension of the linear (untimed) logic LTL [11] and in which temporal operators are constrained by a time interval. Strong non-zenoness is a specific essential property of timed systems. A TA is said to be strongly non-zeno if time can diverge along each path of its semantic graph. Note that the timed τ-simulation, without divergence-sensitivity and stability-respect criteria, preserves all safety properties.

Composability. Composability is an essential property for integration of components. Indeed, it expresses that a component automatically simulates its integration with other ones. Formally, given components C and E, it means that C simulates its integration with E, i.e., the composition $C \| E$. Thus, composability can ensure the preservation of local properties of C for free (properties preserved depends on the notion of simulation which is considered).
The composability property is guaranteed with the timed τ-simulation (without divergence-sensitivity and stability-respect), when integration is achieved with the classic parallel omposition operator. This implies that safety properties are preserved for free during this integration process. However, this is not the case when considering the divergence-sensitivity and stability-respect criteria. Composability does not automatically hold. To ensure this property, the DS timed τ-simulation has to be checked algorithmically. Therefore, we implemented this verification in a tool named VESTA.

3 The tool Vesta

VESTA considers component-based timed models consisting of a set of components (modeled as timed automata) which interact using the classic parallel composition operator $\|$. Therefore, it provides graphical and textual editors to capture these elements. Then, VESTA can automatically generate composite systems, made up by parallel composition of chosen components with respect to the given interactions.

The main feature of VESTA is to check if local properties of a component are preserved when it is merged into an environment, by checking if this component simulates the composite system obtained by this merging. The simulation can be checked either in a "general way", i.e., to ensure preservation of all the local properties of a component, or "partially", i.e., for some specific given properties. This partial verification is presented in details in section 5.1. In both cases, if the simulation is not checked successfully, the tool reports the error found as well as a graphical diagnostic consisting of the trace of the composite system which is not simulated by any traces of the component, and the trace of the component it had to correspond to.

3.1 Architecture of Vesta

VeSTA was developed using both C and Java languages. Java is used for the graphical user interface, which is described in the next section, and C for the core of the tool, which is described below. The architecture of VeSTA is shown in Fig. 1. The models considered consist of three kinds of elements: the set of components (saved in .aut files) and possibly their local properties (prop) in the case of partial verification, the types of the variables used in the components and the interactions between components (sync). From this modeling, VeSTA can automatically generate composite systems by using the classic parallel composition operator between the components (.exp files). Compositions can also have local properties.

To get an efficient representation of this model, VeSTA is based on SMI[3] (Symbolic Model Interface). SMI is a powerful library providing efficient representation for finite-state models, by building an equivalent symbolic representation using decision diagrams. Note that our choice was guided by the functioning of the Open-Kronos tool [8], which is already based on SMI.

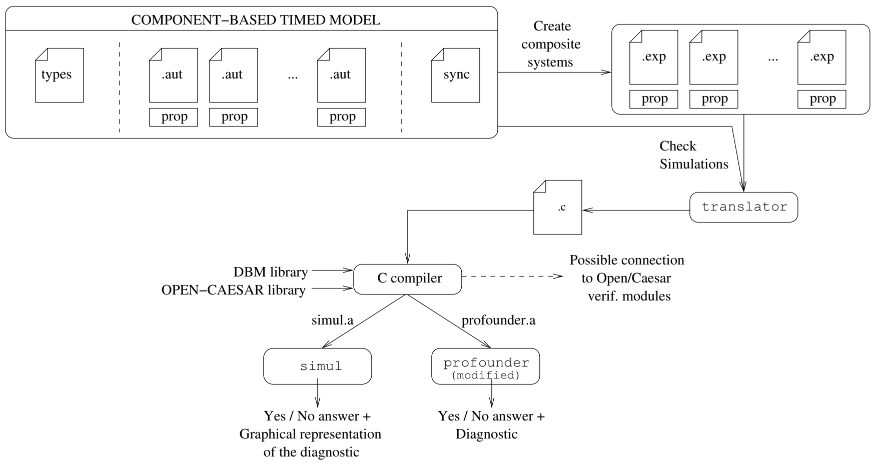

COMPONENT−BASED TIMED MODEL
types
.aut
prop
...
sync
Create composite systems
.exp
Check Simulations
translator
.c
DBM library
OPEN−CAESAR library
C compiler
Possible connection to Open/Caesar verif. modules
simul.a
profounder.a
simul
profounder (modified)
Yes / No answer + Graphical representation of the diagnostic
Yes / No answer + Diagnostic

Fig. 1. Structure of VeSTA

The core of the tool consists of two modules: `translator` and `simul`, taking as input two components, which can be composite systems (.exp files): one corresponding to a component C to be integrated in an environment E, and the other to the composite system $C||E$ obtained after having integrated C in E. `translator` creates a file .c which implements data structures and functions to generate a symbolic graph (the so-called simulation graph) for each input component. The way data structures and functions are created for $C||E$ allows it to be

[3] http://www-verimag.imag.fr/∼async/SMI/index.shtml

connected to the different modules of Open-Caesar. When this file is created, it is compiled and linked to Open-Caesar and Dbm libraries (Dbm libraries allow to manipulate the timing constraints of the model). Then, an executable `simul` is created and run to check the stability-respecting timed τ-simulation. The divergence-sensitivity part is checked thanks to an adaptation of an algorithm of the module Profounder [12] of Open-Kronos. This algorithm, as well as the one implemented in `simul`, is presented in the next section.

3.2 Algorithms

The DS timed τ-simulation is checked in two phases. The divergence-sensitive part (i.e., clause 5 of Def. 1) is checked independently with an adaptation of an algorithm of the module Profounder, which is part of the Open-Kronos tool. Then, the stability-respecting timed τ-simulation is checked in the module `simul` (i.e., clauses 1 to 4 of Def. 1). Thus, VeSTA uses two main algorithms to check the DS timed τ-simulation: one for divergence-sensitivity, and the other for the stability-respecting timed τ-simulation.

Adaptation of the module Profounder to check divergence-sensitivity. For this verification, we use the algorithm called *full DFS* (*full Depth First Search*) defined in [8, 12]. This algorithm was first designed to test the emptiness of a timed Büchi automaton, in the case of a persistent acceptance condition (i.e., from one point on, the automaton only visits accepting states). The algorithm thus consists in detecting non-zeno cycles in the automaton such that they only contain accepting states. For this, it visits all the paths of the simulation graph of the automaton, and puts them in a stack. The exploration of a path stops when reaching a state which is already in the stack (this means that an elementary cycle is found). It only remains to check that the cycle is non-zeno and only contains accepting states.

Algorithm 1 presents the adaptation of this algorithm to detect non-zeno τ-cycles, instead of non-zeno accepting cycles, in a simulation graph $SG = \langle Z, z_0, \Sigma, \mathcal{E}\rangle$, where the alphabet Σ contains the action τ. When a cycle is detected, we test if it is non-zeno and if all the transitions of the cycle are labelled by τ. The procedures `Top`, `Push` and `Pop` are classic operations on stacks, allowing to get the top of a stack, and to add and remove an element in the stack. The procedure `Part(Stack, e)` gets all the elements of the stack `Stack` added after the element `e`. The procedure `Next(Stack, e)` gets the element following `e` in `Stack` (i.e. the element added after `e`). The procedure `non_zeno` is defined as in [8] and performs a syntactic test to check if a path is non-zeno. This test consists in checking that, in the cycle, there exists a clock x which is reset at a point i of the cycle, and that x has a lower bound at a point j of the cycle. Intuitively, this allows to ensure that at least one time unit elapses at each loop in the cycle.

Algorithm 1. A *full DFS* to check divergence-sensitivity

```
divergence_sensitivity(SG){
    Stack := {z_0}
    return non_zeno_τ_cycles()
}

non_zeno_τ_cycles(){
    z := top(Stack)
    cycle := false
    while $\exists$ z $\xrightarrow{e}$ z' $\in \mathcal{E}$ and cycle = false
        if z' $\notin$ Stack then
            Push(z', Stack)
            cycle := non_zeno_τ_cycles()
            Pop(Stack)
        else
            if $\forall$z_1 $\in$ Part(Stack, z'), $\exists$z_1 $\xrightarrow{\tau}$ Next(Stack, z_1) $\in \mathcal{E}$
                                 and non_zeno(Part(Stack, z')) then
                return true
    end while

    return cycle
}
```

Note that a classic DFS is generally not sufficient to detect non-zeno τ-cycles. Indeed, this search can miss cycles. For instance, consider a simulation graph with four states (and, to simplify only τ-transitions), such that there is a zeno τ-cycle visiting the following states: $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$, and a non-zeno one $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$. A simple DFS would explore the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ and find this zeno cycle, which is not retained for divergence-sensitivity checking. Then, the search would explore the path $1 \rightarrow 4 \rightarrow 2$, and stop since the state 2 has already been visited. Thus, the non-zeno cycle is missed. The full DFS would not have missed this cycle since it explores all cycles. However, the drawback of this algorithm is its worst-case complexity: exponential in the size of the simulation graph [12]. The problem exposed above with a simple DFS comes from zeno cycles. For strongly non-zeno simulation graphs (i.e., which do not contain any zeno path), a simple DFS (linear in the size of the graph) is sufficient.

Checking the stability-respecting timed τ-simulation in the module `simul`. Algorithm 2 checks the symbolic stability-respecting timed τ-simulation between two simulation graphs $SG_1 = \langle Z_1, z_{0_1}, \Sigma_1, \mathcal{E}_1 \rangle$, with set of clocks X_1, and $SG_2 = \langle Z_2, z_{0_2}, \Sigma_1 \cup \{\tau\}, \mathcal{E}_2 \rangle$. Formally, it checks that $SG_2 \preceq_{\mathcal{Z}_{ds}} SG_1$, without the divergence-sensitivity clause. This verification is in $\mathcal{O}((|Z_1|+|\mathcal{E}_1|)\times(|Z_2|+|\mathcal{E}_2|))$. The algorithm is cut in four parts, the main one being `verification_`$\mathcal{Z}_{\mathtt{ds}}$. A procedure `verif_`$\mathcal{Z}$`_and_stability_respect` performs a joint depth-first search of SG_2 and SG_1, and at each step of the search, it checks clauses 1 to 4 of Def. 1. A set `Visited` records the already visited pairs of zones in relation, and a stack `Stack` contains the currently checked pairs of zones. This stack also allows to return diagnostics when the verification fails.

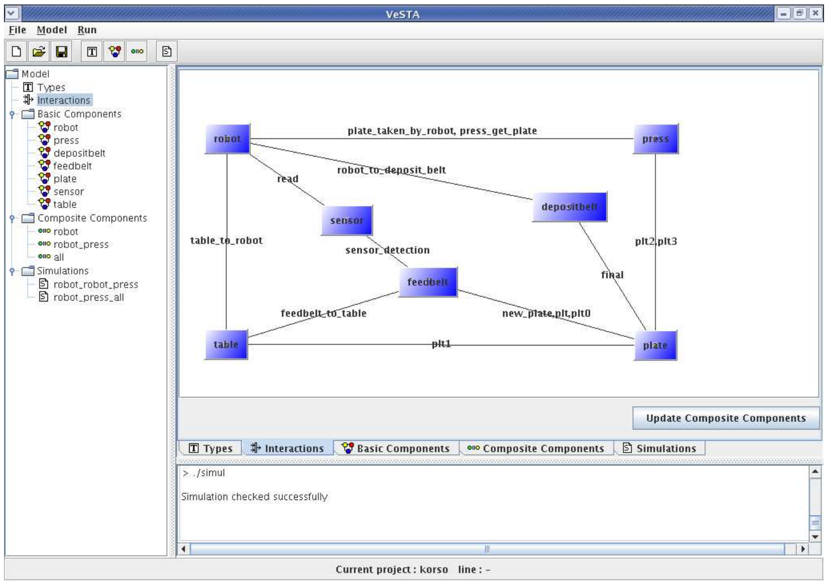
Algorithm 2. Verification of the symbolic ds timed τ-simulation

```
verification_$\mathcal{Z}_{ds}$(SG$_2$, SG$_1$){
    if (divergence_sensitivity(SG$_2$)) then
        return false
    else
        Stack := $\{(z_{0_2}, z_{0_1})\}$
        Visited := $\emptyset$
        return verif_$\mathcal{Z}$_and_stability_respect()
}

verif_$\mathcal{Z}$_and_stability_respect(){
    simul_ok := true
    ($z_2, z_1$) := top(Stack)
    if delays_equality($z_2, z_1$) $\wedge$ stab_respect($z_2, z_1$) then
        while $\exists$ a transition $z_2 \xrightarrow{e_2} z_2'$ in $\mathcal{E}_2$ and simul_ok = true
            if label($e_2$) $\in \Sigma_1$ then
                if $\exists z_1 \xrightarrow{e_1} z_1'$ s.t. label($e_1$) = label($e_2$) $\wedge$
                    strict_simulation($z_1, e_1, z_1', z_2, e_2, z_2'$) = true then
                    if ($z_2', z_1'$) $\notin$ Visited and ($z_2', z_1'$) $\notin$ Stack
                        Push(($z_2', z_1'$), Stack)
                        simul_ok := verif_$\mathcal{Z}$_and_stability_respect()
                        Pop(Stack)
                else
                    return false
            else
                if ($z_2', z_1$) $\notin$ Visited and ($z_2', z_1$) $\notin$ Stack then
                    Push(($z_2', z_1$), Stack)
                    simul_ok := verif_$\mathcal{Z}$_and_stability_respect()
                    Pop(Stack)
        end while
    else
        return false

    if simul_ok = true then Visited := Visited $\cup$ $\{(z_2, z_1)\}$
    return simul_ok
}

strict_simulation ($z_1, e_1, z_1', z_2, e_2, z_2'$){
    return (src_val($z_2, e_2, z_2'$)$\rfloor_{X_1}$ $\subseteq$ src_val($z_1, e_1, z_1'$))
}

stab_respect ($z_2, z_1$){
    return (poly($z_2$)\free(disc($z_2$)))$\rfloor_{X_1}$ $\subseteq$ poly($z_1$)\free(disc($z_1$))
}

delays_equality ($z_2, z_1$){
    return (poly($z_2$)$\rfloor_{X_1}$ $\subseteq$ poly($z_1$))
}
```

3.3 Graphical User Interface

The GUI of VeSTA is shown in Fig. 2. The tree on the left is an explorer to navigate between the elements of the model, the generated assembling of components, and the results of already checked preservations (i.e., simulations). The bottom-right part is a log window, displaying informations such as syntax errors or summarized results of preservation checkings. The top-right part is the main element of the GUI, with five tabs:

- the *Types* tab displays the types of the variables used in the model,
- the *Interactions* tab shows the interactions between the components,
- the *Basic Components* tab contains all the components of the model,
- the *Composite Components* tab contains the assembling of components,
- the *Simulations* tab contains results for each already checked preservation.



```
> ./simul

Simulation checked successfully
```

Current project : korso   line : -

Fig. 2. Graphical User Interface of VeSTA

The menubar and toolbar provide buttons to treat a new model. They allow to create new components, import components from another model, choose components to put together and automatically create the assembling, and check simulations. The interactions between components can be created graphically via the *Interactions* tab. Components (i.e. timed automata[4]) are described through a textual editors, with a simple language which consists in giving the invariant of each location, and the transitions of the component (name, source and target location, guard, reset and, possibly, update of some variables).

[4] Actually, VeSTA considers *extended* timed automata, which can be equiped with boolean, bounded-integer and enumerative-type variables. However, the use of these variables is restricted to a local use for the components (no shared variables).

4 Vesta in practice: incremental verification of a production cell

The tool VESTA allowed us to show the interest of incremental development by integration of components, formalized by the DS timed τ-simulation, in comparison to a direct verification on the complete model of the system. We present in this section a case study concerning a production cell[5]. This 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 treated it with timed automata, as it was in [14].

Presentation of the case study The production cell contains six devices, as shown in Fig. 3: a feed-belt equipped with a sensor, a deposit-belt, an elevating-rotary table, a two-arms robot and a press. It also contains one or several pieces to be treated. Our modeling of the cell follows the one of [14].

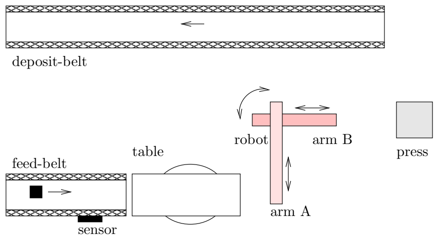
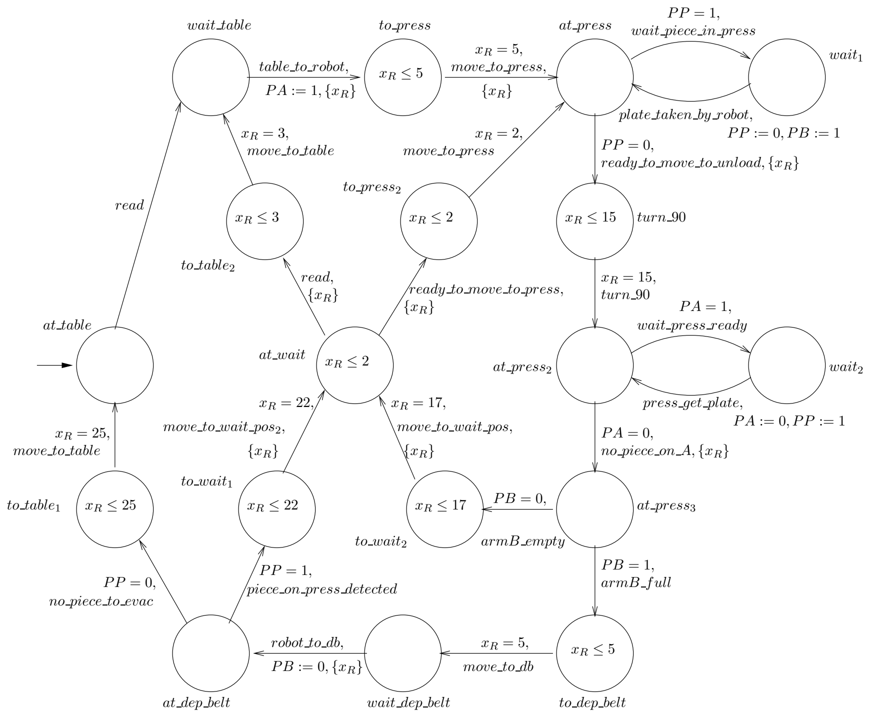


Fig. 3. The Production Cell Example

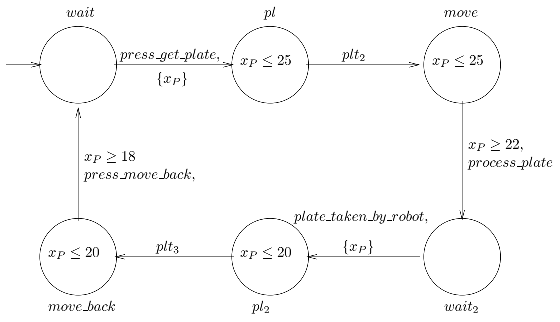
Description of the production cell. A simplified functioning of the cell is the following. Pieces arrive on the feed-belt. The sensor detects when a piece is introduced in the cell, and sends a message to the robot to inform that the piece is going to be available. When it arrives at the end of the belt, it is transferred to the table, which goes up and turns until being in an adequate position to give to the robot the possibility to take it. The robot turns 90° so that its arm A can pick the piece up, and then puts it in the press which processes it. When the treatment is finished, the piece is taken by the arm B of the robot, which transports it to the deposit-belt where it is evacuated. The behavior of each device depends on timing constraints and is modeled by a timed automaton. In the sequel, we focus particularly on local properties concerning the robot, and the assembling robot||press. Fig. 4 and 5 show respectively the timed automata modeling the robot and the press.

[5] The detailed results for this case study, as well as other experimentations, can be found in [13].



wait_table
to_press
at_press
$PP = 1$, wait_piece_in_press
$wait_1$
table_to_robot, $PA := 1, \{x_R\}$
$x_R \leq 5$
$x_R = 5$, move_to_press, $\{x_R\}$
plate_taken_by_robot, $PP := 0, PB := 1$
$x_R = 3$, move_to_table
$x_R = 2$, move_to_press
$PP = 0$, ready_to_move_to_unload, $\{x_R\}$
read
to_press_2
$x_R \leq 3$
$x_R \leq 2$
$x_R \leq 15$
turn_90
to_table_2
read, $\{x_R\}$
ready_to_move_to_press, $\{x_R\}$
$x_R = 15$, turn_90
at_table
$PA = 1$, wait_press_ready
at_wait
$x_R \leq 2$
at_press_2
$wait_2$
$x_R = 22$, $move\_to\_wait\_pos_2$, $\{x_R\}$
$x_R = 17$, move_to_wait_pos, $\{x_R\}$
press_get_plate, $PA := 0, PP := 1$
$x_R = 25$, move_to_table
$PA = 0$, no_piece_on_A, $\{x_R\}$
to_table_1
$x_R \leq 25$
to_wait_1
$x_R \leq 22$
$x_R \leq 17$
$PB = 0$, armB_empty
at_press_3
to_wait_2
$PP = 0$, no_piece_to_evac
$PP = 1$, piece_on_press_detected
$PB = 1$, armB_full
robot_to_db, $PB := 0, \{x_R\}$
$x_R = 5$, move_to_db
$x_R \leq 5$
at_dep_belt
wait_dep_belt
to_dep_belt

Fig. 4. Timed Automaton of the robot



wait
pl
move
press_get_plate, $\{x_P\}$
$x_P \leq 25$
plt_2
$x_P \leq 25$
$x_P \geq 18$ press_move_back,
$x_P \geq 22$, process_plate
plate_taken_by_robot, $\{x_P\}$
$x_P \leq 20$
plt_3
$x_P \leq 20$
move_back
pl_2
$wait_2$

Fig. 5. Timed Automaton of the press

Some local properties. We identified seven main properties of the robot: two safety properties (called P_1 and P_2), two liveness (response) properties (P_3 and P_4) and three bounded liveness properties (P_5, P_6 and P_7)[6]. We also identified a main liveness property (P_8) ensuring the correct functioning of the robot and the press when they are put together. We express these properties in Mitl[7].

[6] Under some conditions, safety properties express that *something bad will not happen*, liveness ones that *something expected will eventually happen* and bounded liveness ones that *something expected will eventually happen within some bounded delay.*

[7] The detailed expressions of these properties can be found in [13].

Our objective is to compare direct verification and incremental verification by integration of components, for MITL properties. The first method consists in assembling all the components, and then to check properties P_1 to P_8 on the complete model obtained. The second method consists in checking these properties locally only on the components they concern, and then to ensure they are preserved when these components are integrated in their environment. That is, properties P_1 to P_7 are checked on the robot component, and property P_8 on the assembling robot$\|$press. Then, the preservation of P_1 to P_7 must be ensured when the robot component is integrated with the press component. The preservation of P_8 must be guaranteed when this assembling is integrated with all the other components of the system. In this way, each locally checked property will hold on the complete model, since preservation is checked thanks to the DS timed τ-simulation, which is a preorder, and thus, is a transitive relation.

Experimental results for the production cell. First note that VeSTA is not a model-checker. Thus to check the properties locally and globally, we used the model-checker KRONOS[15]. KRONOS is a verification tool for timed systems which performs TCTL model-checking [16]. TCTL is a logical formalism that allows to express branching-time properties. Even if we do not consider branching-time properties, we can use it for this example since the MITL properties we consider can also be expressed in TCTL[8]. It turns out that the local and global verification of all the properties, achieved with KRONOS, succeeded. VeSTA allows to ensure the preservation of locally established properties. Therefore, it is first used to check that the local properties of the robot are preserved when it is combined with the press, and then that the property of the assembling robot$\|$press is preserved when these components are integrated with the rest of the components of the cell and one piece. In both cases, the verification succeeded, and thus, the preservation of the MITL properties P_1 to P_8 is guaranteed (as well as the preservation of strong non-zenoness and deadlock-freedom).

Fig. 6 presents the results obtained on the example, by comparing incremental verification by integration of components to direct verification. We compared the time consumed to perform this direct verification on the whole model (column "Global Verification") and the time spent to achieve incremental verification, i.e., local verification and preservation checking. It turns out that, even if the computation times are still acceptable, direct verification consumes much more time (almost 20 seconds) than incremental verification when preservation is achieved with VeSTA (less than one second).

Diagnostics. In section 3, we stated that VeSTA has the ability to provide diagnostics when the verification of the DS timed τ-simulation (and thus of the preservation) fails. To show this functionality, we slightly modify the automaton of the press. We add a guard (for instance $x_p \leq 40$) to the transition

[8] To our knowledge, there is no tool performing MITL model-checking.

Property	Global Verification (Kronos)	Local Verification (Kronos)	Preservation checking (VeSTA)
P_1 (safety)	0.01	< 0.001	0.05
P_2 (safety)	0.01	< 0.001	
P_3 (liveness)	0.98	$< 0, .001$	
P_4 (liveness)	15.79	0.04	
P_5 (bounded liveness)	0.68	< 0.001	
P_6 (bounded liveness)	0.48	< 0.001	
P_7 (bounded liveness)	0.7	< 0.001	
P_8 (liveness)	0.93	0.02	0.46
Total	19.58	0.06	0.51

Fig. 6. Comparison of the local and global verification times (in seconds)

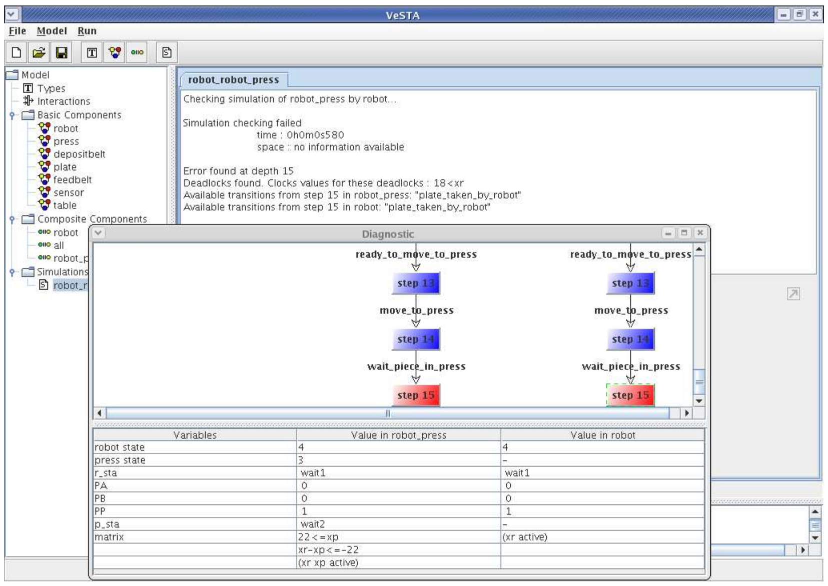
plate_taken_by_robot, which means that the press expects to be unloaded by the robot at most 40 time units after having received a piece. This modification prevents the preservation from being established, when integrating the robot with the press. Indeed, adding this guard introduces a deadlock in the assembling press||robot, which did not exist in the robot component alone. Thus, deadlock-freedom is obviously not preserved. Moreover, Mitl properties P_3 to P_7 are also not preserved since non-introduction of deadlocks is precisely one of the conditions which define the DS timed τ-simulation (clause *stability-respect*), and thus, which ensure the preservation of Mitl properties. Note that properties P_1 and P_2 are still preserved since they are safety properties and, therefore, their preservation does not need neither stability-respect, nor divergence-sensitivity. The graphical diagnostic provided by VeSTA helps detecting where the deadlock is introduced, by showing the trace of the assembling robot||press where the deadlock appears, and the corresponding trace of the component robot that had to simulate it, with respect to the DS timed τ-simulation. Fig. 7 shows how diagnostics are displayed in VeSTA.

5 Additional features

The main functionality of VeSTA is to check the DS timed τ-simulation, using exactly algorithms 1 and 2. In addition, VeSTA proposes an interesting additional feature, consisting in verifying partially the relation to ensure the preservation of some specific given Mitl properties. This kind of verification, as well as the motivations, are explained below.

5.1 Partial verification of the preservation

Let us go back to the second version of the production cell example, in which we modified the guard of the edge *plate_taken_by_robot* in the automaton of the press. As we explained, the deadlock introduced in the assembling robot||press prevents ensuring the preservation of all the properties which can be expressed for the robot, since the verification of the simulation fails. However, the specified local properties of the robot may be preserved. For this reason, we improved



```
Checking simulation of robot_press by robot...

Simulation checking failed
                time : 0h0m0s580
                space : no information available

Error found at depth 15
Deadlocks found. Clocks values for these deadlocks : 18<xr
Available transitions from step 15 in robot_press: "plate_taken_by_robot"
Available transitions from step 15 in robot: "plate_taken_by_robot"
```

Variables	Value in robot_press	Value in robot
robot state	4	4
press state	3	–
r_sta	wait1	wait1
PA	0	0
PB	0	0
PP	1	1
p_sta	wait2	–
matrix	22<=xp	(xr active)
	xr-xp<=-22	
	(xr xp active)	

Fig. 7. Diagnostic provided by VeSTA

VeSTA by giving the possibility to the user to specify the properties to preserve, and to check the preservation only for these properties (instead of a "global preservation"). This is what we call *partial verification of the preservation*.

Thus, the objective of such a verification is to guarantee the preservation of specified local properties of a component, rather than the preservation of all the properties which could be potentially specified. Until now, this functionality is only available for response properties of the form $\Box(p \Rightarrow \Diamond q)$. The reasoning for this kind of verification is the following. In most cases, the verification of the simulation fails due to an introduction of deadlocks (i.e., the clause *stability-respect* of the simulation does not hold). Consider now a component C, a local property $L = \Box(p \Rightarrow \Diamond q)$ of C and an environment E in which C must be integrated. A path π in $C \| E$, in which a deadlock is introduced comparing to the path of C which simulates it, makes the verification of the preservation fail. However, if this path π does not concern L, then the preservation should be guaranteed.

Let us detail how this partial verification is achieved, for a response property of the form $\Box(p \Rightarrow \Diamond q)$. To ensure the preservation of such a property, we must guarantee that, when p is encountered in a path π, this path is not *cut* (by the introduction of a deadlock) before q is reached. Thus, the partial verification consists in checking this non-introduction of deadlock (i.e., the clause *stability-respect*) only for the states of π located between the state satisfying p, and the one satisfying q. Fig. 8 illustrates the principle of this verification for this kind of property. A path is represented, and the states on which stability-respect must be checked are put in grey. Note also that divergence-sensitivity must also be checked for these states, to ensure that the path is not cut by means of the introduction of an infinite sequence of non-observable actions. The verification for divergence-sensitivity consists in checking that these states are not part of a non-zeno cycle only containing non-observable actions.

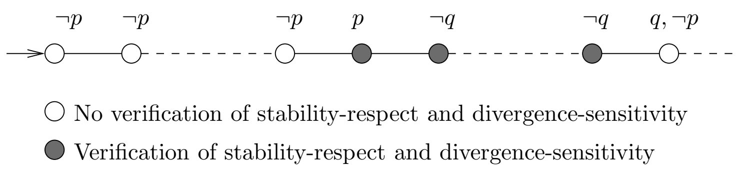

$\neg p$
$\neg p$
$\neg p$
p
$\neg q$
$\neg q$
$q, \neg p$
No verification of stability-respect and divergence-sensitivity
Verification of stability-respect and divergence-sensitivity

Fig. 8. Partial verification for a property of the form $\Box(p \Rightarrow \Diamond q)$

Thus, VeSTA gives the possibility to specify the local properties of C, of the form $\Box(p \Rightarrow \Diamond q)$, which must be preserved, and to verify the simulation only to guarantee the preservation of these properties. Note that, contrary to the classic verification, this partial verification does not guarantee the preservation of strong non-zenoness and deadlock-freedom.

5.2 Connection to other platforms and tools

Another interesting point of VeSTA is the following. Recall that the way the tool was designed was inspired by the Open-Kronos tool. In particular, the syntax used to describe the components, and the symbolic representation of these models, is identical to the one in Open-Kronos. Thus, a direct consequence of this design choice is that models considered in VeSTA can be connected to the Open-Kronos tool. Connection to the Open-Caesar verification platform is also possible as another direct consequence, since this connection was already available from Open-Kronos models.

The connection to Open-Kronos is particularly interesting. Indeed, the ability to connect VeSTA models to Open-Kronos could allow to check Mitl properties directly on the models considered in VeSTA. Recall that, now, we use the tool Kronos to perform model-checking, since there exist no tools for Mitl model-checking. Thus, as Kronos is a Tctl model-checker, we are restricted to Mitl properties which can also be expressed in Tctl. Moreover, VeSTA models must be translated into Kronos syntax. The Open-Kronos tool can perform reachability analysis, but can also test timed Büchi automata (TBA) emptiness. Mitl properties can be translated into TBA which recognize the same language. Thus, with a translator from Mitl to TBA (such translators do not exist yet) and an implementation of the composition of TA with TBA (see [8] to get more details about this special composition), it would be possible to directly connect VeSTA models of components to Open-Kronos, perform Mitl model-checking on these components, and then check with VeSTA the preservation of these properties during the integration of these components.

6 Conclusion and further developments

In this paper, we presented the tool VeSTA, which allows (i) to model incrementally a component-based timed system, by integration of components, and (ii) to ensure the preservation of established local properties of the components on the complete model, instead of performing a direct verification of these properties on this complete model. Timed components are modeled as timed automata, and integration is achieved thanks to the classic parallel composition operator for timed automata. Preservation is checked by means of a divergence-sensitive and stability-respecting timed τ-simulation. Precisely, a successful verification of this relation ensures the preservation of all linear timed properties expressed with the logic MITL, strong non-zenoness and deadlock-freedom.

The first results obtained for incremental verification by integration of components, using VeSTA for the preservation part, are encouraging. On the production cell case study of [14], it turns out that a direct verification consumes almost 20 seconds of computation time, while the incremental one based on preservation needs less than one second. Other experiments showed that VeSTA can handle models up to 400000 symbolic states. Beyond this number, we had not enough memory for the verification of the preservation to be run to completion (on a PC with 1Gb memory). Nevertheless, this number has to be relativized with respect to the number of clocks of the model, which is a direct cause of great memory consumption: 15 clocks for the model from which we obtained this upper bound. Thus, further improvements will be dedicated to handle this limitation, by implementing abstractions such as the *active-clock reduction* [17, 18], allowing to ignore clocks in states where they are inactive.

Another further development concerns the partial verification of the DS timed τ-simulation. The objective of such a verification is to check preservation only for the local properties which are specified for the components, instead of ensuring the preservation of all properties which could potentially be expressed. Until now, this partial verification is only available for response properties of the form $\Box(p \Rightarrow \Diamond q)$. It seems interesting to extend this kind of verification to other patterns of liveness and bounded-liveness properties. Moreover, this kind of verification could optimize computation times. Indeed, recall that partial verification consists, in particular, in checking the stability-respecting part of the simulation only on some specific states, instead of checking it systematically. As stability-respect is checked by means of high-cost operations, such as polyhedra complementation, it is essential to avoid as much as possible to check this clause. Thus, generalizing partial verification could lead to better performances in terms of computation times to check preservation.

ADDITIONAL INFORMATIONS. More informations on VeSTA can be found in its complete documentation and user guide at the following URL:
`http://lifc.univ-fcomte.fr/publis/papers/pub/2006/RT2006-01.pdf`.

Acknowledgments. We would like to thank Stavros Tripakis for having sent to us the distribution of Open-Kronos, particularly the module Profounder and some (useful !) source files. Thanks also for the time spent answering questions.

References

1. Bellegarde, F., Julliand, J., Kouchnarenko, O.: Ready-simulation is not Ready to Express a Modular Refinement Relation. In: Proc. of FASE'00. Volume 1783 of LNCS., Berlin, Germany, Springer-Verlag (2000) 266–283
2. Henzinger, M., Henzinger, T., Kopke, P.: Computing simulations on finite and infinite graphs. In: Proc. of FOCS'95. (1995) 453–462
3. Tasiran, S., Alur, R., Kurshan, R., Brayton, R.: Verifying Abstractions of Timed Systems. In: Proc. of CONCUR'96. Volume 1119 of LNCS., Pisa, Italy, Springer-Verlag (1996) 546–562
4. Jensen, H., Larsen, K., Skou, A.: Scaling up Uppaal : Automatic verification of real-time systems using compositionnality and abstraction. In: Proc. of FTRTFT'00, London, UK, Springer-Verlag (2000) 19–30
5. Bellegarde, F., Julliand, J., Mountassir, H., Oudot, E.: On the contribution of a τ-simulation in the incremental modeling of timed systems. In: Proc. of FACS'05. Volume 160 of ENTCS., Macao, Macao, Elsevier (2005) 97–111
6. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
7. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctuality. Journal of the ACM **43** (1996) 116–146
8. Tripakis, S.: The analysis of timed systems in practice. PhD thesis, Universite Joseph Fourier, Grenoble, France (1998)
9. Garavel, H.: OPEN/CAESAR: An Open Software Architecture for Verification, Simulation and Testing. In Steffen, B., ed.: Proc. of TACAS'98, Lisboa, Portugal (1998)
10. Bouyer, P.: Untameable Timed Automata ! In: Proc. of STACS'03. Volume 2607 of LNCS., Berlin, Germany, Springer-Verlag (2003) 620–631
11. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18^{th} IEEE Symposium on Foundations Of Computer Science. (1977) 46–77
12. Tripakis, S., Yovine, S., Bouajjani, A.: Checking Timed Büchi Automata Emptiness Efficiently. Formal Methods in System Design **26** (2005) 267–292
13. Bellegarde, F., Julliand, J., Mountassir, H., Oudot, E.: Experiments in the use of τ-simulations for the components-verification of real-time systems. In: Proc. of SAVCBS'06, Portland, Oregon, USA (2006) Also available on ACM Digital Library.
14. Burns, A.: 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** (2003) 135–152
15. Yovine, S.: Kronos: A verification tool for real-time systems. Journal of Software Tools for Technology Transfer **1** (1997) 123–133
16. Alur, R., Courcoubetis, C., Dill, D.: Model-Checking in Dense Real-time. Information and Computation **104** (1993) 2–34
17. Daws, C., Yovine, S.: Reducing the number of clock variables in timed automata. In: Proc. of RTSS'96, IEEE Computer Society Press (1996)
18. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Proc. of TACAS'98. Volume 1384 of LNCS., Lisbon, Portugal, Springer-Verlag (1998) 313–329