Improving the Verification of Timed Systems Using Influence Information

Víctor Braberman[1]*, Diego Garbervetsky[1], and Alfredo Olivero[2]**

[1] Computer Science Department, FCEyN, Universidad de Buenos Aires, Argentina
vbraber@dc.uba.ar
dg2y@dc.uba.ar
[2] Department of Information Technology, FIyCE, Universidad Argentina de la Empresa, Argentina
aolivero@uade.edu.ar

Abstract. The parallel composition with observers is a well-known approach to check or test properties over formal models of concurrent and real-time systems. We present a new technique to reduce the size of the resulting model. Our approach has been developed for a formalism based on Timed Automata. Firstly, it discovers relevant components and clocks at each location of the observer using influence information. Secondly, it outcomes an abstraction which is equivalent to the original model up to branching-time structure and can be treated by verification tools such as KRONOS [12] or OPENKRONOS [23]. Our experiments suggest that the approach may lead to significant time and space savings during verification phase due to state space reduction and the existence of shorter counterexamples in the optimized model.

1 Introduction

In formal models of concurrent systems, safety and liveness requirements are commonly expressed in terms of virtual components (Observers) which are composed in parallel with the set of components that constitutes the system under analysis (*SUA*) (e.g., [1]). It is also true that the *SUA* may comprise some irrelevant activities for the evolution of a given observer. In this work the *SUA* and the observer are specified basically in terms of sets of communicating Timed Automata (TAs) [3]. We present a syntactical technique that transforms the original model into an equivalent one up to TCTL formulas [2] over the observer.

Our technique statically calculates, for each location of the observer, a set of components and clocks whose activity may influence the observed evolution of the *SUA*. Then it obtains an optimized system that activates and deactivates components and clocks according to that information about relevance of clocks and components. Experiments show that model-checking tools like KRONOS [12] or OPENKRONOS [23] produce a noticeable smaller state space and shorter counterexamples when fed with the optimized model.

* Research supported by UBACyT grants X156 and TW72.
** Research supported by UADE grant ING6-01

1.1 Related Work

The closest related work on automatic syntactic preprocessing of timed models is the clock reduction technique for TAs presented in [13]. Similarly to our approach, this technique examines timed-components at a syntactic level to derive reductions that preserve the branching-time structure. There is also a limited use of timing information (clocks are variables) to keep the preprocessing as light as possible. However, our technique includes an "activity calculus" that can be applied on a *SUA* given as a parallel composition (i.e., not-already composed) and the optimization also implies the deactivation of irrelevant components (not just clocks) during the possibly on-the fly verification step. It is also worth mentioning a method for the computation of delays on circuits [22], that uses topological structure of the circuit to make a partition of the network and a corresponding symbolic representation of the state space. The idea of ignoring some parts of the model while performing the verification also appears in [10], where dynamic transition relations are introduced in the setting of backward search for models used in the design of synchronous hardware. In essence, all these techniques are related to the cone of influence abstraction [11] and slicing techniques ([18, 5], etc.). Our technique can be understood as "slicing" the synchronized Cartesian product[1] without building it, instead of slicing each component, as previous works on slicing concurrent descriptions. By adapting static analysis ideas to the timed model setting, we developed a method for discovering a set of variables (standing for clocks and components) that can be safely eliminated at different locations of the final composed model. In particular, we need to know which variables may influence the future behavior of the observer at each location. Therefore, the granularity level of the static analysis over the Cartesian product is defined by the set of control locations of the observer. To obtain an optimized model that activates and deactivates those variables we perform a component-wise transformation. We do not resort to the concept of slicing criterion, instead the correctness of the procedure is stated in terms of the preservation of the branching structure of the transition system up to the propositional assignment given over the observer. This implies that our optimized model validates the same set of TCTL formula over the observer. Mentioned works on slicing concurrent models provide abstractions that either just preserve the linear-time structure of the system or are strongly bisimilar w.r.t. to the original one, a stronger condition than ours.

Our technique can also be understood as a way to perform a kind of selective search (avoiding transitions of deactivated components). Unlike partial order methods [16,21], etc., our notions are not based on independence but on influence. In fact, transitions eliminated in the optimized model, in general, are not independent to the remaining ones. Moreover, runs of our optimized model are subruns of the original one (i.e., irrelevant activity is not shown). Partial order methods could be applied in an orthogonal fashion.

[1] If synchronized Cartesian product is though as a non-deterministic guarded command program, traditional program slicing techniques seems to be of little help since all variables are likely to be classified as live.

In [7] we explain several differences between our technique and compositional reduction techniques that work over symbolic state space like [20] and [24].

Structure of the paper: In the next section we recall Timed Automata and some related notions. In Sect. 3 we present an extension of TAs that are the basis of our method: the I/O Timed Components. In this extension the labels are classified as input/output and that "uncontrollable/controllable" division of labels greatly helps to a better understanding of behavioral influence between automata. In Sect. 4 we show the rules that define the relevance of clocks and components, how to build the correct and complete abstraction, and finally some empirical results. The paper is summed up with conclusions.

2 Timed Automata

Timed Automata (TAs) has become one of the most widely used formalism to model and analyze timed systems and is supported by several tools (e.g., [12,4, 20,19], etc.). This presentation partially follows [27]. Given a finite set of clocks (non-negative real variables) $X = \{x_1, x_2, \ldots, x_n\}$ a *valuation* is a total function $v : X \overset{tot}{\rightarrow} I\!R_{\geq 0}$ where $v(x_i)$ is the value associated with clock x_i. We define V_X as the set $[X \overset{tot}{\rightarrow} I\!R_{\geq 0}]$ of total functions mapping X to $I\!R_{\geq 0}$ and $\mathbf{0} \in V_X$ denotes the function that evaluates to 0 all clocks. Given $v \in V_X$ and $t \in I\!R_{\geq 0}$, $v + t$ denotes the valuation that assigns to each clock $x \in X$ the value $v(x) + t$. Given a set of clocks $\alpha \subseteq X$ and a valuation v we define $Reset_\alpha(v)$ as a valuation that assigns zero to clocks in α and keeps the same value than v for the remaining clocks. Given a set of clocks X we define the sets of clock constraints Ψ_X according to the grammar: $\Psi_X \ni \psi ::= x \prec c | x - x' \prec c | \psi \wedge \psi | \neg \psi$, where $x, x' \in X, \prec \in \{<, \leq\}$ and $c \in I\!N$. A valuation $v \in V_X$ satisfies $\psi \in \Psi_X$ ($v \models \psi$) iff the expression evaluates **true** when each clock is replaced with its current value specified in v.

Definition 1 (Timed Automata (TAs)). *A timed automaton (TA) is a tuple $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ where S is a finite set of locations, X is a finite set of clocks, Σ is a set of labels, E is a finite set of edges, (each edge $e \in E$ is a tuple $\langle s, a, \psi, \alpha, s' \rangle$ where: $s \in S$ is the source location, $s' \in S$ is the target location, $a \in \Sigma$ is the label, $\psi \in \Psi_X$ is the guard, $\alpha \subseteq X$ is the set of clocks reset at the edge), $I : S \overset{tot}{\rightarrow} \Psi_X$ is a total function associating with each location a clock constraint called location's Invariant, and $s_0 \in S$ is the initial location.*

Given a TA $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ we define $Locs(A) = S$, $Clocks(A) = X$, $Labels(A) = \Sigma$, $Edges(A) = E$, $Inv(A) = I$, $Init(A) = s_0$, and given an edge $e = \langle s, a, \psi, \alpha, s' \rangle \in E$ we define $src(e) = s$, $lab(e) = a$, $grd(e) = \psi$, $rst(e) = \alpha$, $tgt(e) = s'$. The *State Space* Q_A of A is the set of states $(s, v) \in S \times V_X$ for which $v \models I(s)$ and $q_0 = (Init(A), \mathbf{0})$ is its *initial state*. Given a state $q = (s, v)$ we denote: $q + t = (s, v + t)$, $q^{@} = s$, and $q(x_i) = v(x_i)$. The semantics of A can be given in terms of the *Labeled Transition System* (LTS) of A, denoted $G_A = \langle Q_A, q_0, \mapsto^l_t, \Sigma \rangle$. The relation $\mapsto^l_t$ is the set of (time or discrete) transitions between states. Let $t \in I\!R_{\geq 0}$; the state (s, v) has a *time transition* to $(s, v + t)$ denoted $(s, v) \mapsto^\lambda_t (s, v + t)$ if for all $t' \leq t$, $v + t' \models I(s)$. Let $e \in E$ be an edge;

the state $(src(e), v)$ has a *discrete transition* to the state $(tgt(e), v')$ denoted $(src(e), v) \mapsto_0^{lab(e)} (tgt(e), v')$ if $v \models grd(e)$ and $v' = Reset_{rst(e)}(v)$; e is called the associated edge. We will say that $q \mapsto_t^l$ if there exists q' such that $q \mapsto_t^l q'$. A *run* r of A starting at q is an infinite sequence $q \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} \ldots$ of states and transitions in G_A. The set of runs starting at q is denoted as $R_A(q)$. The *time of occurrence* of the n^{th} transition is equal to $\sum_{i=o}^{n-1} t_i$ and is denoted as $\tau_r(n)$. A *position* in r is a pair $(i, t) \in I\!N \times I\!R_{\geq 0}$ such that $t \leq t_i$. We call L(r) the set of all labels in the run r and $\Pi(r)$ the set of all positions of run r. The time of the position $(i, t) \in \Pi(r)$, denoted $\tau_r((i, t))$ is defined as $\tau_r(i) + t$. Given $(i, t) \in \Pi(r)$ its associated state is $r((i, t)) = q_i + t$. A *divergent run* is a run such that $\sum_{i=o}^{\infty} t_i = \infty$. The set of divergent runs of a TA A starting at state q is denoted $R_A^{\infty}(q)$. A *finite run* starting at state q is simply a finite prefix of a run starting at q. A TA is *non-zeno* when any finite run starting at the initial state can be extended to a divergent run that is, the set of finite runs is equal to the set of finite prefixes of divergent runs. Since we will deal with non-zeno TAs, we say that the state q is *reachable* if there is a finite run starting at the initial state which ends at q; we denote the set of reachable states as $Reach(A)$. Usually, A has associated a mapping $\mathcal{P} : Locs(A) \mapsto 2^{Props}$ which assigns to each location a subset of the set of propositional variables ($Props$). The parallel composition of TAs is defined over classical synchronous product of automata. Given a set $\mathcal{I} = \{0..n\}$, we denote $\|_{i \in \mathcal{I}} A_i$ the parallel composition of an indexed set of TA. If q is a state of that parallel composition $\Pi_i(q)$ will denote the local state of TA A_i (locations and local-clock values).

2.1 Continuous Observational Bisimulations

The theoretical notion that supports the correctness of our abstraction mechanisms is a bisimulation relation extending "branching bisimulation" (e.g., [14]) to timed systems[2]. This notion is weaker than strong timed bisimulation, however it still preserves branching structure (TCTL logic validity) unlike weaker versions of model bisimulations (e.g. the observational timed bisimulation of [25]).

Definition 2 (Observationally-τ transition). *Given a TA A, its associated LTS $G_A = \langle Q_A, q_0, \mapsto_t^l, \Sigma \rangle$, a relation $B \subseteq Q_A \times Q_A$ between states, two states p and q such that $(p, q) \in B$, and $t \in I\!R_{\geq 0}$ such that $p \mapsto_t^{\lambda}$, we say that the state q has an observationally-τ transition w.r.t. B and p, of duration t to q_n, denoted $q \xrightarrow{B,p}_t q_n$ iff there is a finite run $r = q \mapsto_{t_0}^{l_0} q_1 \mapsto_{t_1}^{l_1} \ldots q_n$ such that $\tau_r(n) = t$, and for every position $k \in \Pi(r)$, $(p + \tau_r(k), r(k)) \in B$ holds (remember that $\mapsto_{t_i}^{l_i}$ could be $\mapsto_0^{\lambda}$ i.e., a stutter).*

Definition 3 (Continuous Observational Bisimulations). *Given a LTS $G_A = \langle Q_A, q_0, \mapsto_t^l, \Sigma \rangle$ and a propositional assignment $\mathcal{P} : Locs(A) \mapsto 2^{Props}$. We say that a symmetric binary relation $B \subseteq Q_A \times Q_A$ is a continuous observational bisimulation (CO-Bisimulation) of G_A w.r.t. $\mathcal{P}$ iff $(p, q) \in B$ implies*

[2] A more restrictive notion for timed systems is found in [26].

that $\mathcal{P}(p^{@}) = \mathcal{P}(q^{@})$ *and for all* $a \in \Sigma$, $t \in I\!\!R_{\geq 0}$, *whenever* $p \mapsto_0^a p'$ *then, for some* $q', q'' \in Q$, $a' \in \Sigma \cup \{\lambda\}$, $q \xrightarrow{B,p}_0 q' \mapsto_0^{a'} q''$, *and* $(p', q'') \in B$, *and whenever* $p \mapsto_t^{\lambda} p'$ *then, for some* $q' \in Q$, $q \xrightarrow{B,p}_t q'$ *(which also means that* $(p', q') \in B$*).*

Two TA A_1 and A_2 are *Continuous Observational Bisimilar* (CO-Bisimilar) w.r.t. the propositional assignments $\mathcal{P}_1$ and $\mathcal{P}_2$ ($A_1 \simeq^{\mathcal{P}_1,\mathcal{P}_2} A_2$) iff there exists a CO-Bisimulation B of $G_{A_1} \cup G_{A_2}$ w.r.t. $\mathcal{P}_1 \cup \mathcal{P}_2$, such that $(q_{01}, q_{02}) \in B$. Two CO-Bisimilar TAs are equivalent up to TCTL logic (see [6,9]).

3 I/O Timed Components

I/O Timed Components (I/O TCs) [8] is a formalism built on top of TAs, developed for expressing non-zeno timed behavior (it is always possible to make time diverge) and to support "assume-guarantee" compositional reasoning without resorting to receptiveness games[17]. I/O TCs are immediately supported by several checking tools like Kronos [12], Uppaal [4], etc.[3]. As we discuss in Sect. 4, I/O interfaces make possible our static calculus of influence. Given a TA A, we divide $Labels(A)$ into three sets: (1) the set of input-labels (In_A), (2) the set of output-labels (Out_A) and (3) the set of internal-labels (ϵ_A), such that $\{In_A, Out_A, \epsilon_A\} \in Part(Labels(A))$, where $Part(Set)$ is the set of all partitions of the set Set. We define the set Exp_A of *exported* labels (or interface labels) of A as $Exp_A = In_A \cup Out_A$.

A set of *input selections* of A is a set $I^A = \{I_1^A, \ldots, I_k^A\} \in Part(In_A)$, a set of *output selections* of A is a set $O^A = \{O_1^A, \ldots, O_h^A\} \in Part(Out_A)$. Note that $I^A \cup O^A \cup \{\epsilon_A\} \in Part(Labels(A))$.

Definition 4 (I/O TCs). *An I/O Timed Component (or I/O TC) is a pair* $(A, (I^A, O^A))$ *where* A *is a non-zeno TA and* (I^A, O^A) *is an admissible I/O interface for* A*: the sets* I^A *and* O^A *are input and output selections (resp.) of* A*, and for any state* $q \in Reach(A)$ *the following conditions hold:*

1. *for any input selection* $I_n^A \in I^A$ *there exists a label* $i \in I_n^A$ *such that* $q \mapsto_0^i$*. That is, given any input selection* $I_n^A \in I^A$*, the TA can always synchronize using some of the labels of* I_n^A *(there is always at least one alternative of every input selection enabled at each state).*
2. *there exists a run* $r \in R_A^{\infty}(q)$ *such that* $L(r) \cap In_A = \emptyset$*. Input is not mandatory and thus non-zenoness must be guaranteed without them.*
3. *for any output selection* $O_m^A \in O^A$*, if there exists a label* $o \in O_m^A$ *such that* $q \mapsto_0^o$ *then* $q \mapsto_0^{o'}$ *for all* $o' \in O_m^A$*. All labels of an output selection are simultaneously enabled or disabled.*
4. *for any infinite run* $r \in R_A(q)$*, if a label* $o \in Out_A$ *appears an infinite number of times in* r*, then necessarily* $r \in R_A^{\infty}(q)$ *(non-transientness of outputs).*

[3] Liveness and I/O interfaces have been considered in a general setting for simulation proof methods "à la" Lynch-Vaandrager [17] geared towards theorem provers. A further discussion on related work can be found in [8].

An output selection of size greater than one models alternative behaviors of the component according to the state of the component exporting those labels as input. This is similar to an external non-deterministic choice in Process Algebra like notations.

Definition 5 (Compatible Components). *Given two I/O TCs* $\mathbf{A_1} = (A_1, (I^{A_1}, O^{A_1}))$ *and* $\mathbf{A_2} = (A_2, (I^{A_2}, O^{A_2}))$*, they are compatible components iff:*

1. $\mathit{Labels}(A_1) \cap \mathit{Labels}(A_2) \subseteq \mathit{Exp}_{A_1} \cap \mathit{Exp}_{A_2}$ *(i.e., all common labels are exported by both* A_1 *and* A_2*),*
2. *for all* $I_n^{A_1} \in I^{A_1}$ *and* $I_m^{A_2} \in I^{A_2}$ *if* $\#I_n^{A_1} > 1$ *and* $\#I_m^{A_2} > 1$ *then* $I_n^{A_1} \cap I_m^{A_2} = \emptyset$ *(intersection of input selections of size greater than one must be empty).*
3. $\mathit{Out}_{A_1} \cap \mathit{Out}_{A_2} = \emptyset$ *(there are not common output-labels).*
4. *for all* $O \in O^{A_1} \cup O^{A_2}$ *and* $I \in I^{A_1} \cup I^{A_2}$ *then either* $I \cap O = \emptyset$ *or* $I \subseteq O$*.*

We refer to a set of pair-wise compatible components as a *compatible set of components*. I/O compatibility is a syntactic condition that implies that underlying TAs can not block each other and, moreover, in [8] we show that the composition of compatible components is itself a component and therefore a non-zeno TA.

Example 1. Fig. 1 introduces the example of a pipe-line of sporadic tasks. The pipe-line is composed of tasks together with latches for buffering signals between them. The observer checks whether an end-to-end response-time requirement is guaranteed.

4 Optimizing the Composition of I/O Components

4.1 Influence

The core of our technique is a notion of potential "direct influence" of a component behavior over another component behavior. A naive solution is to consider that two components influence each other if they share a label. Unfortunately, this definition would lead to a rather large symmetric overestimation[4]. By using the I/O interface attached to TA (the I/O TCs), we can define a better asymmetric condition for behavioral influence. Besides, our notion of influence is relative to the locations considered for each automaton (i.e., assuming that A is at some location which belongs to the set $S_A \subseteq \mathit{Locs}(A)$ and B is at some location which belongs to the set $S_B \subseteq \mathit{Locs}(B)$). Using two compatible I/O Timed Component $\mathbf{A} = (A, (I^A, O^A))$ and $\mathbf{B} = (B, (I^B, O^B))$, $S_A \subseteq \mathit{Locs}(A)$, $S_B \subseteq \mathit{Locs}(B)$, $e_a \in \mathit{Edges}(A)$, $e_b, e'_b \in \mathit{Edges}(B)$, and $x \in \mathit{Clocks}(B)$, we define in Table 1 the following notions of influence:

1. **A** influences **B** while they are sojourning S_A and S_B resp., denoted $\mathbf{A}|_{S_A} \rightarrow \mathbf{B}|_{S_B}$.

[4] An asynchronous variable sharing mechanism of communication would make dependence checking easier using read-write operations.

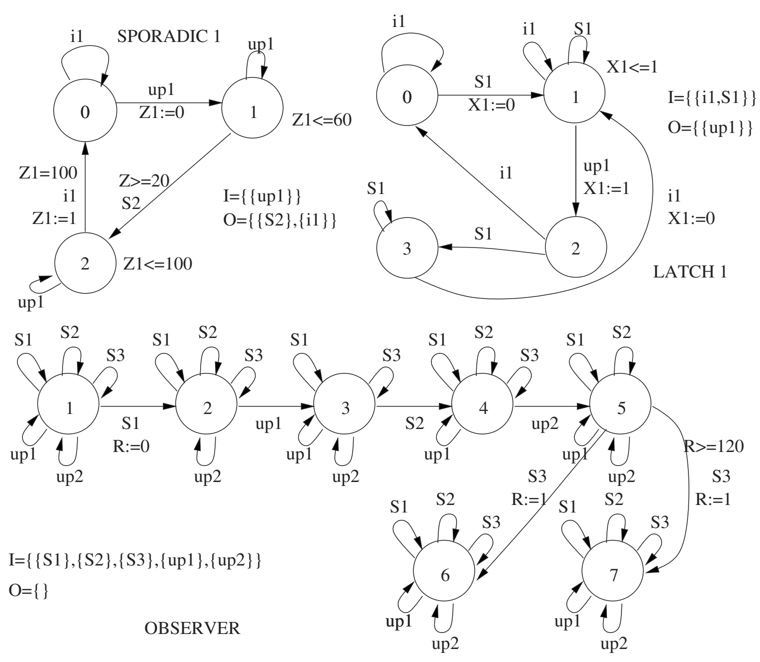

Fig. 1. Some components of Pipe-line System

Table 1. Influence rules

$$\frac{src(e_a)\in S_A, lab(e_a)\in Out_A, src(e_b)\in S_B, lab(e_b)=lab(e_a), tgt(e_b)\neq src(e_b)}{\mathbf{A}|_{S_A}\to\mathbf{B}|_{S_B}}$$

$$\frac{I\in I^A, \#I>1, lab(e_b)\in I, src(e_b)\in S_B}{\mathbf{A}|_{S_A}\to\mathbf{B}|_{S_B}} \qquad \frac{src(e)\in S_B, x\in grd(e)}{x\to\mathbf{B}|_{S_B}} \qquad \frac{l\in S_B, x\in Inv(l)}{x\to\mathbf{B}|_{S_B}}$$

$$\frac{I\in I^A, lab(e_b)\in I, lab(e'_b)\in I, lab(e_b)\neq lab(e'_b), src(e_b)\in S_B, src(e'_b)\in S_B, x\in rst(e_b), x\notin rst(e'_b)}{\mathbf{A}|_{S_A}\to x|_{S_B}}$$

$$\frac{src(e_a)\in S_A, lab(e_a)\in Out_A, src(e_b)\in S_B, lab(e_a)=lab(e_b), x\in rst(e_b)}{\mathbf{A}|_{S_A}\to x|_{S_B}} \qquad \frac{src(e_b)\in S_B, x\in rst(e_b)}{\mathbf{B}|_{S_B}\to x}$$

2. **A** influences clock x of **B** while they are sojourning S_A and S_B resp., denoted $\mathbf{A}|_{S_A} \to x|_{S_B}$.
3. **B** influences its clock x while it is sojourning S_B, denoted $\mathbf{B}|_{S_B} \to x$.
4. x influences its component **B** while it is sojourning S_B, denoted $x \to \mathbf{B}|_{S_B}$.

Note that if $\mathbf{A}|_{S_A} \to \mathbf{B}|_{S_B}$, $S_A \subseteq S'_A$, and $S_B \subseteq S'_B$ then $\mathbf{A}|_{S'_A} \to \mathbf{B}|_{S'_B}$. The same is also true for the rest of notions of influence.

In what follows, we consider the system of I/O timed components $\{A_i / i \in \{0..n\}\}$ where A_0 plays the role of the observer while the rest of components constitutes the *SUA*. We define a function S_l that associates to each observer location l and component A_i a *Sojourn Set*: a set of locations that A_i may visit while the observer is at l.

Definition 6 (Sojourn Set). *Given a set of TAs* $\{A_i / i \in \mathcal{I}\}$ *where* $\mathcal{I} = \{0..n\}$, $l \in \mathit{Locs}(A_0)$*, we define*
$S_l(0) = \{l\}$*, and for* $1 \leq i \leq n$,
$S_l(i) = \{\Pi_i(q)^@ \in \mathit{Locs}(A_i) / q \in \mathit{Reach}(\|_{i \in \mathcal{I}}\, A_i)$ *and* $\Pi_0(q)^@ = l\}$.

The exact calculation of the sojourn set is unpractical because it would require the computation we want to optimize: $\mathit{Reach}(\|_{i \in \mathcal{I}}\, A_i)$. However, these sets can be overestimated by several procedures [7]. Experience has shown us that one of the most cost-effective strategies is $S_l'(i) = \{l' \in \mathit{Locs}(A_i) / \ (l, l') \in \mathit{Locs}(A_0 \parallel A_i)\}$. Since in many cases, even timed systems with relatively small control structures are intractable (a few thousands of nodes), it is reasonable to use the untimed composition to get those sets, i.e., $S_l'(i) = \{l' \in \mathit{Locs}(A_i) / \exists s \in \mathit{Locs}(A_0 \parallel A_1 \parallel ... \parallel A_n) \wedge \Pi_0(s) = l \wedge \Pi_i(s) = l'\}$. Hereafter, we denote

- $\mathbf{A_i}|_{S_l(i)} \rightarrow \mathbf{A_j}|_{S_l(j)}$ as $\mathbf{A_i} \overset{S_l}{\rightarrow} \mathbf{A_j}$,
- $\mathbf{A_i}|_{S_l(i)} \rightarrow x|_{S_l(j)}$ or $\mathbf{A_i}|_{S_l(i)} \rightarrow x$ as $\mathbf{A_i} \overset{S_l}{\rightarrow} x$, and
- $x \rightarrow \mathbf{A_j}|_{S_l(j)}$ as $x \overset{S_l}{\rightarrow} \mathbf{A_j}$.

4.2 Relevance

Firstly, let us present a couple of definitions that are necessary to calculate activity of components and clocks to latter optimize the *SUA* w.r.t. the observer.

Definition 7 (Determination of Components). *Given an I/O Timed Component* $\mathbf{A} = (A, (I^A, O^A))$, $S \subseteq \mathit{Locs}(A)$*, and* $i \in \Sigma_A$*, we say that* i *determine* $\mathbf{A}$ *while it is sojourning* S *denoted* $\mathit{Det}_A(i, S)$ *iff* $\{i\} \in I^A$ *and* $\forall e, e' \in \mathit{Edges}(A) :$ $\mathit{src}(e), \mathit{src}(e') \in S, \mathit{lab}(e) = \mathit{lab}(e') = i$ *implies* $\mathit{tgt}(e) = \mathit{tgt}(e')$ *and we denote* $\mathit{tgt}(e)$ *as* $\mathit{tgt}(S, i)$.

Definition 8 (Determination of Clocks). *Given a Timed Automata* A, $x \in \mathit{Clocks}(A)$, $S \subseteq \mathit{Locs}(A)$*, and* $a \in \mathit{Labels}(A)$, $\mathit{Reset}_A(a, S)$ *is the maximum set of clocks* $X' \subseteq \mathit{Clocks}(A)$ *such that for all* $e \in \mathit{Edges}(A)$, $\mathit{src}(e) \in S$ *and* $\mathit{lab}(e) = a$ *then* $X' \subseteq \mathit{rst}(e)$.

We are ready to show how to calculate a pair of functions, that we call *Active* and *RelClocks*, which defines the components and clocks whose activity may influence the behavior of the observer. Following the static analysis terminology *Active* and *RelClocks* are calculated as the solution of a data flow analysis problem over the control structure of the observer: a combination of a transitive closure of the direct influence relation plus a backwards transference function. For the sake of readability, we present them using a formal system notation in Table 2. We say that *Active* and *RelClocks* conform a correct activity pair of functions for A_0 if they are the minimal functions satisfying the Activity

Table 2. Activity rules

$$\frac{-}{0\in Active(l)}\ \textbf{[ObsRel]} \qquad \frac{j\in Active(l), x \xrightarrow{S_l} \mathbf{A_j}}{x\in RelClocks(l)}\ \textbf{[Transitivity I]}$$

$$\frac{x\in RelClocks(l), \mathbf{A_i} \xrightarrow{S_l} x}{i\in Active(l)}\ \textbf{[Transit. II]} \qquad \frac{j\neq i, j\in Active(l), \mathbf{A_i} \xrightarrow{S_l} \mathbf{A_j}}{i\in Active(l)}\ \textbf{[Transit. III]}$$

$$\frac{src(e)=l, i\in Active(tgt(e)), \neg Det_{A_i}(lab(e), S_l(i))}{i\in Active(l)}\ \textbf{[Tranference I]}$$

$$\frac{src(e)=l, x\in Clocks(A_j), x\in RelClocks(tgt(e)) - Reset_{A_j}(lab(e), S_l(j))}{x\in RelClocks(l)}\ \textbf{[Tranference II]}$$

where $\mathcal{I} = \{0..n\}$, $\{A_i\}_{i\in\mathcal{I}}$ is a set of pair-wise compatible I/O Timed Components, $Active : Locs(A_0) \mapsto 2^{\mathcal{I}}$, $RelClocks : Locs(A_0) \mapsto 2^X$, X is the union of the set of clocks of all components, $l \in Locs(A_0)$, and $e \in Edges(A_0)$.

rules. When a clock is not marked as relevant at l that means that the values it acquires while the *SUA* is sojourning l has no effect on the future observed behavior till it becomes relevant again. The same is true for components that are not marked as active. The optimized system will only keep track of active components and relevant clocks. Table 3 shows a correct activity function *Active* for the example 1.

Table 3. Relevance Function for Sporadic Pipe-line

Obs **loc.**	*Active*(**loc**)
0	$Obs, source, latch_1, proc_1, latch_2, proc_2$
1	$Obs, latch_1, proc_1, latch_2, proc_2$
2	$Obs, proc_1, latch_2, proc_2$
3	$Obs, latch_2, proc_2$
4	$Obs, proc_2$
5	Obs
6	Obs

4.3 Transformation

In this section we define a procedure to build the optimization according to *RelClocks* and *Active* over A_0 (the observer) and the *SUA*. The transformation modifies each component of the *SUA* adding new transitions and an "idle" location named * to model deactivation of the component. The optimization process implies a total relabeling of transitions. Old labels are embedded and new ones are added to communicate the change of locations done by the observer (actually the projections of a new label tell which components should be enabled or disabled when the edge is traversed). We generate a new label for each edge of the observer. The idea is that, when the transformed observer jumps from l to l', disabled components (i.e., active components at l that are inactive at l') synchronize and jump to their idle locations. The effect is similar to the elimination of variables standing for the control location of disabled components. On the other hand, enabled components (active components at l' but non active at l) are forced to jump to their right location: the target location of Def. 7. Clocks are treated in a similar way. What follows formalizes the previous notions.

Given a set of I/O TCs $\mathbf{A_i} = (A_i, (I^{A_i}, O^{A_i}))$, where $A_i = (S_i, X_i, \Sigma_i, E_i, Inv_i, s0_i)$ with $i \in \mathcal{I} = \{0..n\}$, *Active* and *RelClocks* as defined in Table 2, first let us define the new set of labels $NL = \Sigma \times 2^{\mathcal{I}} \times 2^{X} \times (\mathcal{I} \mapsto S)$, with *Lab*, *Dis*, *Rst*, *Ena* as its projections respectively. The operators $Emb()$ and $Nl()$ build new labels from the original ones and the observer edges.

Given a label $a \in \Sigma$, and an edge $e \in E_0$, the new labels $b \overset{def}{=} Emb(a) \in NL$ and $b \overset{def}{=} Nl(e) \in NL$ are such that:

	$Emb(a)$	$Nl(e)$
$Lab(b)$	a	$lab(e)$
$Dis(b)$	$\emptyset$	$Active(src(e)) - Active(tgt(e))$
$Rst(b)$	$\emptyset$	$RelClocks(tgt(e)) - RelClocks(src(e))$
$Ena(b)$	$\emptyset$	$\{(i, tgt(lab(e), S_{src(e)}(i)))/i \in Active(tgt(e)) - Active(src(e))\}$

We define $NLabs = \{Emb(a) \in NL/a \in \Sigma\} \cup \{Nl(e)/e \in E_0\}$ and $NLabs_i = \{b \in NLabs/Lab(b) \in \Sigma_i \vee i \in Dis(b)\}$.

The optimization is defined as

$Opt_{Active}(A_0) = \langle S_0, X_0, NLabs_0, E'_0, Inv_0, s0_0\rangle$,
where $E'_0 = \{\langle src(e), Nl(e), grd(e), rst(e), tgt(e)\rangle/e \in E_0\}$

For $1 \leq i \leq n$:

$Opt_{Active}(A_i) = \langle S_i \cup \{*\}, X_i, NLabs_i, E'_i, Inv_i \cup \{(*, true)\}$, $s0_i$ if $i \in Active(s0_0)$ else $*\rangle$, where E'_i is defined by rules of Table 4.

The Fig. 2 shows the working example optimized respect to *Active*. Although the syntactic size of each component is enlarged, deactivation locations reduces the state space because unnecessary interleaving is avoided [7]. Some experimental results that back up this conjecture are shown in next section. Moreover, *RelClocks* could be used to deactivate and activate clocks in the very same spirit that [13]. More precisely, a clock x such that $x \notin RelClocks(l)$ could be deactivated whenever the observer performs a jump to l. Tools like OpenKronos [23] allows to explicitly specify deactivation of clocks and this information could also dramatically improve performance[5]. It is not difficult to see that this kind of

[5] In the figures the assignment $x := 1$ means that the clock x is deactivated.

Table 4. Edges rules

$\dfrac{b{\in}NLabs_i,\langle s,Lab(b),g,r,s'\rangle{\in}E_i,i{\notin}Dis(b)}{\langle s,b,g,r,s'\rangle{\in}E'_i}$	$\dfrac{b{\in}NLabs_i,\langle s,Lab(b),g,r,s'\rangle{\in}E_i,i{\in}Dis(b)}{\langle s,b,g,r,*\rangle{\in}E'_i}$
$\dfrac{b{\in}NLabs_i,Lab(b){\notin}\Sigma_i,i{\in}Dis(b)}{\langle s,b,true,\{\},*\rangle{\in}E'_i}$	$\dfrac{b{\in}NLabs_i,i{\in}Domain(Ena(b))}{\langle *,b,true,Rst(b){\cap}Clocks(A_i),Ena(b)(i)\rangle{\in}E'_i}$
$\dfrac{b{\in}NLabs_i,Lab(b){\in}In_{A_i},i{\notin}Domain(Ena(b))}{\langle *,b,true,Rst(b){\cap}Clocks(A_i),*\rangle{\in}E'_i}$	$\dfrac{b{\in}NLabs_i,Lab(b){\notin}\Sigma_i,i{\in}Dis(b)}{\langle *,b,true,Rst(b){\cap}Clocks(A_i),*\rangle{\in}E'_i}$

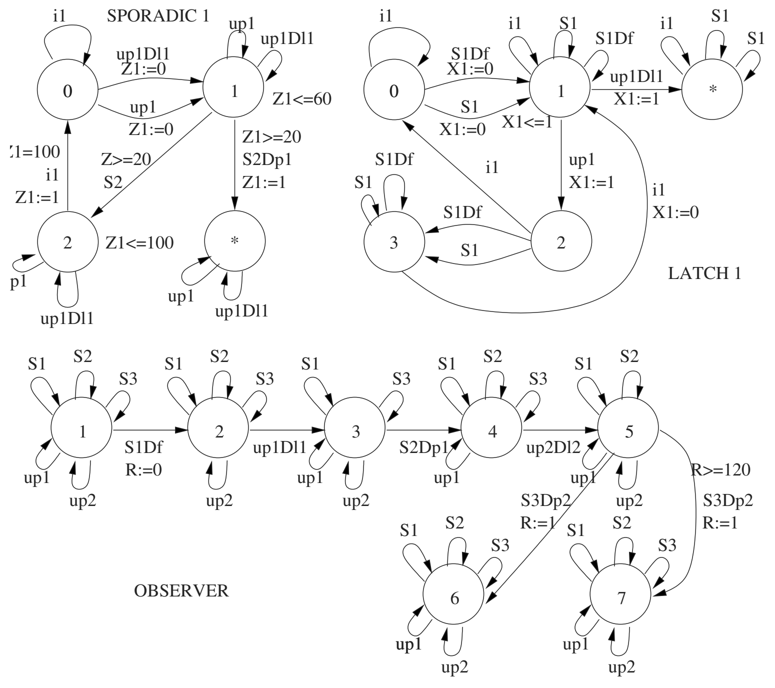

Fig. 2. Transformed components of Pipe-line System

abstraction could be directly built on-the-fly adapting verification engines like OpenKronos [23] or Treat [19] to verify reachability.

The resulting model is CO-Bisimilar to the normal composition and therefore validates the same TCTL formula over the observer for instance reachability. Moreover, in the proof of Theorem 1 which is given in [7], it is noticeable that closed runs[6] of the optimized models are the projections of the runs of the

[6] A run is closed if all events are internal or output event of at least one component.

original model, exhibiting only relevant activity, i.e., lost labels are the events performed by the non relevant components.

Theorem 1. *Given an indexed set of compatible I/O Timed Components* $\{A_i\}_{0 \leq i \leq n}$ *, and an assignment mapping* $\mathcal{P} : Locs(A_0) \mapsto 2^{Props}$*, if Active is a correct activity function then* $(Opt_{Active}(A_0) \parallel Opt_{Active}(A_1) \parallel ... \parallel Opt_{Active}(A_n)) \simeq^{\mathcal{P}'^*, \mathcal{P}^*} (A_0 \parallel A_1 \parallel ... \parallel A_n)$ *where* $\mathcal{P}^*$ *and* $\mathcal{P}'^*$ *are the natural extensions of* $\mathcal{P}$ *on the locations of* $A_0 \parallel A_1 \parallel ... \parallel A_n$ *and* $(Opt_{Active}(A_0) \parallel Opt_{Active}(A_1) \parallel ... \parallel Opt_{Active}(A_n))$ *resp.*

As mentioned, from the TCTL preservation theorem in [6,9] we obtain:

Corollary 1. *Given an indexed set of compatible I/O Timed Components* $\{A_i\}_{0 \leq i \leq n}$ *, and an assignment mapping* $\mathcal{P} : Locs(A_0) \mapsto 2^{Props}$*, If Active is a correct activity function. Then, for all TCTL formula* ϕ $(Opt_{Active}(A_0) \parallel Opt_{Active}(A_1) \parallel ... \parallel Opt_{Active}(A_n)) \models_{\mathcal{P}'^*} \phi \iff A_0 \parallel A_1 \parallel ... \parallel A_n \models_{\mathcal{P}^*} \phi$ *where* $\mathcal{P}^*$ *and* $\mathcal{P}'^*$ *are the natural extensions of* $\mathcal{P}$ *on the locations of* $A_0 \parallel A_1 \parallel ... \parallel A_n$ *and* $(Opt_{Active}(A_0) \parallel Opt_{Active}(A_1) \parallel ... \parallel Opt_{Active}(A_n))$ *resp.*

4.4 Some Experiments

OpenKronos [23] was our testbed to show how these ideas can improve the performance of a verification back-end tool. We have ran several experiments with OpenKronos on an AMD K7 1333Mhz 256Mbytes LINUX 7.2 platform. Tables 5 and 6 show verification times, the size of the symbolic state space built by OpenKronos in the case that the error is not reachable or the length of the shortest counterexample we could find[7]. O/M stands for "out of memory": the internal memory has been exhausted and the verifier process trashed. O/T stands for "out of time", meaning that the verification process over the original model has been stopped after waiting more than 100 times the verifier performance over the optimized model. The first example is the pipe-line of processes. We play with three parameters: the number of stages, the period of the incoming signal, and the deadline expressed in the observer (we take $n \times 60$). Remember that there are two automata for each stage (the process and the latch) therefore, n stages means $n \times 2 + 2$ automata and the same number of clocks. The state space is halved when the error is not reachable and this allows us to treat cases that trash without this proposed optimization. It is remarkable how dramatically the time and length of counterexamples are reduced using the optimized system. Again, some intractable cases has been treated in less than one second. It seems that the exploration tool works in much more guided fashion over the optimized system specially while sojourning observer locations where some components are discarded.

The second example is an extension of the FDDI token ring protocol [23] plus an observer that monitors the time the token takes to return to a given station. The model is composed by a ring and n stations which, in turn, are composed

[7] *time* is the time it takes to inform the existence of a counterexample, while *time** is the time it takes to build the shortest counterexample when we provide a maximum depth for the search.

by the station itself (containing 2 clocks) plus an automata that keeps track of the parity of the times the station received the token (this is needed to know how to manipulate clocks each time). In this case, stations perform some internal activity after releasing the token and before obtaining it again. This increases the state space with intuitively irrelevant activity for the observed behavior. Thanks to transference rules, our method discovers this phenomena. Just one station should be active at each locations of the observer. However, it safely keeps all clocks and parity automata active since they are needed whenever the station is interrupted by a new arrival of the token. Table 6 shows the important impact of this reduction. Other experiments using backward verification of full TCTL can be found in [15].

Table 5. Verification results for the Pipe-line example

Hit Error = false		Original			Optimized		
# stages	period	# st	# tr	time	# st	# tr	time
4	180	4,009	7,485	0.25	2,383	4,391	0.13
4	120	38,376	90,478	3.82	22,817	53,817	2.00
5	180	61,081	136,388	6.90	32,523	71,938	3.23
5	120	O/M	-	-	680,421	2,009,126	234.67
6	180	O/M	-	-	573,541	1,507,124	184.80
6	120	O/M	-	-	O/M	-	-
Hit Error = true							
# stages	period	depth	time	time *	depth	time	time *
4	99	35	O/M	2.90	22	0.15	0.02
4	40	33	3.38	1.56	22	0.01	0.11
5	99	O/M	O/M	O/M	27	0.01	0.01
5	40	50	241.19	241.19	27	0.01	0.01
6	99	O/M	O/M	O/M	299	0.45	65.70
6	40	O/M	O/M	O/M	75	0.02	41.40

5 Conclusions and Future Work

It is well known that in order to mitigate state explosion problem combinations of different techniques should be applied. We provide a correct and complete approach to optimize the analysis of timed systems. Our technique can be fed naively with the components of the timed system under analysis plus the observer. It statically discovers the underlying dependence among components and the observer and provides the set of elements (components and clocks) that can be safely ignored, from the observational point of view. This implies significant time savings during the verification step. The tool is also rather easy to use and integrate as a preprocessor for known tools: it just requires I/O (uncontrollable/controllable) declarations which, in general, are intuitively known

Table 6. Verification results for the FDDI example

Hit Error = false		Original			Optimized		
# stations	deadline	# st	# tr	time	# st	# tr	time
6	430	169,952	1,137,408	66.89	3,828	4,540	0.50
8	480	O/M	O/M	-	18,419	21,833	4.35
10	530	O/M	O/M	-	81,574	96,559	32.68
Hit Error = true							
# stations	deadline	depth	time	time *	depth	time	time *
6	420	447	66.13	0.63	14	0.48	0.01
8	470	2,045	O/T	9.23	18	4.36	0.03
10	520	12,305	O/T	224.61	22	32.62	0.07

by modelers or can be automatically provided by high-level front-end modeling languages. It is worth remarking that in some cases we are able to treat cases where the analysis of the original standard composition is unfeasible under the same conditions. In all case studies, the verification tool run faster and obtains shorter counterexamples over the optimized than over the original model. Two main factors affecting the efficiency of the technique: the influence "coupling" of components and the topology of the observer. For instance there are cases where the more detailed the observer is (in the sense of the sequence of events) the more reduction is obtained; particularly when the detailed observer shows an expected sequence of events and thus irrelevance of components is more likely to be detected at some locations of that observer. We believe that the ideas of our approach can be migrated to several frameworks and are complementary to many state space reduction techniques. Future work also includes less conservative definitions of influence, the extension of these techniques to automata with variables and the integration with other popular tools like Uppaal [4]. Our technique could also obtain the basic information to calculate the relevant events for each component. Then, components could be abstracted to exhibit the same set of runs up to those relevant events. Linear-time properties like reachability may be checked using these simulation equivalent reductions. An ad hoc automatic procedure for the aforementioned reduction is presented in [6].

References

1. B. Alpern and F. Schneider, "Verifying Temporal Properties without Temporal Logic," *ACM Trans. Programming Lang. and Systems*, Vol. 11, No. 1, 1989, pp. 147-167.
2. R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking for Real-Time Systems," In *Information and Computation,*, Vol. 104, No. 1, pp.2-34, 1993.
3. R. Alur and D. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, Vol. 126, 1994, pp. 183-235.

4. J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL- A Tool Suite for the Automatic Verification of Real-Time Systems," *Proc. Hybrid Systems III*, LNCS 1066, 1996, pp. 232-243.
5. M. Bozga, J.-C. Fernandez, and L.Ghirvu, "Using Static Analysis to Improve Automatic Test Generation," *TACAS 2000*, LNCS 1785, March/April 2000.
6. V. Braberman, *Modeling and Checking Real-Time System Designs,* Ph.D Thesis, Universidad de Buenos Aires, 2000.
7. V. Braberman, D. Garbervetsky, and A. Olivero, "Influence Information to Improve the Verification of Timed Systems," *Tech. Report DC-UBA 2001-003.*
8. V. Braberman, and A. Olivero, "Extending Timed Automata for Compositional Modeling Healthy Timed Systems," *2nd Workshop on Models for Timed Critical Systems 2001.* To be published in Electronic Notes in Theoretical Computer Science.
9. V. Braberman, and A. Olivero, "Preserving Branching-Time Structure in Timed Systems," *Argentinian Workshop on Theoretical Computer Science 2001.* 30^{th} JAIIO Proceedings.
10. S. Campos, O. Grumberg, K. Yorav, and C. Fady. "Test Sequence Generation and Model Checking Using Dynamic Transition Relations," *Submitted to : FMCAD 2000.*
11. E. Clarke, O. Grumberg and D. Peled, *Model Checking,* MIT Press, January 2000.
12. C. Daws, A. Olivero, S. Tripakis and S. Yovine, "The Tool KRONOS," *In Proc. of Hybrid Systems III*, LNCS 1066, 1996, pp. 208-219.
13. C. Daws and S.Yovine, "Reducing the Number of Clock Variables of Timed Automata," *Proc. IEEE RTSS '96*, IEEE Computer Soc. Press, 1996.
14. R. De Nicolla, U. Montanari, and F.Vaandrager, "Back and Forth Bisimulations," *Proc. CONCUR '90*, Amsterdam, LNCS 458, 1990, pp.152–165.
15. D. Garbervetsky, *Un Método de Reducción para la Composición de Sistemas Temporizados,* Master Thesis, Univ. de Buenos Aires, 2000.
16. P. Godefroid, "Partial-Order Methods for the Verification of Concurrent Systems", LNCS 1032, 1996.
17. R. Gawlick, R. Segala, J. Sogaard-Andersen, N. Lynch "Liveness in Timed and Untimed Systems," *Proceedings of ICALP* , LNCS 820, Springer Verlag, pp. 166-177, 1994. Also in, *Information and Computation*, March 1998.
18. J. Hatcliff, J. Cobett, M. Dwyer, S.Sokolowski, and H.Zheng "A Formal Study of Slicing for Multi-Threaded Programs with JVM Concurrency Primitives," *SAS.* LNCS 1694, 1999.
19. I. Kang, I. Lee, and Y.S. Kim, "An Efficient Space Generation for the Analysis of Real-Time Systems". *Trans. on Software Engineering*, Vol. 26, No. 5, pp. 453-477, May 2000.
20. K.G. Larsen, F. Laroussinie, "CMC: A Tool for Compositional Model-Checking of Real-Time Systems," *Proc. FORTE-PSTV'98*, 439-456, Kluwer Academic Publishers, 1998.
21. R Sloan, U. Buy, "Stubborn Sets for Real-Time Petri Nets," *Form. Methods in Syst. Design*, Vol. 11(1), pp.23-40, July 1997.
22. S. Tasiran, S. P. Khatri, S. Yovine, R. K. Brayton, and A. Sangiovanni-Vincentelli. "A Timed Automaton-Based Method for Accurate Computation of Circuit Delay in the Presence of Cross-Talk," *FMCAD'98*, 1998.
23. S. Tripakis *L'Analyse Formelle des Systemès Temporisés en Practique,* Phd. Thesis, Univesité Joseph Fourier, December 1998.

24. F. Wang, "Efficient and User-Friendly Verification," *Transaction in Computers*, Accepted for Publication, June 2001.
25. Wang Yi, "Real-Time Behavior of Asynchronous Agents," *Proc. CONCUR'90*, LNCS 458, 1990.
26. S. Yovine, *"Méthodes et Outiles pour la Vérification Symbolique de Systemès Temporisés,"* doctoral disertation, Insitut National Polytechnique de Grenoble, 1993.
27. S. Yovine, "Model-Checking Timed Automata," *Embedded Systems,* G. Rozemberg and F. Vaandrager eds., LNCS 1494, 1998.