Abstraction strategies for computing travelling or looping durations in networks of timed automata

Raymond Devillers, Hanna Klaudel

Abstraction Strategies for Computing Travelling or Looping Durations in Networks of Timed Automata

Raymond Devillers[1] and Hanna Klaudel[2(✉)]

[1] Département D'Informatique,
Université Libre de Bruxelles, City of Brussels, Belgium
rdevil@ulb.ac.be
[2] Laboratoire IBISC, Université D'Evry-Val D'Essonne, Evry, France
hanna.klaudel@ibisc.univ-evry.fr

Abstract. This paper shows how to abstract networks of timed automata in order to accelerate the analysis of quantitative properties such as path or cycle duration, that would otherwise suffer from the state space explosion. Two approaches are introduced, a single step strategy and an iterative one, where a part of the network of timed automata is merged and abstracted. As a consequence, the state space is reduced and model-checking is simplified. These approaches are illustrated on a case study, where the comparison is done by calculating the cycle time of one automaton in the network, both on the real network and on the two abstracted ones, showing that the method reduces significantly the runtime, or simply renders feasible the analysis of the system.

Keywords: Timed automata · State space explosion · Duration · Approximation

1 Introduction

Timed automata [1] are a powerful formalism useful to model and analyse real-time concurrent systems. They extend finite state machines by adding real-valued variables, called clocks, which evolve linearly and can be compared with integer constants in states (yielding invariants), called here locations, and along transitions (yielding guards), where some clocks may also be reset to zero; additional (Boolean or integer) variables may also be introduced, checked in guards and invariants, and updated along transitions. Networks are sets of timed automata, which may synchronise through binary communication channels, meaning that a communication involves exactly two components, one performing an emission $k!$ and the other one performing a reception $k?$ on the same channel k. Model checking may be performed on such models, in particular with tools like UPPAAL [9,12].

It is well known that the automated analysis of complex systems, with many communicating components and different orders of magnitude in the used constants, quickly faces a state explosion problem, making the analysis extremely

time and/or memory demanding, or even unfeasible. In such a context, this paper focuses on the computation of upper/lower bounds for the travelling time of various paths or loops in timed automata components. The objective here is to determine if the timing characteristics of the considered system are satisfactory or not, rather than to compute exact bounds (even if it may often lead to tight bounds). We use approximations, which are based on the analysis of the involved communication structure of the network and replace some parts of it by suitable abstractions. We present two approaches (a one step and an iterative one) accelerating and often simply making it possible to achieve these computations. We shall use as a running example a network of timed automata that occurred in the field of mixed reality applications [2].

Related Work. The problems arising from the fragmentation of the state space due to different orders of magnitude in the constants have already been addressed, for various variants of timed automata, and the solutions have sometimes been incorporated in tools [8,10]. The problems due to the complexity of some components have been handled in [6], for instance, where locations are merged in such a way that the added behaviours do not impact the property at hand, or in [7], where UPPAAL has been used in an industrial case study as a "structured testing" tool in order to find bounds for some activities. By contrast, the fact that the size of the state space of distributed systems increases in an exponential way with the number of components is seldom considered, except maybe in [11] where, in a very specific framework, two kinds of components may be distinguished and each kind may be analysed individually, allowing to inject the results in the analysis of the other kind, iteratively until convergence; and in [3] where the authors exploit the specificities of a 2-level real-time scheduling on a single platform. The idea we shall develop here is rather different: in order to simplify the system and make it more adequate for quantitative analyses, we shall consider subsystems and abstract the rest by simple computation nodes.

Paper's Structure: The next sections present first the necessary definitions, then introduces our running example and recalls how one may use UPPAAL to analyse a network of timed automata. Section 5 presents an abstraction strategy, available when only a few components are out of reach of a direct analysis. Section 6 shows how to proceed when no (or too few) components may be analysed. The last section concludes and discusses some possible extensions.

2 Preliminaries

Syntactically, a timed automaton is an annotated directed (and connected) graph, provided with a finite set of non-negative real variables called *clocks*; additional (Boolean or integer) variables may also be introduced. The nodes (*locations*) are annotated with *invariants*, (predicates allowing to enter or stay in a location); they may also have some qualifiers, like an *urgency* indication. The arcs are annotated with *guards* (predicates allowing to perform a move) or *communication actions*, and possibly with some clock *resets* and variable updates.

We shall not detail here the exact syntax allowed for the predicates, adopting the one used in UPPAAL. As usual, the empty predicate is interpreted as true.

In order to glue together the various components of a network of timed automata, some arcs will be classically annotated with communication actions (variable updating may also serve to materialise interactions between components, but also inside a component) which may be either of the form $k!$, meaning the emission of a signal on a channel k, or a complementary $k?$, meaning the reception of some signal on channel k, supposed to synchronise with a $k!$. A channel may also have some qualifiers, like an *urgency* indication. The absence of synchronisation action on an arc indicates an internal activity of the automaton.

Definition 1. *A* timed automaton *is a tuple* $A = (S, s^0, X, K, V, E, I)$*, where*

- S *is a set of locations and* $s^0 \in S$ *is the initial one,*
- X *is the set of clocks,*
- K *is the set of communication actions,*
- V *is the set of variables,*
- $E \subseteq S \times (K \times B \times U \times 2^X) \times S$ *is a set of arcs between locations, possibly annotated with a communication action in* K*, a guard in* B*, a variable update* U*, and a set of clock resets in* 2^X*,*
- $I : S \to B$ *assigns invariants to locations.*

Definition 2. *A* network *of timed automata is a set* $\mathcal{A} = \{A_1, \ldots, A_n\}$ *where each* $A_i = (S_i, s_i^0, X_i, K_i, V_i, E_i, I_i)$ *is an individual timed automaton, the sets* S_i *being disjoint.*

The semantics of a network of timed automata is that of the underlying timed automaton (synchronising together through channels and possibly also interacting through common variables and clocks) as recalled below, with the following notations. A location vector is a vector $\bar{s} = (s_1, \ldots, s_n)$; the initial location vector is $\bar{s}^0 = (s_1^0, \ldots, s_n^0)$. We denote by $s_i \xrightarrow[r,u]{k,b} s_i'$ the arcs between locations, where k is a communication action (which may be absent), b a guard (empty guard is interpreted as true), r is a set of clocks to be reset (possibly empty), and u is an update of variables (also possibly empty). The invariant predicates are composed of predicates over location vectors $I(\bar{s}) = \bigwedge_i I_i(s_i)$. We write $\bar{s}[s_i'/s_i]$ to denote the vector where the ith element s_i of $\bar{s}$ is replaced by s_i', and $\bar{s}[s_i'/s_i, s_j'/s_j]$ to denote the vector where the ith element s_i of $\bar{s}$ is replaced by s_i' while the jth element s_j of $\bar{s}$ is replaced by s_j'. A valuation is a function ν from the set of clocks to the non-negative reals, and from the set of variables to Boolean or integer values. Let $\mathbb{V}$ be the set of all clock and variable valuations, $\nu_0(y) = 0$ for each clock or integer variable y, and $\nu_0(b) = \mathsf{false}$ for each Boolean variable b. We shall denote by $\nu \vDash F$ the fact that the valuation ν satisfies (makes true) the formula F. If r is a clock reset and u a variable update, we shall denote by $\nu[r, u]$ the valuation obtained after applying clock reset r and the variables update u to ν; and if $d \in \mathbb{R}_{>0}$ is a delay, $\nu + d$ is the valuation such that, for any clock x, $(\nu + d)(x) = \nu(x) + d$, the variables being left unchanged.

Definition 3. *The semantics of a network* $\mathcal{A} = \{A_1, \dots A_n\}$ *is defined as a timed transition system* $(St, st_0, \rightarrow)$*, where* $St = (S_1 \times, \dots \times S_n) \times \mathbb{V}$ *is the set of states,* $st_0 = (\bar{s}^0, \nu_0)$ *is the initial state, and* $\rightarrow \subseteq St \times St$ *is the transition relation defined by:*

- *(silent):* $(\bar{s}, \nu) \rightarrow (\bar{s}', \nu')$ *if there exists* $s_i \xrightarrow[r,u]{b} s_i'$*, for some* i*, such that* $\bar{s}' = \bar{s}[s_i'/s_i]$*,* $\nu \vDash b$*,* $\nu' = \nu[r, u]$ *and* $\nu' \vDash I(\bar{s}')$*,*
- *(sync):* $(\bar{s}, \nu) \rightarrow (\bar{s}', \nu')$ *if there exist two arcs* $s_i \xrightarrow[r_i,u_i]{k?,b_i} s_i'$ *and* $s_j \xrightarrow[r_j,u_j]{k!,b_j} s_j'$ *with* $i \neq j$*, such that* $\nu \vDash b_i \wedge b_j$*,* $\bar{s}' = \bar{s}[s_i'/s_i, s_j'/s_j]$*,* $\nu' = \nu[r_i \cup r_j, u_i; u_j]$ *(assuming the updates* u_i *and* u_j *commute) and* $\nu' \vDash I(\bar{s}')$*,*
- *(timed):* $(\bar{s}, \nu) \rightarrow (\bar{s}, \nu + d)$ *if* $\forall x \in [0, 1] : \nu + x \cdot d \vDash I(\bar{s})$*, there is no urgent synchronisation possible (they have precedence on time passing), and there is no urgent location in* $\bar{s}$ *(time may not progress in an urgent location).*

3 Running Example

In order to illustrate our techniques, we shall use along the paper the running example, originated from [2] and depicted in Fig. 1 modelling an augmented reality system with a sensor I (for *inertial*), two cooperating processing units P (for *priority*) and L (for *lower*), a memory component M and two rendering loops G (for *graphical*) and H (for *haptical*).

Component I, once initialised in s_0, cyclically acquires data in s_1 and sends it on channel k_I to processing unit P. After initialisation (between s_0 and s_3), component P cyclically awaits synchronisations with components I (if available) and L (mandatory), processes data in location s_5 and sends its results to component L on channel k_P. Component L acquires data from component P, then processes them in location s_1 (which takes at least 20 and at most 60 time units), synchronises with memory M (on channel *lock*), writes data in it, which lasts between 10 and 20 time units (in location s_3), and unlocks M. Two rendering components G and H access cyclically M, read and update data in s_1 and process them in s_3.

The underlying communication/synchronisation scheme is represented in Fig. 2.

This kind of system may be handled by UPPAAL. Note that in this example all the communication channels are urgent, i.e., when one or more communications may take place, one of them must occur immediately, and we resume until no more communication is allowed (but other moves may occur before the time progresses): then the time may continue to progress. It is easy to see that no Zeno phenomenon may occur here[1]. Each component is essentially looping, possibly after an initialisation part. We shall here consider the timing characteristics of components L and G, and in particular the bounds of durations for performing their loops.

[1] i.e., a situation where infinitely many moves may/must occur in a finite/zero delay.

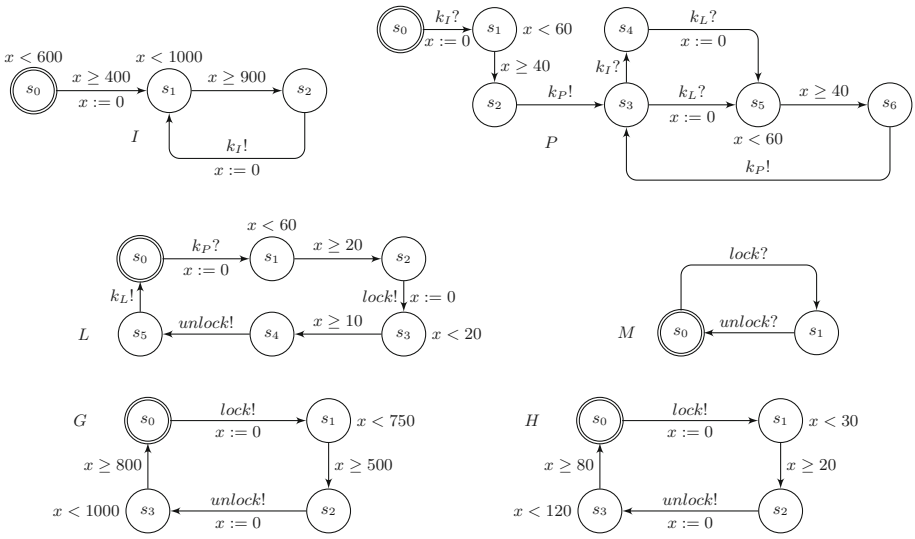

Fig. 1. Network of timed automata of Example 1 (all clocks are local and all channels are urgent).

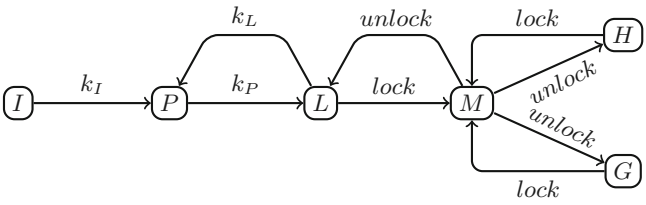

Fig. 2. Communication/synchronisation schema of the running example. The nodes represent components (individual timed automata) and the arcs the synchronisations with the direction from emitting to receiving component.

4 Timing Analysis

Before starting an estimation of the time needed to reach some location s' from location s in a component C, we may first wonder if it may not happen that one gets stuck during the travel (this phenomenon may have various causes - local or global deadlock, starvation, infinite waiting – analysed in [2,5]). This may be checked with a *leads to* property φ - -> φ', verifying if when φ is true it is certain that φ' will eventually become true also, for which UPPAAL has an efficient algorithm. One may thus use a query $C.s$ - -> $C.s'$ to check if it is certain to reach s' from s in C. A variant of this kind of formula is $C.s$ - -> $\neg C.s$, allowing to check that we cannot get stuck in location s. Applying this kind of formula to a loop needs however to *instrument* a bit the considered component, i.e., to add some features which do not modify the component's behaviour but allow

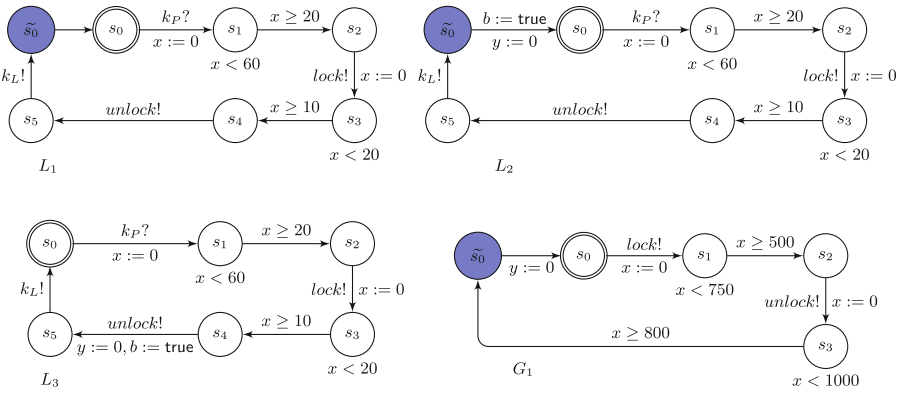

Fig. 3. Some instrumented versions of L: L_1 with added urgent location $\widetilde{s_0}$, L_2 with added urgent location $\widetilde{s_0}$, local clock y and Boolean variable b; L_3 with added local clock y, and Boolean variable b; and instrumented version G_1 of G with added urgent location $\widetilde{s_0}$ and local clock y.

to analyse it. Indeed, the formula $C.s$ - -> $C.s$ will always return *true*, because it is satisfied by the empty path, hence does not correspond to a true looping behaviour. A general technique to solve this kind of problem is to introduce an urgent location $\widetilde{s}$ (where time may not progress, represented in dark blue in the figures) before the considered location s, with all the arcs to s redirected to $\widetilde{s}$. This technique is general, but even if it does not modify the original behaviour, it increases the size of the state space since it introduces a new location. However, in some circumstances, if there is a location s_1 such that it is not possible to loop without visiting it, it is enough to check the pair of formulas $C.s$ - -> $C.s_1$ and $C.s_1$ - -> $C.s$.

As an example, the instrumented version L_1 of component L is illustrated in Fig. 3, and the first three lines of Table 1 give the liveness results for L and L_1.

In order to evaluate timing characteristics, we may use UPPAAL queries of the kind $\mathsf{sup}\{\varphi\}: x$ to compute the supremum (respectively, $\mathsf{inf}\{\varphi\}: x$ to compute the infimum) of clock x when formula φ is true. Note that there may be an asymmetry in their usage; for instance, $\mathsf{sup}\{C.s\}: C.x$ computes the supremum of clock x when *leaving* location s in component C, while $\mathsf{inf}\{C.s\}: C.x$ computes the infimum of clock x when *entering* location s in component C.

However, to make a good use of those queries it may again be necessary to instrument a bit the considered components. First, one should add new clocks allowing to measure the interesting paths, with resets put on the arcs entering the starting point(s), unless existing clocks already do the job. Next, one may need to add urgent locations, for instance when we need to consider the maximal time to enter a location and not the time to leave it. This may also be used to differentiate the various ways to enter a location, when there are many ones. Finally, one may introduce Boolean variables (initialised to *false*) to select paths satisfying some constraints.

Table 1. Model checking results, and execution times observed for a system with Intel Core i5 1.4 GHz and 4GB RAM. We denote by ex1-L_i, for $i = 1, 2, 3$, the specification composed of all the automata of Example 1 where component L is replaced by its instrumented version L_i allowing to perform the desired request. The interpretation of ex1-G_1 is analogous.

Model	Query	Result	Time [s]	Interpretation
ex1	$L.s_0$ - -> $L.s_2$	true	208	first half loop of L feasible
ex1	$L.s_2$ - -> $L.s_0$	true	139	second half loop of L feasible
ex1-L_1	$L_1.s_0$ - -> $L_1.\widetilde{s_0}$	true	345	loop feasible
ex1-L_2	$\mathsf{inf}\{L_2.\widetilde{s_0}\}\colon L_2.y$	> 70	148	lower bound of looping time
ex1-L_2	$\mathsf{sup}\{L_2.\widetilde{s_0}\}\colon L_2.y$	< 2550	150	upper bound of looping time
ex1-L_2	$\mathsf{inf}\{L_2.\widetilde{s_0} \wedge \neg L_2.b\}\colon L_2.y$	≥ 1330	148	lower bound of the first loop
ex1-L_2	$\mathsf{sup}\{L_2.\widetilde{s_0} \wedge \neg L_2.b\}\colon L_2.y$	< 2550	157	upper bound of the first loop
ex1-L_2	$\mathsf{inf}\{L_2.\widetilde{s_0} \wedge L_2.b\}\colon L_2.y$	> 70	158	lower bound of the next loops
ex1-L_2	$\mathsf{sup}\{L_2.\widetilde{s_0} \wedge L_2.b\}\colon L_2.y$	< 950	159	upper bound of the next loops
ex1-L_3	$\mathsf{inf}\{L_3.s_2 \wedge \neg L_3.b\}\colon L_3.y$	> 1320	110	lower bound of $s_0 \rightarrow s_2$ in the first loop
ex1-L_3	$\mathsf{sup}\{L_3.s_1 \wedge \neg L_3.b\}\colon L_3.y$	< 1720	111	upper bound of $s_0 \rightarrow s_2$ in the first loop
ex1-L_3	$\mathsf{inf}\{L_3.s_2 \wedge L_3.b\}\colon L_3.y$	> 60	112	lower bound of $s_5 \rightarrow s_2$ in the next loops
ex1-L_3	$\mathsf{sup}\{L_3.s_1 \wedge L_3.b\}\colon L_3.y$	< 120	113	upper bound of $s_5 \rightarrow s_2$ in the next loops
ex1-G_1	$\mathsf{inf}\{G_1.\widetilde{s_0}\}\colon G_1.y$	–	int. 1h	lower bound of looping in G_1
ex1-G_1	$\mathsf{sup}\{G_1.\widetilde{s_0}\}\colon G_1.y$	–	int. 1h	upper bound of looping in G_1
ex1-G_1	$G_1.s0$ - -> $G_1.\widetilde{s_0}$	–	int. 1h	loop feasible?

For instance, for the timing analysis of component L of Example 1, one may consider its instrumented version L_2, as shown in Fig. 3, and use the pair of queries $\mathsf{inf}\{L_2.\widetilde{s_0}\}\colon L_2.y$ and $\mathsf{sup}\{L_2.\widetilde{s_0}\}\colon L_2.y$ to get the lower and upper bounds (respectively) of the looping time: this is also illustrated in Table 1. However, since component P has an initialisation phase before entering its true looping part, one may suspect that the first loop of L and the next ones behave differently: this is confirmed by the next four queries in the table, where the Boolean variable b in L_2 allows to distinguish the first loop from the next ones (it is also possible to unroll explicitly the first iteration(s) of the loop in order to analyse them successively; this amounts for instance to replace a looping structure α^* by a structure $\alpha(\alpha^*)$ if we want to isolate the first iteration from the next ones; we shall not do it here, but a partial unrolling will be used in the next two sections, for other reasons). For further use, we are also interested in the time to go from s_5 to s_2, and in the time to first enter s_2 (from the initial state); this may be analysed with the instrumented version L_3 of L, also shown

on Fig. 3 and illustrated in Table 1: the Boolean variable b is used to distinguish the first time one enters s_2 (from s_0) from the next ones (from s_5).

Similarly, the bounds of the looping times of components G may be obtained from the instrumented version G_1, also shown on Fig. 3, and the results are detailed at the end of Table 1.

All the computations succeeded, except for the looping times of G, for which the executions were stopped after 1 hour, and the accelerations by over/under-approximations offered by UPPAAL do not help. It could happen that the bounds for the looping time of G may be obtained by allowing more execution time (and/or a much more powerful computer), but in any case this would likely be considered unsatisfactory by the end user, who probably would like to analyse many variants of the model.

Note that, to get bounds for the time needed to follow some path, one may also add the bounds for sub-paths, but the result is generally less accurate than the global estimation; this is due to the well known property that the $\mathsf{inf}(f_1 + f_2) \geq \mathsf{inf}(f_1) + \mathsf{inf}(f_2)$ and $\mathsf{sup}(f_1 + f_2) \leq \mathsf{sup}(f_1) + \mathsf{sup}(f_2)$ for any two functions f_1 and f_2, the equality being only obtained if the two functions are independent, or by mere chance, because the extrema are reached at the same points.

5 Direct Abstraction Strategy

We have seen in the previous section that the computation of timing characteristics may blow up. This is usually due to a combination of a complex system, in particular a highly distributed one, and ill balanced constants in the invariants and guards. We shall now consider strategies allowing to get around this unfortunate phenomenon in some circumstances, at least partly.

First, it may be observed that it is usually not necessary to know the exact infima and suprema, but only to be able to assert that some traveling time is not higher than some value, and possibly also not lower than some other value. Hence it is sufficient to determine an (approximate) interval $[min, max]$ encompassing the traveling time under consideration, i.e., to get an over-approximation of the true bounds. We here considered a closed interval, but sometimes we shall use open or semi-closed ones, when it is known that some extremum may not be reached.

Since one of the main sources of computation failure is the size of the system, we shall delineate a subsystem including the component we want to analyse and abstract away the interactions of this subsystem with the rest of the system, in order to isolate the subsystem and make its analysis feasible. This subsystem should be chosen with care: it should be small enough to allow the computations, but not too small to avoid uselessly large approximations (knowing that the travel time is in interval $[0, \infty]$ is not very useful). This may also be used to only keep in the subsystem components with similar constants. Also, we shall assume that the only interactions of the subsystem with the exterior is through rendez-vous on channels (no shared clock or variable), and that we are able to analyse the times taken by these communications.

In such a subsystem, we may distinguish internal components, which do not communicate with the exterior, and border components, which communicate both with the subsystem and with the exterior. The intuitive idea behind the method is then to replace those communications in the border components by one or more computation arcs, of the kind illustrated in Fig. 5, over-approximating the true time needed by those communications: guard $x \geq min$ and invariant $x \leq max$ ensure a delay in the interval $[min, max]$. Strict inequalities are also possible, for example guard $x > min$ and invariant $x < max$ ensure a delay in the interval (min, max), and analogously for other combinations. The values min and max, as well as the choice between strict or weak inequalities, should be provided by the analysis of the abstracted communication in the border component behaving in the full system.

In our example, since we want to analyse component G, we shall consider the subsystem $ss1$ (with the interior composed of M, H and G, and border component L), illustrated on top of Fig. 4. Note that the choice of the actual boundary of $ss1$ is quite arbitrary provided that it contains G, does not communicate with the rest of the system through clocks or variables, (which is the case here since all the clocks and variables are local) and the border may be analysed in the full system (which is the case in our example, as detailed in Table 1).

In general, if all the bordering components may be analysed in the full system, in order to abstract away the interactions with the exterior of the subsystem, the

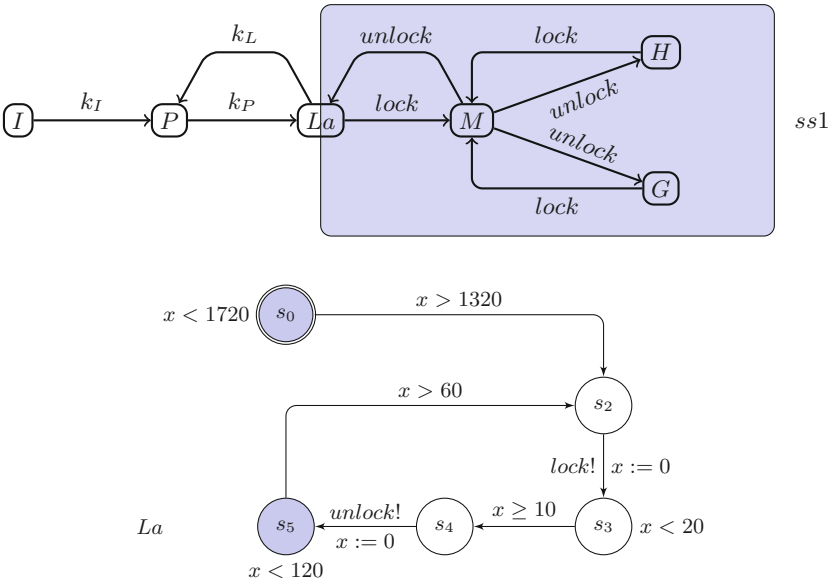

Fig. 4. Subsystem $ss1$ of Example 1, depicted within the communication scheme, and the border component La (obtained from L by abstracting the exterior of $ss1$, i.e., components I and P).

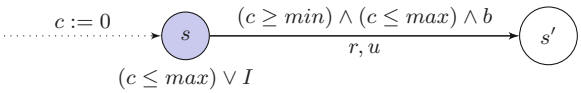

Fig. 5. A computation arc over-approximating (by an interval $[min, max]$ on some 'computation' clock c, with the guard b, the final clock resets r and the global variable updates u) the traveling through a communication arc or phase. Clock c is reset on all the incoming arcs to s and should not be reset by other components; it is in principle a new clock introduced for the abstraction, but it may also be an existing one that is available at that point. I materialises the invariants driving the other ways to leave s, usually in the form of a disjunction of inequalities of the kind $c \leq max'$ or $c < max'$.

most direct technique is simply to replace each communication arc ($k!$ or $k?$) with the exterior by an adequate computation arc, i.e., with an interval $[min, max]$ if we know from the analysis of the component that the communication takes between min and max time units. (see Fig. 5; note that constraints of the kind $0 \leq c$ and $c \leq \infty$ may be simply dropped, and that we assumed here to have inclusive extrema, while they may be exclusive.) Also, as already noticed, in order to avoid too large approximations due to an initialisation phenomenon, it may be useful to unroll the first loop execution(s).

It may happen that a same communication arc allows to interact both with the interior and the exterior of the subsystem (in our example, if the left border of $ss1$ was shifted right so that the border becomes component M and component L is now in the exterior, channels *lock* and *unlock* connect M both to the exterior L and to the interior G and H). In this case, we should duplicate those arcs in order to separate the communications (in our modified example, this would mean replace the *lock* in M by a choice between $lock1$ and $lock2$, $lock1$ being used in L and $lock2$ in G and H, and similarly for *unlock*).

The technique we just sketched is general but it uses the least possible granularity for the abstraction of the communications with the exterior of the subsystem, which unfortunately multiplies the intermediate computations (to evaluate the min/max values) and accumulates the propagations of approximations (due to the fact already mentioned that the sum of two mins/maxs may be lower/higher than the min/max of the sum). Hence, instead of abstracting away the outside communication arcs individually, we may search the coarsest possible abstraction. This amounts to search for the largest possible *phases* of communications with the exterior (see Definition 4) and to abstract each phase by a computation arc of the kind described in Fig. 5 for the individual abstractions.

Definition 4. *A communication phase is a part of a component with a first location s and a last one s' (possibly the same) together with intermediate locations and arcs: the arcs are either computation ones or communications with the exterior of the considered subsystem; they link locations of the phase, and all arcs to/from the intermediate locations belong to the phase; moreover, all the interactions with the exterior of the phase (through clock resets, variable updates and guards) should be equivalent to what happens with a single arc, so that traveling*

through the phase may be over-approximated by a single computation arc of the form described in Fig. 5.

For instance, for each clock reset during the phase, if its value is used outside the phase (before a further reset), one must have a reset on each arc to s' inside the phase (hence terminating the phase): all those resets yield the reset r. For each variable modified in the phase whose value is used outside the phase but in the same component (before a further modification), the final modification must be the same whatever the path from s to s'; if the value of the variable is used in another component of the subsystem, the (same) modification should only occur on all the arcs to s' inside the phase: those final modifications yield the updates u. Finally, the guard b used to summarise the guards inside the phase, should be implied by all the guards on the arcs from s inside the phase, and its value should not change during the rest of the phase. Note that we may have various phases between the same end locations s and s', and that a single communication arc always constitutes a phase by itself, but not often a maximal one.

For our example, in the (unique) border component L of subsystem $ss1$, the outside channels are k_L and k_P, and we may abstract the phase constituted by the whole path from s_5 to s_2, by a single computation arc. However, since the initial state is inside the abstracted path, we must also abstract the first time the path from s_0 to s_2 is ran; by the way, this also constitutes a partial unrolling of the first loop. We thus get the abstracted component La illustrated on the bottom of Fig. 4: the timing constraints for the computation arcs originating at locations s_0 and s_5 (depicted in blue) are those obtained by the analysis of L_3 in Table 1. The bounds for the looping time of G are then computed in the system $ss1$-La (i.e., $ss1$ with border component La). The results for the corresponding queries are presented in Table 2.

Table 2. Model checking results of G with the direct abstraction method. Notations for models are as in Table 1.

Model	Query	Result	Time [s]	Interpretation
ss1-La	$G_1.s_0$ - -> $G_1.\widetilde{s_0}$	true	284	loop feasible
ss1-La	$\mathsf{inf}\{G_1.\widetilde{s_0}\}: G_1.y$	≥ 8100	16	lower bound of looping in G_1
ss1-La	$\mathsf{sup}\{G_1.\widetilde{s_0}\}: G_1.y$	< 11750	16	upper bound of looping in G_1

It may be observed that, while the computation of the looping time of component G exploded for the full system, it becomes quite immediate in the simplified and abstracted subsystem $ss1$. Of course, it is not sure that the bounds we obtained are very tight with respect to the true infimum/supremum, but they may be satisfactory with respect to the question the practitioner will ask about the behaviour of G. In particular that means that there is no deadlock or starvation phenomena.

Proposition 1. Soundness of the direct strategy

The direct strategy, consisting in replacing each arc or phase of the subsystem communicating with the exterior of the considered subsystem by a computation arc with an interval encompassing the actual delay needed to cross it in the full system, provides a correct over-approximation of the subsystem.

Proof. Obvious since the true evolutions of the components in the subsystem (in the full system) are compatible with the ones in the isolated subsystem. As a consequence, if a traveling time in a component is larger than some constant and/or smaller than another one in the isolated subsystem, this is also true in the full system; in other words, the intervals obtained for the isolated subsystem are over-approximations of the true ones, in the full system. Similarly, if it is sure from some location to reach another one in the isolated subsystem, this is also true in the full system, since this means it is not possible to escape visiting the second location. □ 1

6 Iterated Abstraction Strategy

When one or more boundary components are too complex to be directly analysed, the technique developed in the previous section may be inefficient, because the only bounds we may use for them is $[0, \infty]$. However, we may circumvent the problem by analysing their abstractions as viewed both from the interior and the exterior of the considered subsystem.

Let us thus assume that, in a complex system to be model-checked, we consider a subsystem $ss1$ such that the direct analysis of some of its bordering components fails. Besides subsystem $ss1$, we shall also consider the subsystem $ss2$ composed of the components exterior to $ss1$ and the bordering one(s), as illustrated in the upper part of Fig. 6. If $ss2$ is still too complex, we may cut it in the same way, and at the end we shall get a family of small subsystems covering the whole system, with interior components, and with border ones communicating with at least two subsystems.

For each subsystem and each of its border components, we may then build the abstracted version of the latter, as viewed from this subsystem, in a way similar to what we have explained in the previous section.

The general idea of the iterated abstraction strategy is then to proceed in a succession of rounds. In each round, we consider successively each subsystem in some order, with each of its border components abstracting the exterior of the considered subsystem, replacing the communications with each exterior subsystem (arc or phase) by computational arcs, possibly after duplicating channels and unrolling the main loop. If the border component is analysable, the bounds and invariants will be derived as in the previous section; otherwise, they will be parameterised, and those parameters will be initialised in such a way that we are sure the induced behaviour encompasses the actual one (for instance we may use the interval $[0, \infty]$). When analysing this simplified component in its isolated subsystem, we may then estimate bounds for the communications between the

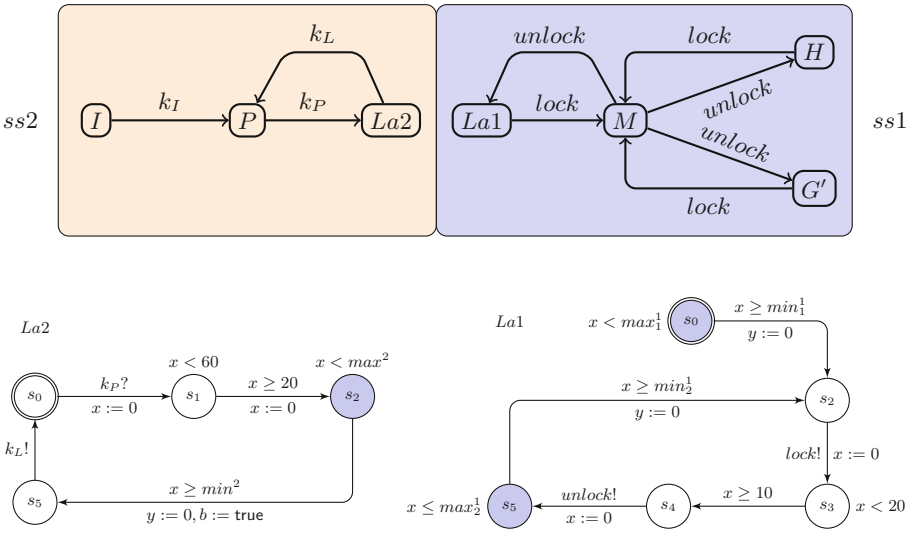

Fig. 6. Two subsystems $ss1$ and $ss2$ of Example 1' used for the iterated abstraction strategies, and the two versions of component L, abstracted as viewed from $ss2$ ($La2$) and from $ss1$ ($La1$). The superscripts for parameters min^i and max^i refer to subsystems, while subscripts identify different mins and maxs if necessary.

border component and this subsystem: this will be used to get better bounds for the abstractions of the same component when viewed from the subsystems to be considered next.

We proceed that way until no improvement is obtained when going from one round to the next one, i.e., when no bound is improved when analysing the various borders of the various subsystems, or when the practitioner considers the approximation obtained up to now is satisfactory with respect to his needs (in general, this means all the upper bounds are considered low enough), or desperate (in general because one of the lower bounds is too high, so that the situation will never be satisfactory, and it will be necessary to adapt the structure of the system).

One then may estimate bounds for the looping times (or traveling times) one is interested in. Note that, again, there is no guarantee that the obtained bounds will be very tight: all we know is that the true interval will be inside the result.

This is summarised in Algorithm 1.

Proposition 2. Soundness of the iterated strategy

Starting with $[0, \infty]$ intervals, Algorithm 1 terminates, and the successive stages of the successive rounds produce correct over-approximations of the original system.

Proof. Let us assume that, at the end of the first round, the obtained intervals are smaller or equal to the initial ones (this will in particular be the case if we start with $[0, \infty]$ intervals). Then, since the intervals at the beginning of the second round are the same than at the end of the first round, during the second round the possible evolutions are compatible with the specifications of the first round, so that at the end of the second round, the obtained intervals are smaller or equal to the ones obtained at the end of the first round. Iterating the reasoning we get that the successive rounds will yield a series of nested intervals.

If, moreover, the initial intervals encompass the ones in the original system (again, this will in particular be the case if we start with $[0, \infty]$ intervals), the same kind of argument as the one used in the proof of Proposition 1 shows that, at each stage, the evolutions of the original system are compatible with the approximate components (be it internal to a subsystem or a bordering one), so that the bounds obtained for the latter are correct.

If the bounds stabilise, the algorithm terminates (possibly before, if a satisfactory situation is reached). Since the bounds of each abstracted interval at each stage are natural numbers (or ∞ for the upper bound), the only way we shall have non-stabilisation is when the successive values of an interval are of the form $[k_i, \infty]$ with increasing but unbounded k_i's (this corresponds to a very particular case where it is impossible to reach the end of an abstracted part). But then the algorithm will terminate because the situation will be considered as desperate by the end user at some point. (Note that in our experiences, we always got stabilisation, and quite fast.) □ 2

In order to illustrate this technique, let us replace in our running example the component G by a slower one; we shall thus consider a system Example 1', which is as Example 1 except for G which has been replaced by G', in which the

Algorithm 1. Iterated abstraction method

Data: network of timed automata N, a property ϕ to be analysed in a subsystem S of N
Result: analysis of ϕ in an approximated subsystem S
Construct a family of subsystems including S, covering N;
foreach subsystem **do**
  Determine the bordering components of it and construct its abstracted version ;
end
Choose the initial values of the min/max parameters;
Choose an ordering of the subsystems;
repeat
  Analyse the subsystems cyclically, following the chosen ordering, and determine new approximate values for the min/max parameters;
until *no progress is made, or the situation is judged satisfactory or desperate by the end user*;
Analyse ϕ in the abstracted version of S, approximated with the final parameter values;

computation interval for s_1 has been changed to $[3100, 4200)$, and that of s_3 to $[5000, 7500)$. We also consider the instrumented version G'_1, defined in the same way as G_1.

Since automatic analyses are usually made complicated, even unfeasible, when there are different orders of magnitudes in the constants of a system, we may expect new difficulties with respect to the case of Example 1: indeed, all the queries of Table 1 now blow up.

We shall use the decomposition in subsystems illustrated on top of Fig. 6. There is thus still a unique border, L, and its (parameterised) abstracted variants as viewed by subsystems $ss1$ and $ss2$ are shown on the bottom of this figure.

The iteration then takes the form of rounds, which stabilise very fast:

Round 1:

Step 1: Since the problem arises from G', we shall first consider subsystem $ss2$, with L replaced by $La2$ (see Fig. 6), with the initial parameters $min^2 = 0$ and $max^2 = \infty$ (which amounts to drop the constraints on min^2, max^2). (Note that we could have used $min^2 = 10$, since a closer look at L shows that at least 10 time units are spent between *lock*! and *unlock*!, but this will not be necessary). We may then use the queries shown on top of Table 3 to obtain a first estimation of min_1^1, max_1^1, min_2^1 and max_2^1.

Step 2: With the bounds obtained in the previous step, analyse component $La1$ in $ss1$, and search for the bounds min^2 and max^2 with the next two queries in Table 3.

Round 2:

Step 1: With the bounds obtained in step 2 of round 1, analyse component $La2$ in $ss2$, and search for the next estimation of the bounds min_1^1, max_1^1, min_2^1 and max_2^1 with the next four queries in Table 3. Since no improvement is obtained with respect to the results of round 1, we may stop the iterations, and estimate the bounds for the looping time of G' and $La1$ in $ss1$, as shown by the last queries in Table 3 (for L, we do not need true computations: the approximate bounds for the loop, after some initialisation, is given by the sums of the bounds of the two half-loops, from s_2 to s_5 and from s_5 to s_2).

We may observe that, while none of the components L and G were analysable in the original system of Example 1', with our iterated abstraction strategy no computation took more than a few seconds (but the loop feasibility which takes a few minutes), leading to bounds that satisfied the practitioners at the origin of this kind of system.

Table 3. Results and execution times of the iterative abstraction process for Example 1'. Notations for models as in Table 1.

Round 1	**Step1**			
model	query	result	time [s]	interpretation
$ss2$-$La2$	$\mathsf{inf}\{La2.s_2 \wedge \neg La2.b\}\colon La2.y$	≥ 1320	< 1	min_1^1
$ss2$-$La2$	$\mathsf{sup}\{La2.s_1 \wedge \neg La2.b\}\colon La2.y$	< 1720	< 1	max_1^1
$ss2$-$La2$	$\mathsf{inf}\{La2.s_2 \wedge La2.b\}\colon La2.y$	≥ 60	< 1	min_2^1
$ss2$-$La2$	$\mathsf{sup}\{La2.s_1 \wedge La2.b\}\colon La2.y$	< 120	< 1	max_2^1
Round 1	**Step2**			
model	query	result	time [s]	interpretation
$ss1$-$La1$	$\mathsf{inf}\{La1.s_5\}\colon La1.y$	> 10	2	min^2
$ss1$-$La1$	$\mathsf{sup}\{La1.s_4\}\colon La1.y$	< 4280	2	max^2
Round 2	**Step1**			
model	query	result	time [s]	interpretation
$ss2$-$La2$	$\mathsf{inf}\{La2.s_2 \wedge \neg La2.b\}\colon La2.y$	≥ 1320	< 1	min_1^1
$ss2$-$La2$	$\mathsf{sup}\{La2.s_1 \wedge \neg La2.b\}\colon La2.y$	< 1720	< 1	max_1^1
$ss2$-$La2$	$\mathsf{inf}\{La2.s_2 \wedge La2.b\}\colon La2.y$	≥ 60	< 1	min_2^1
$ss2$-$La2$	$\mathsf{sup}\{La2.s_1 \wedge La2.b\}\colon La2.y$	< 120	< 1	max_2^1
	Final analysis			
model	query	result	time [s]	interpretation
$ss1$-$La1$	$G_1'.s_0$ - -> $G_1'.\widetilde{s_0}$	true	2125	loop of G' feasible
$ss1$-$La1$	$\mathsf{inf}\{G_1'.\widetilde{s_0}\}\colon G_1'.y$	≥ 8100	18	lower bound of looping in G'
$ss1$-$La1$	$\mathsf{sup}\{G_1'.\widetilde{s_0}\}\colon G_1'.y$	< 11750	18	upper bound of looping in G'
$ss1$-$La1$	$min_2^1 + min^2$	> 70	0	lower bound of looping in $La1$
$ss1$-$La1$	$max_2^1 + max^2$	< 4400	0	upper bound of looping in $La1$

7 Conclusion and Future Work

We proposed and showed the soundness of two abstraction methods allowing to accelerate (or make possible) the model-checking analysis of timed properties of components of networks of timed automata. We illustrated on a typical example how the abstraction strategy may help in analysing the timing properties of a network of timed automata when a state space explosion occurs. We also applied our iterative strategy on the behaviour of component L in our first example, with the same decomposition since the system has the same structure, and we observed that only a fraction of second is necessary to analyse each of the three

steps, compared to a few minutes in the direct analysis summarised in Table 1 (and in this case the obtained bounds are exactly the same).

Example 1 has not been chosen on purpose: it arose in one of our previous works, as a solution to deadlock problems occurring in a small but realistic model of augmented reality application, but we should of course examine how our techniques apply to larger realistic systems. We should also derive good ways to decompose a large system into subsystems of adequate size and characteristics. Finally, in addition to local analyses like the loop or travelling time bounds, we could be interested in the time needed to transfer an information from a source component to a consumer one, like in our example from a sensor to a rendering loop.

Acknowledgements. We are grateful to Jean-Yves Didier, Mathieu Moine and Johan Arcile for their ideas in the early stage of this work. We would like also to thank the anonymous referees for their careful reading and inspiring suggestions.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
2. Arcile, J., Didier, J., Klaudel, H., Devillers, R., Rataj, A.: Indefinite waitings in MIRELA systems. ESSS 2015, pp. 5–18 (2015). http://dx.doi.org/10.4204/EPTCS.184.1
3. Carnevali, L., Pinzuti, A., Vicario, E.: Compositional verification for hierarchical scheduling of real-time systems. IEEE Trans. Softw. Eng. **39**(5), 638–657 (2013)
4. Devillers, R., Didier, J.-Y., Klaudel, H.: Specifications, Implementing Timed Automata: The "Sandwich" Approach. ACSD 2013, pp. 226–235. IEEE (2013)
5. Devillers, R., Didier, J.-Y., Klaudel, H., Arcile, J.: Deadlock and temporal properties analysis in mixed reality applications. ISSRE 2014. IEEE, pp. 55–65 (2014)
6. Finkbeiner, B., Peter, H.-J., Schewe, S.: Synthesising certificates in networks of timed automata. IET Softw. **4**(3), 222–235 (2010)
7. Hendriks, M., Verhoef, M.: Timed automata based analysis of embedded system architectures. In: Workshop on Parallel and Distributed Real-Time Systems (2006)
8. Hendriks, M., Larsen, K.: Exact acceleration of real-time model checking. Electr. Notes Theor. Comput. Sci. **65**(6), 120–139 (2002)
9. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1–2), 134–152 (1997)
10. Möller, M.O.: Parking can get you there faster- model augmentation to speed up real-time model-checking. Electr. Notes Theor. Comput. Sci. **65**(6), 202–217 (2002)
11. Perathoner, S., et al.: Influence of different abstractions on the performance analysis of distributed hard real-time systems. Design Autom. Emb. Sys. **13**(1–2), 27–49 (2009)
12. UPPAAL. http://www.uppaal.org/