in *Proc. RTCSA Conference*, 2002, pp. 191-199.

Verification of Real-Time Embedded Systems using Petri Net Models and Timed Automata

Luis Alejandro Cortés, Petru Eles, and Zebo Peng
Dept. of Computer and Information Science
Linköping University, Linköping, Sweden
{luico,petel,zebpe}@ida.liu.se

Abstract

There is a lack of new verification methods that overcome the limitations of traditional validation techniques and are, at the same time, suitable for real-time embedded systems. This paper presents an approach to formal verification of real-time embedded systems modeled in a timed Petri net representation. We translate the Petri net model into timed automata and use model checking to prove whether certain properties hold with respect to the system model. We propose two strategies to improve the efficiency of verification. First, we apply correctness-preserving transformations to the system model in order to obtain a simpler, yet semantically equivalent, one. Second, we exploit the structure of the system model by extracting its the sequential behavior. Experimental results demonstrate significant improvements in the efficiency of verification.

1. Introduction

Embedded systems are typically constituted by heterogeneous components including both hardware and software elements. Besides their heterogeneity, embedded systems are characterized by dedicated function, real-time behavior, and high requirements on reliability and correctness [3].

For the levels of complexity typical to modern electronic systems, traditional validation techniques, like simulation and testing, are not sufficient to verify their correctness. Formal methods are becoming a practical alternative to ensure the correctness of designs.

In this paper we propose a verification method suitable for real-time embedded systems. Our approach allows the formal verification of systems represented in PRES+ [5] by using model checking. PRES+ is a Petri net based model extended to capture relevant characteristics of real-time embedded systems. We translate PRES+ models into timed automata in order to make use of available model checking tools.

The approach proposed in [5] translates PRES+ models into a collection of timed automata. One automaton with one clock variable is obtained for each transition. The main drawback of such an approach is that the complexity of model checking of timed automata is exponential in the number of clocks and thus verification is not feasible for medium or large-size systems.

We address in this paper two strategies to alleviate the complexity of verification. First, we present a transformational approach aimed at reducing the verification cost of systems modeled in PRES+. It makes use of correctness-preserving transformations in order to reduce the system model, so that the resulting one is simpler (still semantically equivalent to the original model) and therefore cheaper to verify. Thus if the simplified model is correct, the initial one is guaranteed to be correct.

Second, we propose a clustering algorithm that extracts the sequential behavior of the Petri net and thus the number of automata/clocks resulting after translating the PRES+ model can be reduced. In this manner we improve significantly the procedure to translate PRES+ models into timed automata presented in [5] and consequently the efficiency of the verification process. We also show how the efficiency of verification can be further improved by combining the transformational and clustering approaches.

Several analysis techniques based on time-extensions of Petri nets have been proposed [15], [13], [16], [10]. These approaches, though implementing efficient verification algorithms, are not totally suitable for real-time embedded systems since their modeling formalisms do not capture important features of such systems, for instance data dependencies as expressed by guards in PRES+.

The rest of this paper is organized as follows. A description of the design representation that we use to model real-time embedded systems is presented in Section 2. Our approach to the verification of systems described in PRES+ is introduced in Section 3, as well as the two strategies we propose in order to improve the efficiency of verification. Experimental results are presented in Section 4. Finally, some conclusions are drawn in Section 5.

2. Model

The notation we have defined for modeling embedded systems is PRES+ (Petri net based Representation for Embedded Systems). PRES+ overcomes some of the drawbacks of the classical Petri nets model: it captures explicitly

This research has been sponsored by the Swedish Agency for Innovation Systems (VINNOVA) in the frame of the SAVE project.

timing information; it is more expressive as tokens might carry information; systems may be represented at different levels of granularity; both control and data information may be captured by a unified design representation. PRES+ has been also extended by introducing the concept of hierarchy [7].

In this section we informally present the main characteristics of PRES+. The reader is referred to [5] for a formal definition of the model.

A *PRES+ model* is a five-tuple $N=(P, T, I, O, M_0)$ where P is a set of *places*, T is a set of *transitions*, I is a set of *input* (place-transition) *arcs*, O is a set of *output* (transition-place) *arcs*, and M_0 is the *initial marking* of the net. A marking is an assignment of tokens to the places of the net.

A *token* is a pair $k=\langle v, r\rangle$ where v is the *token value* (may be of any type) and r is the *token time* (a non-negative real number). Thus tokens carry data and time information attached to them as stamps. The *token type* associated to a place p, denoted $\tau(p)$, is the type of value that a token may bear in p. For the initial marking M_0 in the model shown in Figure 1, p_a is the only marked place and its token $k_a=\langle v_a, r_a\rangle$ has token value $v_a=2$ and token time $r_a=0$.

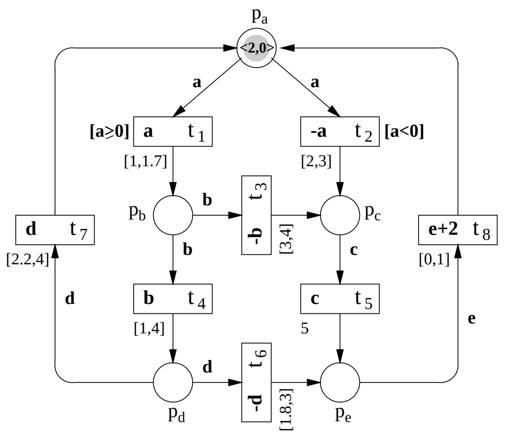

Figure 1. A PRES+ model

Every transition $t \in T$ has one *transition function* associated to it. Such a function takes as arguments the token values of tokens in the pre-set of the transition. The pre-set $^\circ t$ of a transition $t \in T$ is the set of input places of t. The post-set t° is the set of output places of t. Correspondingly, the pre-set $^\circ p$ and the post-set p° of a place $p \in P$ are the sets of transitions for which p is output and input place respectively. In Figure 1 we inscribe transition functions inside transition boxes: the function associated to t_8, for example, is given by $f_8(e)=e+2$ where e is the token value of the token in p_e when marked. We use inscriptions on the input arcs of a transition in order to denote the arguments of its transition function and/or those of its guard.

A transition $t \in T$ may have a *guard G*, a condition that must be satisfied in order to enable the transition when all its input places hold tokens. The guard of a transition is a function of the values of tokens in the places of its pre-set. For instance, $a < 0$ is the guard of t_2 in the example of Figure 1. Note that, for the initial marking, t_2 is not enabled even though its only input place is marked.

For every transition $t \in T$, there exist a *minimum transition delay* d^- and a *maximum transition delay* d^+. The non-negative real numbers $d^- \leq d^+$ represent the lower and upper bounds for the execution time (delay) of the function associated to the transition. Transition delays give the limits in time for the firing of a transition since it becomes enabled, unless it is disabled by the firing of another transition. Assuming in Figure 1, for instance, that t_1 fires at 1 time units and accordingly the token in p_a is removed and a new token $k_b=\langle 2, 1\rangle$ is deposited in p_b, then t_3 and t_4 become enabled at 1 time units. Thus t_3 may not fire before 4 time units and must fire before or at 5 time units, unless it becomes disabled by the firing of t_4. When a transition fires, all tokens in its output places get the same token value and token time.

3. Verification of Real-Time Embedded Systems

Model checking is an approach to formal verification used to determine whether the model of a system satisfies its *specification*, that is, certain required properties. The two inputs to the model checking problem are the system model and the properties that such a system must satisfy, usually expressed as temporal logic formulas.

For verification purposes, we restrict ourselves to *safe* PRES+ nets, that is, a place $p \in P$ may hold at most one token for a certain marking M. Otherwise, the formal analysis would become more cumbersome. This is a trade-off between expressiveness and analysis power.

Our approach allows to determine the truth of CTL (Computation Tree Logic) [4] and TCTL (Timed CTL) [1] formulas with respect to a (safe) PRES+ model. Formulas in CTL are composed of atomic propositions, boolean connectors, and temporal operators. Temporal operators consist of forward-time operators (**G** globally, **F** in the future, **X** next time, and **U** until) preceded by a path quantifier (**A** all computation paths, and **E** some computation path). TCTL is a real-time extension of CTL that allows to inscribe subscripts on the temporal operators to limit their scope in time. For instance, $\mathbf{AF}_{<n} P$ expresses that, along all computation paths, the property P becomes true within n time units. In our approach the atomic propositions of CTL/TCTL correspond to the marking of places in the net. Thus the atomic proposition p holds iff $p \in P$ is marked.

There are several types of analysis that can be performed on PRES+ models: the absence/presence of tokens in places of the net, time stamps of such tokens, and their token values. These analyses have been called reachability, time, and functional analysis respectively. Our approach to verification focuses on the first two, that is, reachability and time analyses. If the system model does not bear guards, we can ignore transition functions as reachability and time analyses will not be affected by token values.

In order to verify the correctness of a real-time embedded system, a systematic procedure to translate PRES+ into timed automata was proposed in [5] (in the sequel, this procedure is referred to as *naive translation*), so that it is possible to make use of existing model checking tools, namely HyTech [11], KRONOS [12], and UPPAAL [14]. In such an approach the resulting representation consists of a collection of timed automata which operate and coordinate with each other through shared variables and synchronization labels. One automaton with one clock is obtained for each transition. Since the complexity of model checking of timed automata is exponential in the number of clocks, that approach is not practical for large systems.

In this paper we improve the verification approach introduced in [5] in two different ways: applying a transformation-based concept in order to simplify the system model; exploiting the structure of the net by clustering transitions. These reduce considerably the complexity of the verification process.

3.1. Reduction of Verification Complexity by using Transformations

For the sake of reducing the verification effort, we first transform the system model into a simpler one, still semantically equivalent, and then verify the simplified model. If a given model is modified using correctness-preserving transformations and then the resulting one is proved correct with respect to its specification, the initial model is guaranteed to be correct, and no intermediate steps need to be verified. This simple observation allows us to reduce significantly the complexity of verification.

We can define a set of transformation rules that make it possible to transform only a part of the system model. A simple but useful transformation is shown in Figure 2. We do not intend to provide here a comprehensive set of transformations but rather illustrate the transformation of just a portion of the model (such a set of transformation rules has been defined in [8]). Assume that two subnets N' and N'' are *total-equivalent* in the sense defined in [6]. The intuitive idea behind *total-equivalence* is as follows (the reader is referred to [6] for a formal definition): (a) there exist bijections that define one-to-one correspondence between in(out)-ports[1] of N' and N''; (b) having initially tokens with the same token value and time in corresponding in-ports of N' and N'', there exists a firing sequence which leads to a marking with the very same token value and time in corresponding out-ports. It is not difficult to prove that N' and N'' are total-equivalent, provided the conditions given in Figure 2 are satisfied and that neither t_1 nor t are in conflict with any other transition. An interesting aspect for this transformation is that if the part of the system model represented by the subnet N' is replaced by N'', the overall behavior is the same in both cases. Such a transformation rule could be used, therefore, to simplify PRES+ models and accordingly reduce the complexity of the verification process.

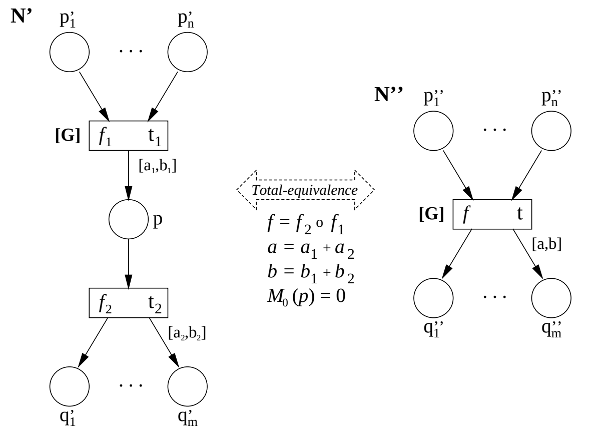

Figure 2. A simple transformation rule

We may take advantage of transformations to reduce the complexity of verification. The idea is to simplify the system model using transformations from a library. In the case of total-equivalence transformations, since an external observer could not distinguish between two total-equivalent nets (for the same tokens in corresponding in-ports, the observer would get in both cases the very same tokens in corresponding out-ports), the global system properties are preserved in terms of reachability, time, and functionality. Therefore such transformations are *correctness-preserving*: if a property P holds in a net that contains a subnet N'' (into which a total-equivalent subnet N' has been transformed), it does in another that contains N'; if P does not hold in the first net, it does not in the second either.

3.2. Reduction of Verification Complexity by Clustering Transitions

In order to reduce the number of automata/clocks resulted from the translation of PRES+ models into timed automata, we propose an algorithm that extracts the sequential behavior of the Petri net by *clustering* transitions. Intuitively, each *cluster* consists of a sequence of transitions where the firing of one of them *enables* the next one. The input of the algorithm is a safe Petri net and its output is a set of clusters, each representing a sequential part of the net. Then we obtain the timed automata, with one automaton and one clock per cluster (instead of one automaton and one clock per transition).

A *cluster* is an ordered tuple of distinct transitions denot-

[1] A place $p \in P$ is an *in-port* of the subnet $N=(P, T, I, O, M_0)$ iff $(t, p) \notin O$ for all $t \in T$. A place $p \in P$ is an *out-port* of N iff $(p, t) \notin I$ for all $t \in T$.

ed $C=(t_1, \ldots, t_n)$, such that t_{i+1} becomes enabled iff t_i fires, for $1 \le i < n$. We say that t_1 and t_n are, respectively, the *head* and the *tail* of C. In Figure 3, a possible cluster is $C=(t_1, t_3, t_5)$ with head t_1 and tail t_5.

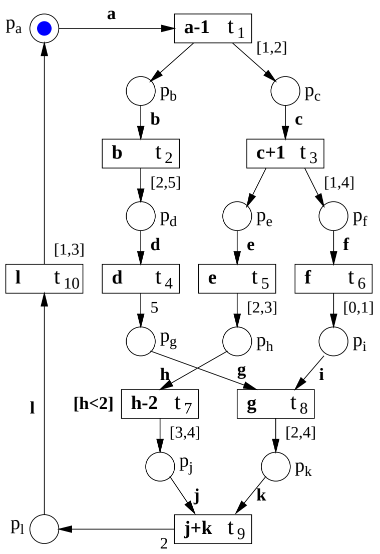

Figure 3. PRES+ model to be clustered

The *cluster set* S_C of a cluster $C=(t_1, \ldots, t_n)$ is the set of transitions that are components of C, that is $S_C=\{t_1, \ldots, t_n\}$. We explicitly make a distinction between *cluster* and *cluster set* because in the former case the order of the components is relevant whereas the order of elements in a set is immaterial. The objective of our clustering algorithm is to find a set of clusters such that their cluster sets form a partition of T (the set of transitions of the Petri net). In other words, we aim at finding a number of clusters such that each transition $t \in T$ is in one and only one cluster.

We define the *anterior set* of a transition $t \in T$, denoted $ant(t)$, as the set of those transitions that when fired will deposit a token in some place in the pre-set $^\circ t$, that is, $ant(t) = \bigcup_{p_i \in {}^\circ t} {}^\circ p_i$.

Similarly, the *posterior set* $post(t)$ of a transition $t \in T$ is the set of transitions that will get a token in some place of their pre-set when t is fired, that is, $post(t) = \bigcup_{p_i \in t^\circ} p_i^\circ$.

We define the *anterior set* $ant(C)$ of a cluster $C=(t_1, \ldots, t_n)$ as the anterior set of its head t_1, that is, $ant(C)=ant(t_1)$. The *posterior set* $post(C)$ of a cluster $C=(t_1, \ldots, t_n)$ is the posterior set of its tail t_n, that is, $post(C)=post(t_n)$. Consider, for example, the cluster $C=(t_{10}, t_1, t_3)$ in the net shown in Figure 3. Its anterior and posterior sets are $ant(C)=\{t_9\}$ and $post(C)=\{t_5, t_6\}$ respectively.

The clustering algorithm we propose tries to add a new head or tail to an existing cluster C. We keep a list of "free" transitions $freeT$, i.e. transitions not allocated yet to any cluster. Let $C=(t_h, \ldots, t_t)$ be a cluster with head t_h and tail t_t and let $freeT$ be the set of free transitions. We may add a *new tail* t_{nt} to the cluster C if $ant(t_{nt})$-$\{t_{nt}\}=\{t_t\}$ and $t_{nt} \in freeT$. We may add a *new head* t_{nh} to C if $t_{nh} \in freeT$ and $ant(C)$-$\{t_h\}=\{t_{nh}\}$. Consider again the example given in Figure 3. Assume this time $C=(t_9, t_{10}, t_1)$ and $freeT=T$-$S_C=\{t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$. Given that t_2, $t_3 \in freeT$ and $ant(t_2)=ant(t_3)=\{t_1\}$, both t_2 and t_3 fulfill the requirements for new tail stated above, but only one of them can be added as new tail to the cluster. In our algorithm this choice is made arbitrarily. If, for instance, t_3 is added to the cluster we obtain $C=(t_9, t_{10}, t_1, t_3)$ and $freeT=\{t_2, t_4, t_5, t_6, t_7, t_8\}$. Note that t_3 was removed from $freeT$. It is not hard to see that there is no transition to be added as new head of the cluster.

Our clustering algorithm starts by selecting arbitrarily a transition t from the free list. A new cluster C is formed so that t is initially both head and tail of C, and t is removed from $freeT$. The next step is to examine only those transitions in $post(C)$ that are also in $freeT$ and check whether they may be a new tail of C. If so, the cluster is enhanced by adding a new tail. We repeat the process until no new tail may be added to the cluster. Then, in a similar fashion, we try to enhance the cluster by adding a new head and repeat until there is no new head candidate in the free list. When the cluster can no longer be enhanced, we select another transition from $freeT$, form a new cluster, and repeat the process until all transitions have been allocated to a cluster. The clustering algorithm is shown in Figure 4.

```
clustering($safePN\ N$)
    set $freeT := T$
    while $freeT \neq \emptyset$ do
        with an arbitrary $t \in freeT$ do
            new cluster $C = (t)$
            set $freeT := freeT - \{t\}$
            set $newhead :=$ true
            set $newtail :=$ true
            // try to add a new tail $t_{nt}$
            while $newtail$ do
                set $newtail :=$ false
                with an arbitrary $t_{nt} \in post(C) \cap freeT$
                 such that $ant(t_{nt}) - \{t_{nt}\} = \{t_t\}$ do
                    add $t_{nt}$ to $C$
                    set $freeT := freeT - \{t_{nt}\}$
                    set $newtail :=$ true
                endwith
            endwhile
            // try to add a new head $t_{nh}$
            while $newhead$ do
                set $newhead :=$ false
                with $t_{nh} \in ant(C) \cap freeT$
                 such that $ant(C) - \{t_h\} = \{t_{nh}\}$ do
                    add $t_{nh}$ to $C$
                    set $freeT := freeT - \{t_{nh}\}$
                    set $newhead :=$ true
                endwith
            endwhile
        endwith
    endwhile
endclustering
```

Figure 4. Clustering algorithm

By applying our clustering algorithm on the system of Figure 3, we obtain the clusters $C_1=(t_9, t_{10}, t_1, t_2, t_4)$, $C_2=(t_3, t_5, t_7)$, $C_3=(t_6)$, $C_4=(t_8)$. Note that the output of the algorithm is not unique since there might be new-tail transitions chosen arbitrarily. We could also have got, for instance, $C_1'=(t_9, t_{10}, t_1, t_3, t_6)$, $C_2'=(t_2, t_4)$, $C_3'=(t_5, t_7)$, $C_4'=(t_8)$. However, in either case, the number of clusters is the same.

A simple analysis shows that the proposed algorithm has a (worst-case) time complexity $O(n^2)$, where n is the number of transitions in the net. We have applied the clustering algorithm to three different examples that can be scaled up. It is not our intention to discuss them here but rather use these examples in order to illustrate the performance of the algorithm in terms of execution time. Figure 5 shows the execution times of the clustering algorithm for the three cases studied.

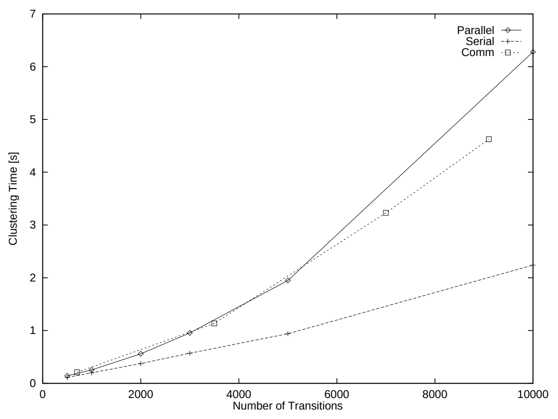

Figure 5. Performance of the clustering algorithm

3.3. Translating PRES+ into Timed Automata

A timed automaton is a finite automaton augmented with a finite set of real-valued clocks [2]. Timed automata can be thought as a collection of automata which operate and coordinate with each other through shared variables and synchronization labels. There is a set of real-valued variables, named *clocks*, all of which change along the time with the same constant rate. There might be conditions over clocks that express timing constraints.

An extended Timed Automata model (TA) can be expressed as $\overrightarrow{M}=(L, L_0, E, \Sigma, \sigma, X, V, \Phi, \upsilon, R, A, I)$, where

- L is a finite set of *locations*;
- $L_0 \subseteq L$ is a set of *initial locations*;
- $E \subseteq L \times L$ is a set of *edges*;
- Σ is a finite set of *labels*;
- $\sigma : E \rightarrow \Sigma$ is a mapping that labels each edge in E with some label in Σ;
- X is a finite set of real-valued *clocks*;
- V is a finite set of *variables*;
- Φ is a mapping that assigns to each edge $e=(l, l')$ a *clock condition* $\Phi(e)$ over X that must be satisfied in order to allow the automaton to change its location from l to l';
- υ is a mapping that assigns to each edge $e=(l, l')$ a *variable condition* $\upsilon(e)$ over V that must be satisfied in order to allow the automaton to change its location from l to l';
- $R : E \rightarrow 2^X$ is a *reset function* that gives the clocks to be reset on each edge;
- A is the *activity mapping* that assigns to each edge e a set of *activities* $A(e)$;
- I is a mapping that assigns to each location l an invariant $I(l)$ which allows the automaton to stay at location l as long as its invariant is satisfied.

In order to verify the correctness of system represented in PRES+, we translate the system model into timed automata so that model checking tools can be used. In what follows we describe the systematic procedure to translate PRES+ models into TA after clustering has been performed. The resulting model will consist of one automaton and one clock per cluster. The translation procedure that we propose here is correct as long as the underlying untimed Petri net is safe. We use the example of Figure 3 in order to illustrate the translation procedure, consisting of the following steps.

Step 1. Define one clock in X for each cluster. Define one variable in V for each place p_x of the Petri net, corresponding to the token value v_x when p_x is marked. ▪

Step 2. Define the set Σ of synchronization labels as the set of transitions in the Petri net. ▪

Steps 3 through 9 must be performed for each one of the clusters obtained by using the clustering algorithm. Consider a cluster $C=(t_1, \ldots, t_n)$ with head t_1 and tail t_n. For $t_i \in S_C$ (t_i denotes the i-th transition in cluster C), let f_i be the transition function associated to t_i, and let d_i^- and d_i^+ be the minimum and maximum transition delays associated to t_i. Let G_i be the guard associated to the transition t_i. Let v_x be the value of the token in the place p_x when marked. The timed automaton corresponding to the cluster C is denoted $\overrightarrow{C}$. The clock corresponding to $\overrightarrow{C}$ is denoted c. For the sake of clarity, we first present the translation steps for the simplest case: we initially assume that t_i is not in conflict with any other transition, for all $t_i \in S_C$, and that $(post(C)\text{-}\{t_n\}) \cap S_C = \varnothing$. Later we will discuss the general case where these assumptions do not hold.

Step 3. Define $m+n$ locations $a_1, \ldots, a_m, b_1, \ldots, b_n$, where $m=|ant(C)\text{-}\{t_1\}|$ and $n=|S_C|$. These are the locations of $\overrightarrow{C}$. Define m edges (a_j, a_{j+1}), for $j=1, \ldots, m\text{-}1$, with synchronization labels corresponding to the transitions in $ant(C)\text{-}\{t_1\}$. Define also m edges (a_m, b_1) with synchronization labels corresponding to the transitions in $ant(C)\text{-}\{t_1\}$. Then define one edge (b_i, b_{i+1}), for $i=1, \ldots, n\text{-}1$, with synchronization label t_i. Define one edge (b_n, a_1) with synchronization label t_n. ▪

Consider the cluster $C_1=(t_9, t_{10}, t_1, t_2, t_4)$ for the model given in Figure 3. We have $n=5$ for this cluster. Since $ant(C_1)=\{t_7, t_8\}$ we have $m=2$. Therefore, the automaton $\overrightarrow{C_1}$ corresponding to the cluster C_1 has 7 locations $a_1, a_2, b_9, b_{10}, b_1, b_2, b_4$ and its edges are as shown in Figure 6. Note that b_k corresponds to the location in which tran-

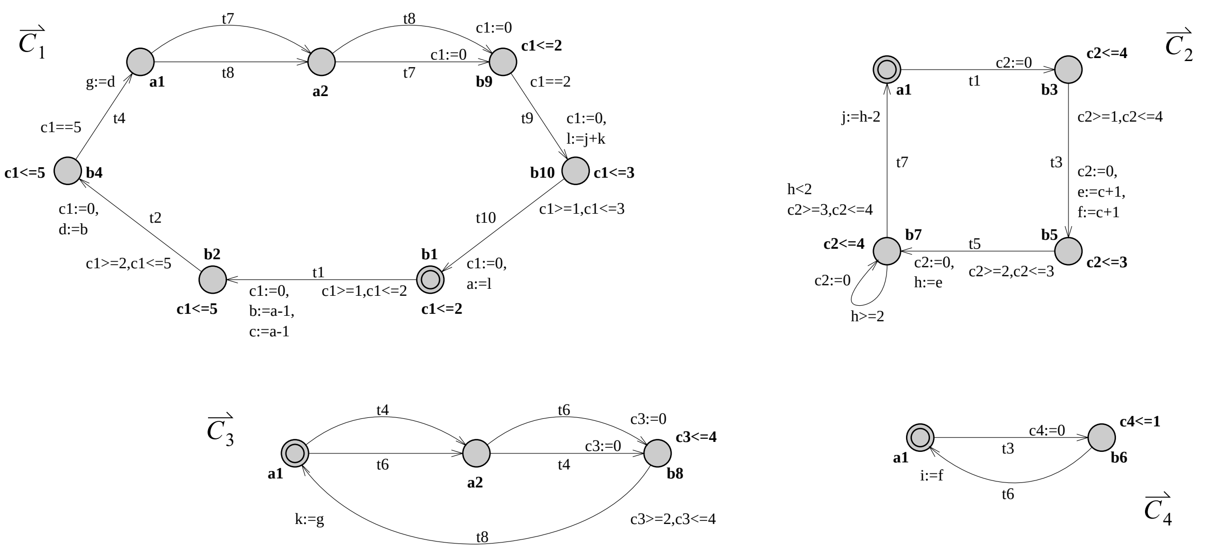

Figure 6. Equivalent timed automata

sition t_k is enabled (if t_k has no guard). The change of location, for example, from b_1 to b_2 corresponds to the firing of transition t_1.

Step 4. For every edge $e_j=(a_m, b_1)$ and every edge $e_i=(b_i, b_{i+1})$, $1 \le i < n$, define $R(e_j)=R(e_i)=\{c\}$. For any other edge e in $\overrightarrow{C}$, define $R(e)=\varnothing$. ■

This means that on all edges but (a_j, a_{j+1}), $1 \le j < m$, and (b_n, a_1) the clock c will be reset. In Figure 6, the assignment $c_k := 0$ represents the reset of clock c_k.

Step 5. For every location b_i, $1 \le i \le n$, define its location invariant as $c \le d_i^+$. ■

This enforces the firing of t_i before or at its latest trigger time.

Step 6. To every edge with synchronization label t_i, where $t_i \in S_C$, assign the clock condition $d_i^- \le c \le d_i^+$. ■

In Figure 6, for example, the edge (b_2, b_4) (with synchronization label t_2) of the automaton $\overrightarrow{C}_1$ has a clock condition $2 \le c_1 \le 5$ where 2 and 5 are the minimum and maximum transition delays of t_2.

Step 7. For every edge with synchronization label t_i, where $t_i \in S_C$, and for every $p_j \in t_i^\circ$ assign to such an edge the activities $v_j := f_i$. ■

For instance, the activities assigned to the edge (b_1, b_2) with synchronization label t_1 in the automaton $\overrightarrow{C}_1$ are $\mathrm{b} := \mathrm{a\text{-}1}$ and $\mathrm{c} := \mathrm{a\text{-}1}$, where $\mathrm{a\text{-}1}$ is the transition function of t_1.

Step 8. If the transition $t_i \in S_C$ has a guard G_i, assign the variable condition G_i to the edge with synchronization label t_i. Then add an edge $e=(b_i, b_i)$ with no synchronization label, variable condition $\overline{G_i}$ (the complement of G_i), and $R(e)=\{c\}$. ■

Note the variable condition $h < 2$ on (b_7, a_1) and the edge (b_7, b_7) in the automaton $\overrightarrow{C}_2$. This is due to the guard $h < 2$ of transition t_7.

Step 9. If the transition $t_i \in S_C$ is enabled in the initial marking, make the location b_i the initial location of $\overrightarrow{C}$. Otherwise, if there are k places initially marked in the preset $^\circ t_1$ of the head t_1 ($0 \le k < m$ so that t_1 is not enabled), make a_{k+1} the initial location of $\overrightarrow{C}$. ■

In our example, b_1 is the initial location of $\overrightarrow{C}_1$ because the transition $t_1 \in S_{C_1}$ is enabled in the initial marking of the net. The automaton $\overrightarrow{C}_2$ has a_1 as initial location because none of the transitions of the cluster $\overrightarrow{C}_2$ is initially enabled.

Observe that one and only one of the transitions of a given cluster will be enabled at a time. If two transitions in a cluster were enabled simultaneously, that would imply that the (underlying untimed) Petri net is not safe.

We have assumed that t_i is not in conflict with any transition, for all $t_i \in S_C$, and that $(post(C)\text{-}\{t_n\}) \cap S_C = \varnothing$. Now we discuss the cases in which these assumptions do not hold:

a) In case that $post(C)\text{-}\{t_n\} = \{t_1\}$ (the posterior set of the cluster tail is the singleton containing the cluster head) the automaton $\overrightarrow{C}$ will have n locations $b_1, \ldots, b_n$, where $n=|S_C|$, but no a_i locations. There will be additionally one edge (b_n, b_1) with synchronization label t_n and clock condition, variable condition, clock reset, and activities similar to the other edges (b_i, b_{i+1});

b) If one of the transitions $t_i \in S_C$ is in conflict with another transition t_c, just add to the automaton $\overrightarrow{C}$ one edge (b_i, a_1) with synchronization label t_c.

4. Experimental Results

This section presents two examples which we use in order to illustrate our verification approach and the proposed improvement techniques.

4.1. Ring-Configuration Processes

We illustrate our verification approach on a scalable example, comparing the technique based on a naive translation from PRES+ into automata introduced in [5], the transformational approach presented in Section 3.1, and the one formulated in Section 3.2 where the structure of the net is exploited.

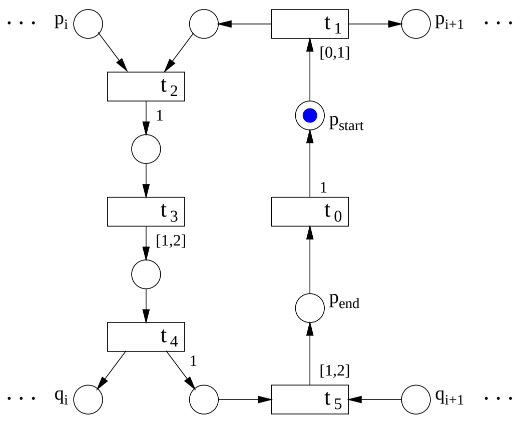

Figure 7. Model for one ring-configuration process

The example that we use represents a number n of processes arranged in a ring configuration. The model for one such process is illustrated in Figure 7. Each one of the n processes in the system has a bounded response requirement, namely whenever the process starts it must strictly finish within a time limit, in this case 25 time units. Referring to Figure 7, the start of one such process is denoted by the marking of p_{start} while the marking of p_{end} denotes the end of the process. This requirement is expressed by the TCTL formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25}\ p_{end})$.

We have used UPPAAL [14], running on a Sun Ultra 10 workstation, in order to model-check the timing requirements of the processes in the ring-configuration example. The results are summarized in Table 1.

Table 1. Verification of the ring-configuration example

Number of processes (n)	Verification Time [s]			
	Naive [5]	Transfor-mations	Clustering	Transf. + Clustering
2	1	<1	<1	<1
3	29	5	2	1
4	704	85	31	17
5	8700	1275	453	205
6	NA*	13260	5771	2295
7†	NA*	NA*	NA*	16200

* Not available: out of time

† Specification does not hold

The second column of Table 1 corresponds to the verification time using the approach of [5] (naive translation of PRES+ into timed automata). Note that only systems up to 5 processes can be handled with such an approach. The third column in Table 1 gives the results of verification when using the approach presented in Section 3.1: transformation of the model into a semantically equivalent and simpler one in order to reduce complexity, followed by naive translation into timed automata. The verification time for the example of processes in a ring configuration using the clustering approach of Section 3.2 is shown in the fourth column of Table 1. These results include the execution time of the clustering algorithm. By combining the clustering technique and the transformation-based one, it is possible to further improve the efficiency of the verification process as shown in the last column of Table 1. We have plotted all these experimental results in Figure 8.

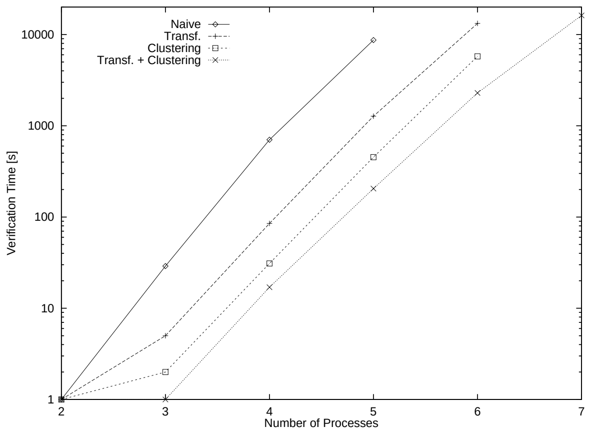

Figure 8. Verification of ring-configuration processes

Observe that for $n=7$ the bounded response requirement expressed by the formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25}\ p_{end})$ is not satisfied, a fact which, at first glance, is not obvious at all. An informal explanation is that since transition delays are given in terms of intervals, one process may take longer to execute than another; thus different processes can execute "out of phase" and this phase difference may be accumulated depending on the number of processes, causing one such process to take eventually longer than 25 time units (for $n=7$). It is also worth mentioning that, although the model has relatively few transitions and places, this example is rather complex because of its large (untimed) state space which is due to the high degree of parallelism.

4.2. Verification of a GMDFα

In this section we model and verify a realistic system: a GMDFα (Generalized Multi-Delay frequency-domain Filter) [9]. GMDFα has been used in acoustic echo cancellation for improving the quality of hand-free phone and teleconference applications. The GMDFα algorithm is a

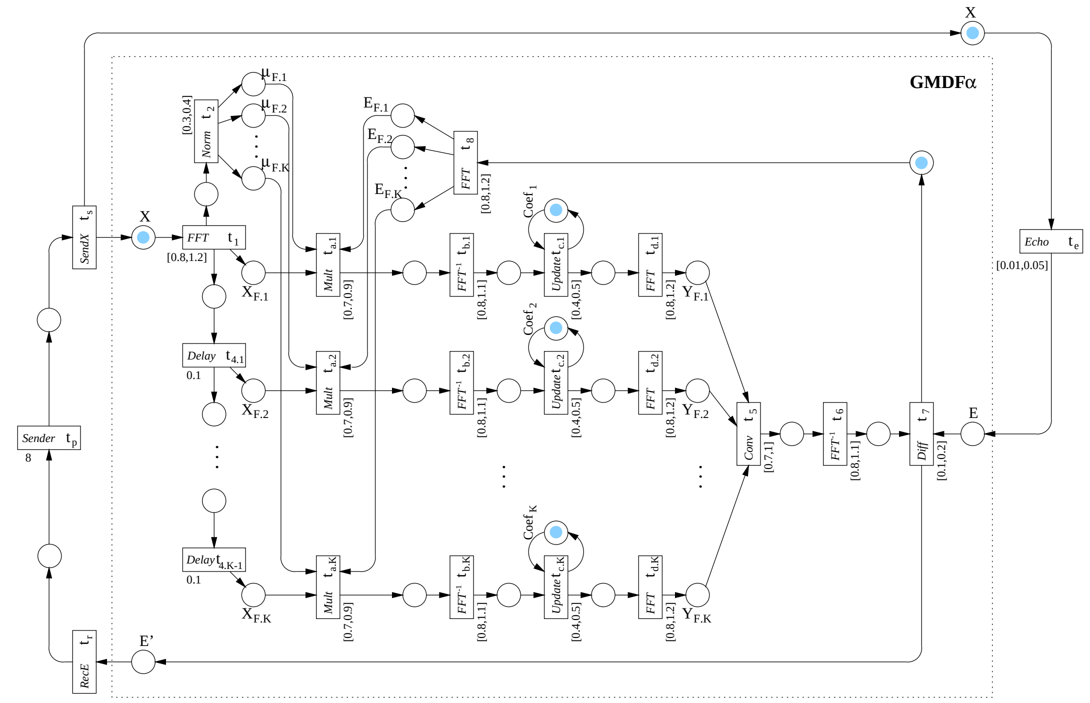

Figure 9. GMDFα modeled using PRES+

frequency-domain block adaptive algorithm: a block of input data is processed at a time, producing a block of output data. The impulse response of length L is segmented into K smaller blocks of size N (K=L/N), thus leading to better performance. R new samples are processed at each iteration and the filter is adapted α times per block (R=N/α).

The filter inputs are the signal X and its echo E, and the output is the reduced or cancelled echo E'. In Figure 9 we show the PRES+ model of the GMDFα. The transition t_1 transforms the input signal X into the frequency domain by a FFT (Fast Fourier Transform). t_2 corresponds to the normalization block. Transitions $t_{a.i}$, $t_{b.i}$, $t_{c.i}$, and $t_{d.i}$ constitute a basic cell, where a filter coefficient is updated and thus the filter is adapted by using FFT^{-1} and FFT operations. There are K instances of such a basic cell. Transitions $t_{4.i}$ serve as delay blocks. t_5 computes the estimated echo in the frequency domain by a convolution product and then it is converted into the time domain by t_6. The difference between the estimated echo and the actual one (signal E) is calculated by t_7 and output as E'. Such a cancelled echo is also transformed into the frequency domain by t_8 to be used in the next iteration when updating the filter coefficients. In Figure 9 we also model the environment with which the GMDFα interacts: t_e models the echoing of signal X, t_s and t_r represent, respectively, the sending of the signal and the reception of the cancelled echo, and t_p is the entity that emits X. Transition delays in Figure 9 are given in ms.

We consider two cases of a GMDFα of length 1024: a) with an overlapping factor α=4, we have the following parameters: L=1024, K=4, N=256, and R=64; b) with an overlapping factor α=2, we have the following parameters: L=1024, K=8, N=128, and R=64.

As seen in Figure 9, K affects directly the dimension of the model and, therefore, the complexity of verification. Having a sampling rate of 8 kHz, the maximum execution time for one iteration is in both cases 8 ms (64 new samples must be processed at each iteration). The completion of one iteration is determined by the marking of the place E'.

We want to prove that the system will eventually complete its functionality. According to the time constraint of the system, it is not sufficient to finish the filtering iteration but also to do so with a bound on time (8 ms). This aspect of the specification is captured by the TCTL formula $\mathbf{AF}_{<8}\ E'$.

Table 2. Verification of the GMDFα

GMDFα L=1024	Verification Time [s]			
	Naive [5]	Transformations	Clustering	Transf. + Clustering
α=4, K=4	108	1	2	<1
α=2, K=8	NA*	9	540	1

* Not available: out of time

By running UPPAAL on a Sun Ultra 10 workstation, we have verified that in both cases the specification formula $\mathbf{AF}_{<8}\ E'$ indeed holds. Table 2 shows the verification time for both cases (K=4 and K=8) comparing the techniques proposed in this paper with previous work. Observe how significant is the improvement, especially when the transformational and clustering approaches are combined.

5. Conclusions

We have presented an approach to formal verification of real-time embedded systems represented in PRES+. We make use of model checking to prove the correctness of such systems with respect to reachability and time, specifying design properties as temporal logic formulas. In order to use available model checking tools the Petri net model is translated into timed automata.

We have proposed two techniques aimed at improving the efficiency of verification. The first is an approach that makes use of correctness-preserving transformations in order to simplify the system model and, therefore, facilitate verification. The second one improves the translation procedure from PRES+ into time automata: we have proposed an algorithm of complexity $O(n^2)$ that extracts the sequential behavior of the net by clustering transitions; thus we obtain one automaton with one clock per cluster, instead of one automaton with one clock per transition.

Experimental results have demonstrated the worthiness of such improvement techniques, and that by combining the clustering strategy and the transformational approach the efficiency of verification is improved considerably.

References

[1] R. Alur, C. Courcoubetis and D. L. Dill, "Model Checking for Real-Time Systems," in *Proc. Symposium on Logic in Computer Science*, 1990, pp. 414-425.

[2] R. Alur, "Timed Automata," in *Computer-Aided Verification*, D. Peled and N. Halbwachs, Eds. *LNCS 1633*, Berlin: Springer-Verlag, 1999, pp. 8-22.

[3] R. Camposano and J. Wilberg, "Embedded System Design," in *Design Automation for Embedded Systems*, vol. 1, pp. 5-50, Jan. 1996.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," in *ACM Trans. on Programming Languages and Systems*, vol. 8, pp. 244-263, April 1986.

[5] L. A. Cortés, P. Eles, and Z. Peng, "Verification of Embedded Systems using a Petri Net based Representation," in *Proc. ISSS*, 2000, pp. 149-155.

[6] L. A. Cortés, P. Eles, and Z. Peng, "Definitions of Equivalence for Transformational Synthesis of Embedded Systems," in *Proc. ICECCS*, 2000.

[7] L. A. Cortés, P. Eles, and Z. Peng, "Hierarchical Modeling and Verification of Embedded Systems," in *Proc. Euromicro Symposium on Digital System Design*, 2001.

[8] L. A. Cortés, "A Petri Net based Modeling and Verification Technique for Real-Time Embedded Systems," *Licentiate Thesis*, Dept. of Computer and Information Science, Linköping University, Linköping, Sweden, 2001.

[9] L. Freund, M. Israel, F. Rousseau, J. M. Bergé, M. Auguin, C. Belleudy, and G. Gogniat, "A Codesign Experiment in Acoustic Echo Cancellation: GMDFα," in *ACM Trans. on Design Automation of Electronic Systems*, vol. 4, pp. 365-383, Oct. 1997.

[10] H. Hulgaard and S. M. Burns, "Efficient Timing Analysis of a Class of Petri Nets," in *Computer-Aided Verification*, P. Wolper, Ed. *LNCS 939*, Berlin: Springer-Verlag, 1995.

[11] HyTech: The HYbrid TECHnology Tool, `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`

[12] KRONOS, `http://www-verimag.imag.fr/TEMPORISE/kronos/`

[13] T. G. Rokicki and C. J. Myers, "Automatic Verification of Timed Circuits," in *Computer-Aided Verification*, D. L. Dill, Ed. *LNCS 818*, Berlin: Springer-Verlag, 1994, pp. 468-480.

[14] UPPAAL, `http://www.uppaal.com/`

[15] E. Verlind, G. de Jong, and B. Lin, "Efficient Enumeration for Timing Analysis of Asynchronous Systems," in *Proc. DAC*, 1996, pp. 55-58.

[16] T. Yoneda and B.-H. Schlingloff, "Efficient Verification of Parallel Real-Time Systems," in *Formal Methods in System Design*, vol. 11, pp. 187-215, Aug. 1997.