Verification of the correctness of Real Time systems specified with timed Petri nets

Piotr Szwed
Katedra Automatyki, Akademia Górniczo-Hutnicza, al. Mickiewicza 30, 30-059 Kraków
pszwed@ia.agh.edu.pl

Abstract. *This paper tackles the problem of the verification of the correctness of Real Time systems. In our approach a Real Time system is modeled as a timed Petri net. We specify requirements using another type of timed Petri net and the observation function that maps transitions in one net into another. The paper introduces both timed Petri net models, defines partial and total correctness and presents formal tools for their automatic verification.*

1. Introduction. The verification of the correctness of Real Time systems should cover two types of requirements: the first are related to an ordering of operations exhibited by the checked system, the second are the timing requirements. In our paper we propose how to specify both types requirements and how to verify them. The results described in this paper are the continuation of former works [1,2,3] on application of observation functions in specification and verification of concurrent systems.

The correctness problem consists of three objects: the *verified* Petri net modeling the examined Real Time system, the *criterion net* specifying the requirements and an observation function coupling the verified and the criterion net.

We use two models basing on places and transitions Petri nets (PT-net) enhanced with time: *PTRT* nets are used to model a distributed Real Time application while *PTR* nets specify requirements concerning the ordering as well as the timing of operations.

In both models classical PT-nets are extended with time counters which can be modified by an assignment and whose values can be tested in guards of transitions. Method of specifying timing requirements in *PTR* nets is closely related to timed automata approach [4]

We define the relative correctness problem using a linear observation function. The function maps selected transitions in verified system into sets of transitions of the criterion net. The results of the function are somehow similar to an application of restriction and remapping operators of CCS i CSP algebras [5, 6].

The paper is organized as follows. Chapter 2 gives the basic definitions concerning place and transitions Petri nets. Chapter 3 defines the *PTRT* model, Chapter 4 the *PTR* model. In Chapter 5 the notions of partial and total correctness are introduced. and the methods of relative correctness verification are discussed.

2. Petri nets. Petri net is a bipartite directed graph whose vertices belong to two disjoint sets: *places* and *transitions*. Arcs can bind only the places with transitions and transitions with places. *Marking* represents a state of a net. It is an assignment of non-negative values to places. Due to the graphical representation of Petri nets those values are called tokens. Marking M is a vector of non-negative integer numbers whose size is equal to the number of places. By $M(p)$ we will denote the number of tokens in the place p.

Definition 1. A Petri net is a tuple $PN = (P, T, F, W, M_0)$, where: P is a finite set of places, T is a finite set of transitions, $F \subset (P \times T) \cup (T \times P)$ is a finite set of arcs, $W : F \to \mathrm{N}$ is a function assigning weights to arcs, $M_0 : P \to \mathrm{N} \cup \{0\}$ is an initial marking. ■

Execution (*firing*) of a transition changes the net state (marking). When a transition fires, it moves tokens from its input places to output places. Number of removed and added tokens is determined by the arcs weights. A transition t removes $W(p_{\mathrm{in}}, t)$ tokens from its input place p_{in} and adds $W(t, p_{\mathrm{out}})$ tokens to the output place p_{out}. Firing a transition can not lead to negative markings. Thus a transition t is enabled and can fire, if for any its input place p holds $M(p) \geq W(p, t)$.

A marking is called *dead* if it enables no transitions. The set of enabled transitions for a given marking can contain a number of elements. In this way the Petri net can specify the nondeterminsim in the modeled system.

Transitions t_1 and t_2 are called *concurrent* for a marking M if both are enabled and for all their common input places the condition $M(p_i) \geq W(p_i, t_1) + W(p_i, t_2)$ holds.

A marking M_n is reachable from M_0 if there is a firing sequence $\rho = \langle t_1, t_2, \ldots, t_n \rangle$ that leads from M_0 to M_n. The set of all markings reachable from M_0 is denoted by $R(PN)$. The net is called *bounded* if $\exists\, k \in \mathrm{N} \,.\forall\, M \in R(PN) \,.\forall\, p \,.\, M(p) \leq k$.

Many properties of Petri nets can be verified by the analysis of their *matrix representation* [7, 8]. As any graph, a Petri net can be described by an incidence matrix. The incidence matrix A of a Petri net has $|\,P\,|$ rows and $|\,T\,|$ columns. Its elements $A(p\,,\,t)$ specify how many tokens are removed or added to a place p as the result from firing a transition t. The value of the matrix element $A(p\,,\,t)$ is calculated basing on the weight function W as: $A(p\,,\,t) = A(p\,,\,t)^{+} - A(p\,,\,t)^{-}$, where $A(p\,,\,t)^{+} = W(p,\,t)$ for $(p,t) \in F$; $A(p\,,\,t)^{-} = W(t,\,p)$ for $(t,p) \in F$.

The summary effect of a firing sequence ρ leading from the marking M_0 to M_n can be described as an integer valued vector $x_n \in \mathbf{R}^{|T|}$ whose i—th element is equal to the number of occurrences of transition t_i in the sequence ρ. As a consequence we obtain the state equation of a Petri net: $M_n = M_0 + A\,x$, where $\mathrm{x} \in \mathrm{X} = \mathbf{R}^{|T|}$.

We represent a behavior of a Petri net by nondecreasing sequences of vectors $s = \langle x_1, x_2, \ldots, x_j, x_{j+1}, \ldots, x_n, \ldots \rangle$. We will call vectors $x_j \in \mathrm{X}$ the *solutions*. An i—ths component $x_j(i)$ of a vector x_j reports the number of firings of a transition t_i starting from the initial marking.

The assumed representation of a net behavior is more general then the firing sequence, because it enables modeling instantaneous execution of a number of concurrent transitions in one step. It admits also multiple repetitions of the same element in a sequence.

W introduce temporal information by assigning to solutions additional attribute representing a moment, when corresponding net transitions occurred.

By a *timed solution sequence* we will denote a sequence of pairs (x_i, T_i) , where x_i belongs to a certain solutions space X, whereas T_i is a nonnegative number: $T_i \in \mathbf{R}$. For a given sequence $s = \langle (x_0,T_0), (x_1,T_1), \ldots, (x_{i-1}, T_{i-1}), (x_i, T_i), \ldots \rangle$, time values satisfy $T_0 = 0$ and $T_i < T_{i+1}$ for $i \geq 0$.

3. Modeling Real Time systems. In this chapter we will introduce *PTRT* nets, an extension to *PT* nets allowing to model a distributed multitasking Real Time application. An application consists of several active processes (tasks) that are statically assigned to multiple processors. We define tasks as disjoint sets of transitions. We assume that there is a global time that is common to all processors. The time of a processor is distributed among the tasks that are executed on it. Each transition is assigned a parameter defining the time of its execution. It is a single rational number.

In classical Petri net approach, a task state can be interpreted as the presence of tokens in a certain places of a PT-net modeling the whole application. We extend the task state by additional components: time counters (clocks): *delay* and *timeout*. First of them allows a simulation of a task suspension for a given period, the second is used to model operations with timeouts. A clock z can be set by an assignment of a constant being a nonnegative rational number to a time counter $z := c$. Single counter can be an argument of a comparison $z = 0$ or $z > 0$.

Valuation tv is a function that assigns to a set of time counters C nonnegative real numbers. Function $tv' = tv + t$, where $t \in \mathbf{R}$, is defined as follows: $tv'(z) = tv(z) - t$, if $tv(z) \geq t$ or $tv'(z) = 0$ in other case.

Definition 2. For a Petri net $PN = (P, T, F, W, M_0)$ *task* Θ_i is defined as $\Theta_i = (T_i, C_i, Guard_i, Set_{Di}, Set_{Ti})$, where $T_i \subset \mathrm{T}$. $C_i = \{delay_i, timeout_i\}$ is a set of time counters. We assume that their initial values are equal to zero. $Guard_i$: $T_i \rightarrow \{true,\ delay_i = 0,\ delay_i > 0,\ timeout_i = 0,\ timout_i > 0\}$ is a function, that assigns to each task transition a predicate taking time counters as arguments. Set_{Di}: $T_i \rightarrow \mathbf{C}^{0+}$ is a function that specifies nonnegative rational constant c, that should be used in the assignment $delay_i := c$ when a transition $t_j \in T_i$ is fired. Similarly, Set_{Ti}: $T_i \rightarrow \mathbf{C}^{0+}$, defines assignment constants for expressions $timeout_i := c$. ■

For a set of tasks $\Theta = \{\Theta_i\}$, their sets of transitions are disjoint and cover the whole set of transitions: $T_i \cap T_j = \varnothing$ for $\mathrm{j} \neq \mathrm{j}$ and $\bigcup T_i = T$.

In our model tasks are statically assigned to processors. A processor is defined by specifying tasks that are executed on it. We have assumed that:

1. Time of all processors advance in parallel. Execution of a transition belonging to a task consumes the processor's time.
2. If in a set of tasks assigned to a processor exists tasks that are *ready*, a transition of one of them should be immediately fired, what results in consuming the processor's time.
3. Transitions are atomic. While a task executes a transition, the other executed on the same processor are suspended.

The observable system behavior is a timed sequence of events signaling finalization of subsequent transitions. Transitions on a given processor are interleaved while transitions on different processors execute in parallel.

Definition 3. Let $\Theta = \{\Theta_i\}$ be a set of tasks defined for a Petri net *PN*. *Processor* is defined as $\Pi_I = (\Theta(\Pi_i), utc_i)$, where

- $\Theta(\Pi_i) \subseteq \Theta$ is a set of tasks assigned to a processor; we assume $\Theta(\Pi_i) \cap \Theta(\Pi_j) = \varnothing$
- utc_i – is an *unassigned time counter*. ■

The goal of unassigned time counters utc_i is to synchronize the time flow on processors. If we fire a transition α that is assigned to a processor Π_j and the time of its execution is t_α, then the global time advances by $\Delta gt = t_\alpha - utc_j$. Unassigned time counters of other processors Π_i, $i \neq j$ are increased by Δt. In the case when the value $utc_i + \Delta gt$ is greater then the maximum time of all transitions ready to execute on a processor Π_i, we treat the transition α as infeasible (any ready transition from the processor Π_i should terminate earlier.)

Definition 4. A *PTRT* net is defined as $PTRT = (PN, \Theta, \Pi, time)$, where:
$PN = (P, T, F, W, M_0)$ is a Petri net; Θ is a set of tasks; Π is a set of processors; $time\colon T \to \mathbf{C}^+$. ■

Timed Petri net *PTRT* is defined by specifying a set of tasks, a set of processors and the function *time* that assigns time of execution to transitions of underlying Petri net. A time of execution is always a rational number. The model introduces additional state members: time counters *delay* and *timeout* for tasks and unassigned time counters *utc* for processors. Those counters can have only values that are rational numbers and are bounded. Timers *timeout* are *delay* are bounded by rational constants used to initialize them. Unassigned time counters are bounded by the maximum execution time of transitions assigned to a given processor.

Due to the fact that execution time of all transitions and constants in assignment and comparison expressions are rational numbers, all time parameters can be transformed to integer values. As all timers in the model are bounded, the set of their valuations *TV* is finite.

By the state of *PTRT* net we will denote a pair (M, tv), where M is a marking of the Petri net *PN* and $tv \in TV$ is a valuation of time counters. We assume that initial values of all counters are zero.

Let us assume that *PN* component of a *PTRT* net is described with a sate equation $M_n = M_0 + A\,x$, where $x \in X = \mathbf{R}^{|T|}$. Any transition t_i in *PTRT* net can be described by a pair $(v, \Delta gt)$, where $v \in X$ is a vector whose i—th element is equal to 1, whereas Δgt is a change of the global time. Execution of a transition is accompanied by changing the current marking M and the valuation of time counters tv.

A behavior of a *PTRT* net is described by a timed solution sequence $s = \langle (x_0,T_0), \ldots,(x_{i-1}, T_{i-1}), (x_i, T_i),\ldots\rangle$, for which $x_0 = 0$, $T_0 = 0$ and $x_i = x_{i-1} + v_i$, $T_i = T_{i-1} + \Delta gt_i$, where $(v_i, \Delta gt_i)$ is a transition executed in the *i*—th step.

4. Modeling requirements. The PTR net is an extension of PT net allowing to specify temporal requirements. It introduces to the model a set of clocks and timing constraints using the clock values. This approach is related to the solutions from the theory of timed automata [4].

Clocks are variables taking nonnegative real values. Their goal is to store information on the time flow starting from the moment when they were reset. All clocks advance with the same speed.

Let Z be a set of clocks. Timing constraints δ belong to a set of predicates $\Phi(Z)$ defined recursively as:

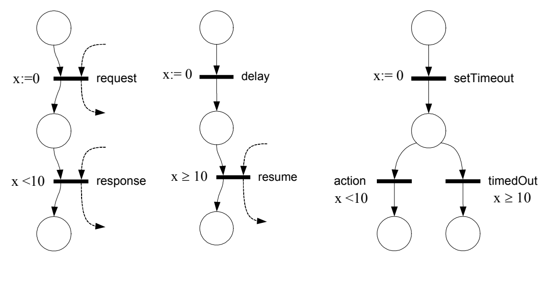
$\delta := true \mid z \leq c \mid c \leq z \mid \neg\,\delta \mid \delta_1 \wedge \delta_2$,

where $z \in Z$ is a clock, c is a constant from the set of nonnegative rational numbers $\mathbf{C}^+$.

Valuation $cv\colon Z \to \mathbf{R}^{0+}$ is a function that assigns nonnegative real values to clocks. Valuation cv satisfy timing constraint δ, if predicate δ is true for cv. For a clock $z \in Z$ we will denote by C_z the greatest constant appearing in the expressions $\delta := z \leq c \mid c \leq z$. Let $cv' = cv + t$ denote new valuation for clocks $z \in Z$ after a time t: $cv'(z) = cv(z) + t$, if $cv(z) + t \leq C_z$ or $cv'(z) = C_z$, in other case.

Clock values can grow until they reach the greatest constant appearing in the constraints, thus a clock z can have values from the range $[0, C_z]$. If a clock z reached a value greater that the constant C_z this would have no influence on the satisfaction of the constraints $\delta(z)$.

We show in Fig.1. how clocks can be used to formulate real-time requirements. Clock in Fig.1.a specify bounded response time. At the moment, when the transition *request* fires, the clock variable x is reset. The transition *response* meets timing requirements if: $T(response) - T(request) < 10$. (By $T(a)$ we denote the moment when transition a occurred.) A fragment of a net in Fig.1.b specify a delay. Execution times of subsequent transitions should satisfy: $T(resume) - T(delay) \geq 10$.



(*a*) (*b*) (*c*)

Fig. 1 Examples of using clocks in the specification of real-time constraints

A net in Fig.1.c specifies a transition that should be accomplished in a certain time interval. The sequence $\langle \ldots setTimeout, \ldots, action, \ldots \rangle$ is correct if

$T(action) - T(setTimeout) < 10$ holds, otherwise the transition *timedOut* should be observed.

Definition 5. *PTR* net is defined as a tuple $PTR = (PN, C, cv_0, \Phi(C), g, r)$, where $PN = (P, T, F, W, M_0)$ is a Petri net; C – is a set of clocks; cv_0 – is the initial valuation; $\Phi(C)$ – is a set of timing constraints. g: $T \rightarrow \Phi(C)$ is a function that assign condition expressions to transitions, in particular it may assign a guard *true*). r: $T \rightarrow 2^C$ is a function specifying which clocks should be reset after executing a transition. ■

Let us assume that *PN* (Petri net) component of a *PTR* is described with a state equation $M = M_0 + A\,x$, $x \in X = \mathbf{R}^{|T|}$. The *state* of a PTR net is a pair (M, vc), where M is a marking of the Petri net PN, vc is a valuation of clock variables.

Vector $v \in X$ represents a set of transitions if its elements have values from {0,1}. By $Guard(v,cv)$ we will denote a Boolean function that evaluates to *true* if all conditional expressions $g(t_i)$ for $v(i)=1$ are true for a given clock valuation cv. Analogously, by $Reset(v)$ we will denote the set of clocks that should be reset by execution of all transitions in v.

Let $e(i) \in X$ be a vector whose exactly one i—th element is equal to 1, other are 0. Vector $e(i)$ corresponds to a transition t_i. Pair $(v, \Delta\, t)$ is a *feasible step* of *PTR* net at the state (M, cv) if $\forall\, i.v(i)=1.\, M + A\, e(i) \geq 0$ and $Guard(v, cv + \Delta\, t)$ = *true*. The feasibility of a sets of transitions is based on two conditions: the transitions should be enabled and concurrent in the marking M and they should be feasible according to the timing constraints.

Let us define the function $newClokValue(cv, v, \Delta\, t)$, that calculates new clock values basing on their current state cv, executed transitions v and the change of global time $\Delta\, t$:

$cv' = newClockValue(cv, v, \Delta\, t) \Leftrightarrow$

- $cv'(z) = (cv{+}\Delta\, t)(z)$, if $z \notin Reset(v)$ or
- $cv'(z) = 0$, if $z \in Reset(v)$

Definition 6. We say that timed solution sequence $s = \langle (x_0,T_0),\ldots,(x_{i-1}, T_{i-1}), (x_i, T_i),\ldots\rangle$ is accepted by PTR net if

1. There exist a sequence of states of PTR net $\sigma = \langle (M_0,cv_0),\ldots, (M_i, cv_i),\ldots\rangle$, where $M_i = M_0 + A_I\, x_i$, and $cv_i = newClockValue(cv_{i-1}, x_i - x_{i-1}, T_i - T_{i-1})$
2. For all $i \geq 1$ transition $(x_i - x_{i-1}, T_i - T_{i-1})$ is a feasible step at the state (M_{i-1}, cv_{i-1}). ■

5. Relative correctness. For two spaces $X = \mathbf{R}^n$ and $X' = \mathbf{R}^m$ we define a matrix L of size of size $m \times n$ whose elements take values from the set {0,1}. The function $\pi : X \rightarrow X'$ defined as $\pi(x) = L{\cdot}x$ is called the *observation* function. If we treat X and X' as spaces appearing in the state equations of Petri nets, we can notice that the observation function π is capable of mapping transitions in one net to another. We use this property to formulate the correctness problem consisting of the elements:

- the verified net $PTRT = (PN, \Theta, \Pi, time)$ described by the state equation $M = M_0 + A\,x$, $x \in X$; this net models distributed real-time application;
- the criterion net $PTR = (PN', C, cv_0, \Phi(C), g, r)$ described by the state equation $M' = M_0' + A'\,x$, $x \in X'$;
- the observation function π: $X \rightarrow X'$

5.1 The correctness definition. The main idea of the presented approach is that we execute the net *PTRT* and obtain a timed solution sequence $s = \langle (x_0,T_0),\ldots,(x_{i-1}, T_{i-1}), (x_i, T_i),\ldots\rangle$ representing its behavior. Then we transform it to the X' space using the observation function obtaining the timed solution sequence: $s' = \langle (\pi(x_0),T_0),\ldots,(\pi(x_i), T_{i1}),,\ldots\rangle$. The sequence s representing a behavior of the verified net is considered correct if it is accepted by the criterion *PTR* net specifying requirements regarding correct sequences of operations as well as their timings.

Let us define a predicate $dead(x, PN)$, that evaluates to true, if the marking $M = M_0 + A\,x$ is dead. We use additional information on how the net models underlying system to distinguish markings, that can be identified as correctly reached final states. We define a predicate $final(x, PN)$ that is true if the marking $M = M_0 + A\,x$ is dead and only selected elements of the vector M has non-zero values.

Definition 7. For Let $s = \langle (x_0,T_0),\ldots,(x_i, T_i),\ldots\rangle$ be a timed solution sequence of a net PTRT.

The sequence s is correct with respect to the observation function π and the criterion net *PTR*, what we denote by the predicate $correct(s, \pi, PTR')$ if

1. The timed solution sequence $s' = \langle (\pi(x_0),T_0),\ldots,(\pi(x_i), T_{i1}),,\ldots\rangle$ is accepted by the net *PTR*.
2. There is no such element x_i in the sequence that $dead(x_i, PN) \wedge \neg final(x_i, PN)$
3. $\forall\, j \geq 0\,.\ final(x_j, PN) \Rightarrow final(\pi(x_j), PN')$. ■

The second condition is introduced to prevent the situation when an infinite solutions sequence reaches a marking that is dead and not final. Occurrence of such marking can be considered as finding a deadlock state.

Definition 8. Let us denote by $S(PTRT)$ the set of all timed solutions sequences of the net *PTRT*. The net *PTRT* is partially correct relatively to *PTR* if the following condition holds:
$\forall\, s \in S(PTRT)\,.\ correct(s, \pi, PTR)$ ■

One of the properties of an observation function is that it can make a transition in verified system non-observable. Such non observable transition corresponds to a column of matrix of observation function L containing only zero values. If non-observable transitions in verified net build up a loop, after executing them the verified net returns to the same marking and is ready to continue the loop infinite number of times. In the same time we do not observe any transitions and as the consequence the marking of the criterion net does not change. From the observers point of view the system does not manifest any desired activity although its still alive. Such situation can be referred as a violation of the *livness* property (we expect that a desired transition occur and it does not happen) . Following the CSP [5] we denote such situation with the term divergence.

In the case of various types of clocks present in both *PTRT* and *PTR* specifications the conditions for divergence are slightly more complicated, because they should take into account valuation of clock variables.

Definition 9. Let $s = \langle (x_0,T_0),\dots,(x_i, T_i),\dots\rangle$ be a timed solution sequence of a verified net *PTRT*. Let $\sigma = \langle (M_0,tv_0),\dots, (M_i, tv_i),\dots\rangle$ be a corresponding sequence of states of the *PTRT* net. For timed sequence $s' = \langle (\pi(x_0),T_0),\dots,(\pi(x_i), T_i),\dots\rangle$ σ' is a sequence of states of the criterion *PTR* net $\sigma' = \langle (M'_0,cv_0),\dots, (M'_i, cv_i),\dots\rangle$

Observation of the sequence s is divergent, if it contains a subsequence $s_{div} = \langle (x_{div1},T_{div1}),\dots,(x_{divn}, T_{divn}), \rangle$ satisfying:

1. $M_{div\,1} = M_{div\,n} \wedge tv_{div\,1} = tv_{div\,n}$
2. $\forall\, i : div_1 \le i \le div_n\, .\, M_i' = M_{div1}' \wedge cv_i = cv_{div1}$ ■

We define a predicate *divergent*(s, *PTRT*, π, *PTR*), which is true, if for a given observation function π and the criterion net *PTR* a timed sequence of solutions s of the net *PTRT* is divergent.

Definition 10. Let *PTRT* be a verified net, *PTR* a criterion net and π an observation unction. The net *PTRT* is totally *correct in* relation to *PTR* if:

1. It is partially correct
2. The observation of its behavior covers the whole space X': $\forall\, i \le dim\, X'\, .\, (\exists\, s \in S(PTRT)\, .\, \exists\, x_k \in s.\ x' = \pi(x_k) \wedge x'(i) > 0)$, where *dim* X' denotes the size of the space X' .
3. There is no divergence: $\forall\, s \in S(PTRT)\, .\, \neg\, divergent(s, PTRT, \pi, PTR)$ ■

5.2 Relative correctness verification. Automatic correctness verification for linear observation function is based on the construction of the *graph of coupled net execution G* describing synchronous execution of both verified and criterion nets [1,2,3]. Analysis of the properties of this graph allows to determine partial correctness of the verified system.

Definition 11. For a verified Petri net *PTRT* described by state equations $M = M_0 + A\cdot x$, $x \in X$, a criterion net *PTR* with corresponding state equation $M' = M_0' + A'\cdot x'$, where $x' \in X'$ and an observation function π we define the graph of coupled net execution G as the tuple $G = (P, E, F, s, t)$, where

- P is a set of graph vertices
- $E \subset P \times P$ is a set of edges
- $F \subset P$ is a set of final (leaf) vertices
- s: $P \rightarrow (\mathbf{M} \times TV) \times (\mathbf{M'} \times CV)$, is a function that assigns states of both nets to the graph vertices; $\mathbf{M}$ is a set of all markings of the net *PTRT*, TV is a set of all valuations of their clocks; similarly, $\mathbf{M'}$ is a set of all markings of a net PTR and CV the set of all valuations of their clocks.
- t: $E \rightarrow X \times X' \times \mathbf{C}^+$ is a function that assigns tuples $(v, \pi(v), \Delta\, t)$ to edges; v is a vector representing a transition in the verified net, $\pi(v)$ its image in observation function, Δ t is a change of the global time. ■

The graph of coupled net execution G is constructed as an upper bound of a sequence $G_0, G_1,\dots,G_n,\dots$. We start with G_0 $(P_0, E_0, F_0, s_0, t_0)$ satisfying: $P_0 = \{p_0\}$, $E = \varnothing$, $F = \varnothing$, $s(p_0) = (M_0, tv_0, M_0', cv_0)$, $t = \varnothing$.

At i—th step we select a vertex p from the set $P \setminus F$, for which there exist an enabled transition v_i not assigned to edges starting form p: $\forall\, e=(p,\bullet) \in E.\ \forall t(e) = (v_e,\pi(v_e),\Delta\, t_e)\ v_e \neq v_i$. From tv state component in $s(p) = (M, tv, M', cv)$ the change of global time $\Delta\, t_i$ and new clock valuation tv_i are evaluated. Then we check if the timed transition $(v_i, \Delta\, t_i\ )$ is feasible in *PTRT* model. For feasible transition we calculate $\pi(v_i)$, $M_i' = M' + A'\,\pi(v_i)$ and $cv_i = newClockValue(cv, \pi(v_i), \Delta\, t_i)$.

New graph components are a vertex p_i and an edge (p,p_i). Relations between the graph G_{i-1} and G_i are as follows:

- $P_i = P_{i-1} \cup \{p_i\}$
- $E_i = E_i \cup \{(p,p_i)\}$
- $s_i = s_{i+1} \cup \{(s_i, (M_i,tv_i,M_i',cv_i)\}$
- $t_i = t_{i+1} \cup \{((p,p_i), v_i, \pi(v_i), \Delta\, t_i)\}$

The new vertex p_i is added to the set F_i in the following situations:

1. There exist a vertex $p_k \in P_{i-1}$ such that $s_i(p_k) = s_i(p_i)$. That means that we have reached a vertex that has been already analyzed or that has been scheduled for further analysis.
2. There exist a vertex $p_k \in P_{i-1}$ assigned a coupled net state (M_k, tv_k, M_k', cv_k) such that $M_k = M_i$, $tv_k = tv_i$, $M_k' < M_i'$ and $cv_k = cv_i$. This case is similar to the previous with one difference: The marking M_i' covers M_k'. However, for all successors of the state (M_i',cv_i) the *PTR* net will be capable of accepting the same transitions as for (M_k',cv_k).
3. There exist a vertex $p_k \in P_{i-1}$ assigned a coupled net state (M_k, tv_k, M_k', cv_k) such that $M_k < M_i$. In this case the verified net is not bounded (safe). We treat this case as undecidable.

4. The step ($\pi(v_i)$, Δt_i)) is not accepted by the *PTR* net in the state (M_i',cv_i). In this case we state incorrectness of the verification problem and a path leading from p_0 to p_i serve as a counterexample.

Theorem 1. Graph of coupled net execution G is finite. ■

Sketch of the proof: It can be noticed that the rules of adding vertices to the set *F* provide that the graph is acyclic and prevent unbounded growth of markings *M* and *M'*. The sets of possible valuations of clocks in *PTRT* and *PTR* models are also bounded and finite. This provides that there is no infinite strongly growing subsequence of G_0, G_1,…,G_n,…. Thus applying the Koenig's lemma, see [8] the graph *G* is finite.

Theorem 2. If the problem is decidable and all edges of the graph *G* are marked with correct transitions, the verified net *PTRT* is partially correct relatively to π and *PTR*. ■

Sketch of the proof: It can be observed that any timed solutions sequence is represented by a path in the graph or can be constructed by the concatenation of several graph paths. After reaching a vertex in the set *F* we can continue exploration from the vertex which is assigned the same state values.

Technically, in the implemented prototype software for correctness verification we do not construct the full graph *G*. Instead, we remember only the current sequence of solutions and coupled states corresponding to a path in a graph. We remember also a set *S* of encountered coupled net states: (*M*,*tv*,*M'*,*cv*). At each step we try to calculate new element of the current sequence by execution of a transition that is enabled in the last state. If the transition is incorrect or a deadlock occurs, the current sequence of solutions and states serves as a counterexample. If the selected transition leads to a state already present in *S* (see above conditions 1 and 2) we remove one or more last elements from the current solutions sequence until we can select a new transition to execute. The process finishes when the current sequence of solutions and states becomes empty.

6. Conclusions. The paper describes a relative correctness problem for Real Time systems and presents the formal tools that are the basis for the implementation a software for correctness verification. We model a Real Time system as a timed Petri net. To describe the system requirements we use a Petri net extended with time counters. Specification of timing requirements is similar to the approach from the theory of timed automata [4]. Using the matrix representation of Petri nets we define the partial correctness as the correctness of all timed solutions sequences. The satisfaction of total correctness requires additionally the lack of divergences in observations and covering all transitions in the criterion net specifying the requirements.

A graph of coupled execution is a formal tool for relative correctness verification. The verification algorithm constructs this graph exploring a finite state space being a Cartesian product of state spaces of both nets. In case of large size problems it is possible to apply partial search algorithms, e.g. those from [11].

The paper was supported by the KBN grant Nr *4T11C 035 24 Zastosowanie metod formalnych do wspomagania wytwarzania poprawnego oprogramowania systemów czasu rzeczywistego.*

References.

[1] Szwed, P.: *Analiza poprawności oprogramowania współbieżnego z wykorzystaniem funkcji obserwacji*, praca doktorska, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki AGH, Kraków 1999

[2] Szwed, P.: *Zastosowanie liniowej funkcji obserwacji do analizy poprawności oprogramowania współbieżnego*, Materiały VII Konferencji Systemy Czasu Rzeczywistego, Kraków 2000,Katedra Automatyki Akademii Górniczo-Hutniczej, 99–108

[3] Szwed, P.: Zastosowanie liniowej funkcji obserwacji do analizy poprawności oprogramowania czasu rzeczywistego, w Pod red. Szmuc T. Werewka J. Analiza i projektowanie systemów komputerowych czasu rzeczywistego o różnym stopniu rozproszenia, Kraków 2001

[4] Alur, R., Dill, D.: *A Theory of Timed Automata*, Theoretical Computer Science, 126; 1994, 183-235

[5] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs, 1985

[6] Milner, R.: *Communication and Concurrency.* Prentice Hall, Englewood Cliffs, 1989

[7] Murata T.: *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol.77, No 4, April 1989

[8] Reisig W.: *Petri Nets – An Introduction*, EATCS Monographs on Theoretical Computer Science, Volume 4. Springer. 1985

[9] Holzmann, G.J., *An improved reachability analysis technique*, Software Practice and Experience, Vol. 18, No. 2, 1988, 137–161