Scenario-Based Timing Consistency Checking for Time Petri Nets*

Li Xuandong, Bu Lei, Hu Jun, Zhao Jianhua, Zhang Tao, and Zheng Guoliang

State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology
Nanjing University, Nanjing, Jiangsu, P.R. China 210093
lxd@nju.edu.cn

Abstract. In this paper, we solve the consistency checking problems of concurrent and real-time system designs modelled by time Petri nets for the scenario-based specifications expressed by message sequence charts (MSCs). The algorithm we present can be used to check if a time Petri net satisfies a specification expressed by a given MSC which requires that if a scenario described by the MSC occurs during the run of the time Petri net, the timing constraints enforced to the MSC must be satisfied.

1 Introduction

Scenarios are widely used as a requirements technique since they describe concrete interactions and are therefore easy for customers and domain experts to use. Scenario-based specifications such as message sequence charts offer an intuitive and visual way of describing design requirements. Message sequence charts (MSCs) [1] is a graphical and textual language for the description and specification of the interactions between system components. The main area of application for MSCs is as overview specification of the communication behavior of real-time systems, in particular telecommunication switching systems.

Time Petri nets [3] have been proposed as one powerful formalism for modelling concurrent and real-time systems because they can model both concurrency and real-time constraints in natural way. There are plenty of applications of time Petri Nets in modelling system specifications and designs.

Since Unified Modelling Language (UML) [2] became a standard in OMG in 1997, MSC-like diagrams (UML sequence diagrams) and time Petri nets-like models (UML activity diagrams) have become a main class of artifacts in software development processes. It follows that we often need to use MSCs and time Petri nets together in specification and design of software projects [4-6]. Usually, MSCs and time Petri nets are used in the different software development steps. Even used in the same step, e.g. requirements analysis, MSCs are used

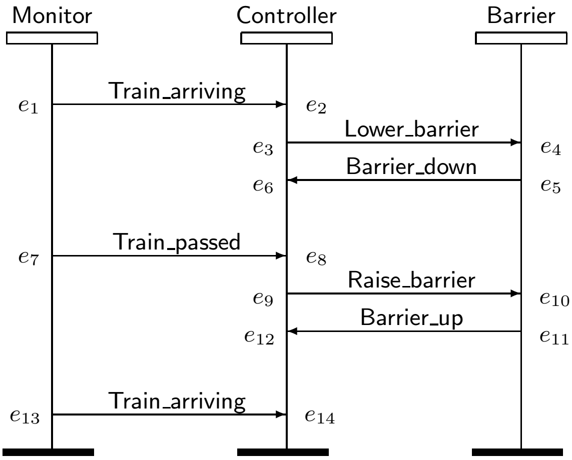
* Supported by the National Natural Science Foundation of China (No.60425204, No.60233020), the National Grand Fundamental Research 973 Program of China (No.2002CB312001), and by the Jiangsu Province Research Foundation (No.BK2004080).

E. Najm et al. (Eds.): FORTE 2006, LNCS 4229, pp. 388–403, 2006.

usually to describe the scenario-based requirements provided directly by the customers, while time Petri nets are used to model the workflow synthesized by the domain and technical experts. So it is necessary and important to keep the consistency between these two kinds of models for software quality assurance.

In this paper, we introduce a more expressive mechanism in MSCs to describe timing constraints, and give the solution to the problem of checking concurrent and real-time system designs modelled by time Petri nets for the scenario-based specifications expressed by MSCs, which require that if a scenario described by a given MSC occurs during the run of a time Petri net, the timing constraints enforced to the MSC must be satisfied.

The paper is organized as follows. In next section, we introduce MSCs and the related timing constraints, and use them to represent the scenario-based specifications. In Section 3, we review the definition and some basic properties of time Petri nets. Section 4 gives the solution to checking time Petri nets for the scenario-based specifications expressed by MSCs. The related works and some conclusions are given in the last section.



$$e_1 - e_{13} \leq -100, \quad e_6 - e_7 < 0, \quad e_{12} - e_{13} < 0$$

$$(e_{13} - e_1) - 2(e_{13} - e_{12}) \leq 0$$

Fig. 1. A bMSC describing the railroad crossing system

2 Message Sequence Charts with Timing Constraints

MSCs represent typical execution scenarios, providing examples of either normal or exceptional executions of the proposed system. The MSC standard as defined by ITU-T in Recommendation Z.120 [1] introduces two basic concepts: *basic MSCs* (bMSCs) and *High-Level MSCs* (hMSCs). A bMSC consists of a set of processes that run in parallel and exchange messages in a one-to-one, asynchronous fashion. A hMSC graphically combines references to bMSCs to describe parallel, sequence, iterating, and non-deterministic execution of the bMSCs. In this paper, we just use bMSCs to represent the scenario-based specifications, which are incomplete and usually specify the requirements provided directly by the customers. For example, a MSC is depicted in Figure 1, which describes a scenario about the well-known example of the railroad crossing system in [4,10]. This system operates a barrier at a railroad crossing, in which there are a railroad crossing monitor and a barrier controller for controlling the barrier. When the monitor detects that a train is arriving, it sends a message to the controller to lower the barrier. After the train leaves the crossing, the monitor sends a message to controller to raise the barrier.

The semantics of a MSC essentially consists of the sequences (of traces) of messages that are sent and received among the concurrent processes in the MSC. The order of communication events (i.e. message sending or receiving) in a trace is deduced from the visual partial order determined by the flow of control within each process in the MSC along with a causal dependency between the events of sending and receiving a message [1,6,7,9]. In accordance with [9], without losing generality, we assume that each MSC corresponds to a visual order for a pair of events e_1 and e_2 such that e_1 precedes e_2 in the following cases:

- **Causality:** A sending event e_1 and its corresponding receiving event e_2.
- **Controllability:** The event e_1 appears above the event e_2 on the same process line, and e_2 is a sending event. This order reflects the fact that a sending event can wait for other events to occur. On the other hand, we sometimes have less control on the order in which receiving events occur.
- **Fifo order:** The receiving event e_1 appears above the receiving event e_2 on the same process line, and the corresponding sending events e'_1 and e'_2 appear on a mutual process line where e'_1 is above e'_2.

For facilitating the specifications of real-time systems, the timers [1], interval delays [7,8], and timing marks [2] have been introduced to describe timing constraints in MSCs. All of these mechanisms are suitable to describe simple timing constraints which are only about the separation in time between two events. In this paper, we introduce more general and expressive timing constraints in MSCs. In a MSC, we use event names to represent the occurrence time of events. So, timing constraints can be described by boolean expressions on event names. Here we let any timing constraint be of the form

$$c_0(e_0 - e'_0) + c_1(e_1 - e'_1) + \ldots + c_n(e_n - e'_n) \sim c\,,$$

where $e_0, e'_0, e_1, e'_1 \ldots, e_n, e'_n$ are event names, $c, c_0, c_1, \ldots, c_n$ are real numbers, and $\sim \in \{\leq, <\}$. For example, in the MSC depicted in Figure 1, the boolean expression $e_1 - e_{13} \leq -100$ represents the separation in time between the sending events e_1 and e_{13} is not smaller than 100 time units. Furthermore, if we require that the separation in time between the sending event e_{13} and the sending event e_1 is not greater than two times the one between the sending event e_{13} and the receiving event e_{12}, we can describe the requirement by the timing constraint $(e_{13} - e_1) - 2(e_{13} - e_{12}) \leq 0$.

Compared to the timers, interval delays, and timing marks, the timing constraints we consider here can be used to describe more complex timing requirements in practical use. For the scenario of the railroad system depicted in Figure 1, we suppose that when a train has passed, a new train could come after at least 100 time units. Figure 1 depicts a specification for this system represented by a MSC in which we require that from the time one train is arriving to the time the next train is arriving, the barrier stay up for at least half of this period, which is represented by $(e_{13} - e_1) - 2(e_{13} - e_{12}) \leq 0$. Clearly, this timing constraint is about the relation between two separations in time between events (one is the separation in time between e_{13} and e_{12}, and the other is the separation in time between e_{13} and e_1), and the timers, interval delays, and timing marks can not be used to describe such a timing requirement since they are suitable to describe the simple timing constraints only about the separation in time between two events.

For checking the scenario-based specification expressed by MSCs, we formalize MSCs as follows.

Definition 1. A MSC is a tuple $D = (P, E, M, L, V, C)$ where

- P is a finite set of processes. E is a finite set of events corresponding to sending a message and receiving a message.
- M is a finite set of messages. Each message in M is of the form (e, g, e') where $e, e' \in E$ corresponds to sending and receiving the message respectively, and g is the message name which is a character string. For any message $(e, g, e') \in M$, we use $g!$ and $g?$ to represent the sending and the receiving for the message respectively if we just concern the message name, and let $\phi(e) = g!$ and $\phi(e') = g?$.
- $L : E \rightarrow P$ is a labelling function which maps each event $e \in E$ to a process $L(e) \in P$ which is the sender (receiver) while e corresponds to sending (receiving) a message.
- V is a finite set whose elements are a pair (e, e') where $e, e' \in E$ and e precedes e', which is corresponding to a visual order.
- C is a set of timing constraints on event names enforced on D. □

We use *event sequences* to represent the *traces* of MSCs which are corresponding to the untimed behavior of MSCs. Any event sequence is of the form $e_0\hat{}e_1\hat{}\ldots\hat{}e_m$, which represents that e_{i+1} takes place after e_i for any i $(0 \leq i \leq m-1)$.

Definition 2. Let $D = (P, E, M, L, V, C)$ be a MSC. An event sequence of the form $e_0\hat{}e_1\hat{}\ldots\hat{}e_m$ is a *trace* of D if and only if the following conditions hold:

- all events in E occur in the sequence, and each event occurs only once, i.e. $\{e_0, e_1, \ldots, e_m\} = E$ and $e_i \neq e_j$ for any i, j $(0 \leq i < j \leq m)$; and
- $e_1, e_2, \ldots, e_m$ satisfy the visual order defined by V, i.e. for any e_i and e_j, if $(e_i, e_j) \in V$, then $0 \leq i < j \leq m$. □

Corresponding to the sending or receiving for messages, we can transform the traces of a MSC into the *message trails* of the MSC.

Definition 3. Let $D = (P, E, M, L, V, C)$ be a MSC. For any trace of D of the form $e_0\hat{}e_1\hat{}\ldots\hat{}e_m$, replacing each e_i with $\phi(e_i)$ $(0 \le i \le m)$, we get a sequence $\phi(e_0)\hat{}\phi(e_1)\hat{}\ldots\hat{}\phi(e_m)$ of the sending or receiving for messages in M, which is a *message trail* of D. □

Notice that for a MSC D, all events in a trace of D are distinct, but there may be the same events in a message trail of D which are corresponding to the message sending or receiving. For example, the events e_1 and e_{13} are distinct in the MSC depicted in Figure 1, but $\phi(e_1) = \phi(e_{13}) =$Train_arriving!.

We use *timed event sequences* to represent the behavior of MSCs. Any timed event sequence is of the form $(e_0, \delta_0)\hat{}(e_1, \delta_1)\hat{}\ldots\hat{}(e_m, \delta_m)$ where e_i is an event and δ_i is a nonnegative real numbers for any i $(0 \le i \le m)$, which describes that e_0 takes place δ_0 time units after the system starts, then e_1 takes place δ_1 time units after e_0 takes place, so on and so forth, at last e_m takes place δ_m time units after e_{m-1} takes place.

Definition 4. A timed event sequence $\omega = (e_0, \delta_0)\hat{}(e_1, \delta_1)\hat{}\ldots\hat{}(e_m, \delta_m)$ is a behavior of a MSC $D = (P, E, M, L, V, C)$ if and only if $e_0\hat{}e_1\hat{}\ldots\hat{}e_m$ is a trace of D and $\delta_0, \delta_1, \ldots, \delta_m$ satisfy the timing constraints described by C, i.e. for any boolean expression $\sum_{i=0}^{n} c_i(f_i - f_i') \sim c$ in C, $c_0\lambda_0 + c_1\lambda_1 + \ldots + c_n\lambda_n \sim c$ where for each i $(0 \le i \le n)$, if $f_i = e_j$ and $f_i' = e_k$, then

$$\lambda_i = \begin{cases} \delta_{k+1} + \delta_{k+2} + \ldots + \delta_j & \text{if } j > k \\ -(\delta_{j+1} + \delta_{j+2} + \ldots + \delta_k) & \text{if } j < k \end{cases} .$$ □

3 Time Petri Nets

Time Petri nets [3] are classical Petri Nets where to each transition t a time interval $[a, b]$ is associated. The times a and b are relative to the moment at which t was last enabled. Assuming that t was enabled at time c, then t may fire only during the interval $[c + a, c + b]$ and must fire at the time $c + b$ at the latest, unless it is disabled before by the firing of another transition. Firing a transition takes no time. The time Petri nets considered in this paper are 1-safe.

Definition 5. Let $\mathbf{N}$ be the set of natural numbers. A time Petri net is a six-tuple, $N = (P, T, F, Eft, Lft, \mu_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*; $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions* $(P \cap T = \emptyset)$; $F \subset (P \times T) \cup (T \times P)$ is the *flow relation*; $\mu \subset P$ is the *initial marking* of the net.
- $Eft, Lft : T \to \mathbf{N}$ are functions for the *earliest* and *latest firing* times of transitions, satisfying that for any $t \in T$, $Eft(t) \le Lft(t) < \infty$.

A *marking* μ of N is any subset of P. For any transition t, ${}^\bullet t = \{p \in P | (p, t) \in F\}$ and $t^\bullet = \{p \in P | (t, p) \in F\}$ denote the *preset* and *postset* of t, respectively. A transition t is *enabled* in a marking μ if ${}^\bullet t \subseteq \mu$; otherwise, it is *disabled*. Let *enabled*(μ) be the set of transitions enabled in μ. □

Definition 6. Let $\mathbf{T}$ be the set of nonnegative real numbers. A *state* of a time Petri net $N = (P, T, F, Eft, Lft, \mu_0)$ is a pair $s = (\mu, c)$, where μ is a marking of N, and $c : \mathit{enabled}(\mu) \to \mathbf{T}$ is called the *clock function*. The *initial state* of N is $s_0 = (\mu_0, c_0)$ where $c_0(t) = 0$ for any $t \in \mathit{enabled}(\mu_0)$. □

For the firing of a transition to be possible at a certain time, four conditions must be satisfied.

Definition 7. A transition t may fire from state $s = (\mu, c)$ after delay $\delta \in \mathbf{T}$ if and only if (1) $t \in \mathit{enabled}(\mu)$, (2) $(\mu - {}^{\bullet}t) \cap t^{\bullet} = \emptyset$, (3) $Eft(t) \leq c(t) + \delta$, and (4) $\forall t' \in \mathit{enabled}(\mu) : c(t') + \delta \leq Lft(t')$. □

The first condition is the normal firing condition for Petri nets. The second condition requires *contact-freeness*. The third condition specifies that the transition may only fire if its clock has reached the Eft value of the transition. The last condition quantifies over all other enabled transitions, and makes sure that the delay δ doesn't cause any of the Lft bounds to be invalidated. The new state is then calculated as follows.

Definition 8. When transition t fires after delay δ from state $s = (\mu, c)$, the new state $s' = (\mu', c')$ is given as follows: $\mu' = (\mu - {}^{\bullet}t) \cup t^{\bullet}$, and for any $t' \in \mathit{enabled}(\mu')$, if $t' \neq t$ and $t' \in \mathit{enabled}(\mu)$, then $c'(t') = c(t') + \delta$ else $c'(t') = 0$. This is denoted by $s' = fire(s, (t, \delta))$. □

The new marking is calculated normally. For clocks we have two cases: if a transition remains enabled in the new marking its clock value is incremented with δ, while for newly enabled transition the clock value is 0. The behavior of a time Petri net is described in term of *runs*.

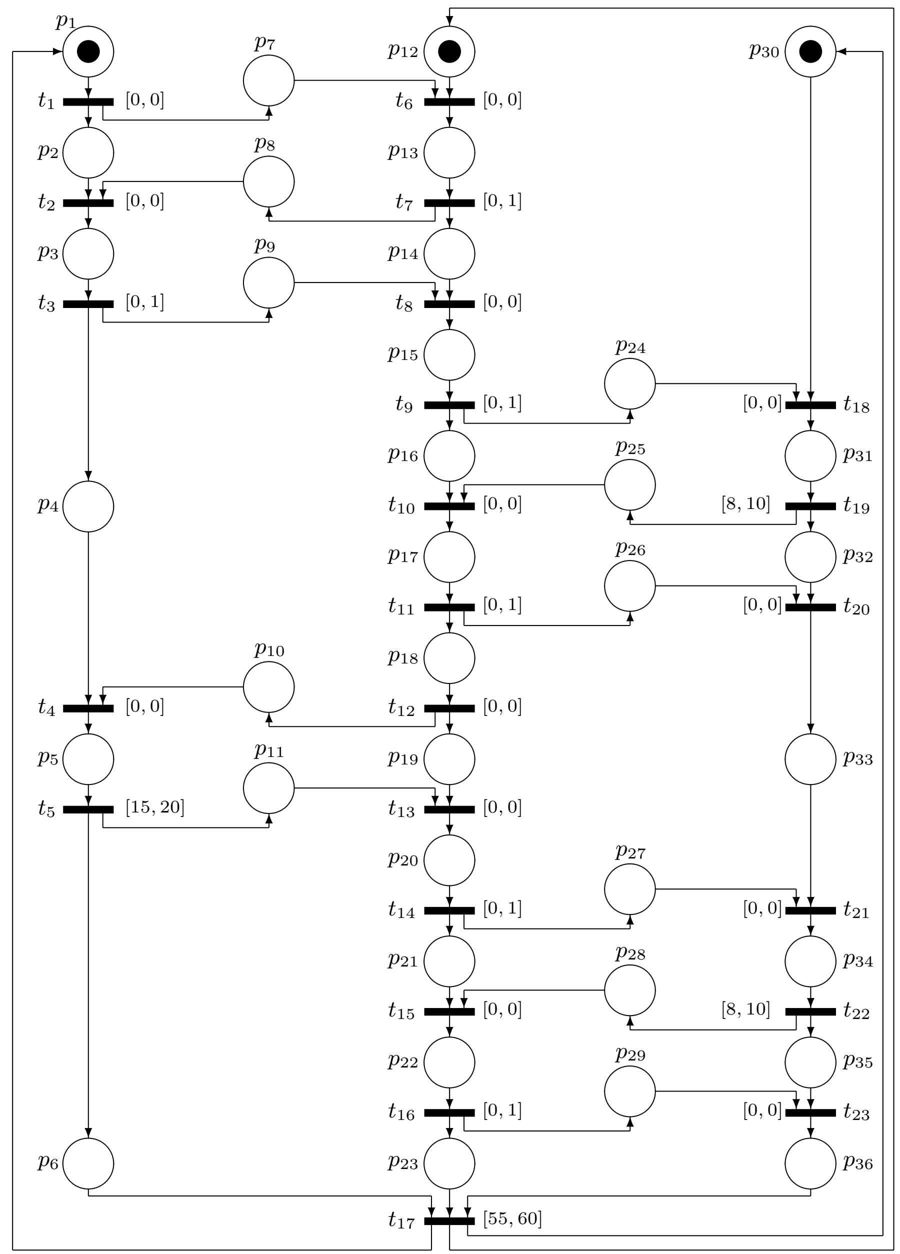
Definition 9. For any time Petri net, a *run*

$$\rho = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} \ldots$$

is a finite or infinite sequence of states, transitions, and delays such that s_0 is the initial state, and for every $i \geq 1$, s_i is obtained from s_{i-1} by firing a transition t_{i-1} after delay δ_{i-1} which satisfies that $s_i = fire(s_{i-1}, (t_{i-1}, \delta_{i-1}))$. □

As a tool used for modelling systems, time Petri nets are such that their transitions represent the potential events in the systems. Since in this paper we consider the problem of checking time Petri nets for the scenario-based specifications expressed by MSCs, for any time Petri net we consider in this paper, each transition t is labelled with an event denoted by $\varphi(t)$, which may be corresponding to a message sending or receiving in a MSC. That is, for a MSC $D = (P, E, M, L, V, C)$, for a transition t of a time Petri net, there may be a message $(e, g, e') \in M$ such that $\varphi(t) = g! = \phi(e)$ or $\varphi(t) = g? = \phi(e')$.

For example, for the railroad crossing system described in the above section, its design can be described by a time Petri net depicted in Figure 2. In the system, when the monitor detects that a train is arriving, it sends the message Train_arriving at once to the controller. The controller sends a message back for



$\varphi(t_1)$ =Train_arriving!	$\varphi(t_2)$ =Acknowledgement?	$\varphi(t_3)$ =Approaching!	$\varphi(t_4)$ =Crossing_secured?
$\varphi(t_5)$ =Train_passed!	$\varphi(t_6)$ =Train_arriving?	$\varphi(t_7)$ =Acknowledgement!	$\varphi(t_8)$ =Approaching?
$\varphi(t_9)$ =Low_barrier!	$\varphi(t_{10})$ =Barrier_down?	$\varphi(t_{11})$ =Power_off!	$\varphi(t_{12})$ =Crossing_secured!
$\varphi(t_{13})$ =Train_passed?	$\varphi(t_{14})$ =Raise_barrier!	$\varphi(t_{15})$ =Barrier_up?	$\varphi(t_{16})$ =Power_off!
$\varphi(t_{18})$ =Low_barrier?	$\varphi(t_{19})$ =Barrier_down!	$\varphi(t_{20})$ =Power_off!	$\varphi(t_{21})$ =Raise_barrier?
$\varphi(t_{22})$ =Barrier_up!	$\varphi(t_{23})$ =Power_off!		

Fig. 2. Time Petri net model for the railway crossing system

acknowledgement in one time units, and the monitor gives a reply in one time units. Once the controller receives the confirmed message Approaching, it sends the message Low_barrier to the barrier in one time unit. The barrier is put down in $[8, 10]$ time units after receiving the message Low_barrier, and the message Barrier_down is sent to the controller. Then in one time unit the controller sends the message Power_off to the Barrier, and the message Barrier_secured to the monitor. It takes $[15, 20]$ time units for the train to pass the crossing after the monitor receives the message Barrier_secured. Once the train passes the crossing, the monitor sends a message to the controller, and after receiving the message the controller takes one time unit to send the message Raise_barrier to the barrier. The barrier becomes up in $[8, 10]$ time units after receiving the message from the controller, and the message Barrier_up is sent to the controller. Once receiving the massage Barrier_up, the controller takes one time unit to send a message to the barrier for turning off the power. The barrier holds up in the coming $[55, 60]$ time units, and then another train is arriving.

4 Checking Time Petri Nets for the Scenario-Based Specifications Expressed by MSCs

In this section, we give the solution to checking of time Petri nets for the scenario-based specifications represented by MSCs.

4.1 Definition of the Satisfaction Problem

Given a MSC $D = (P, E, M, L, V, C)$, we can get a scenario-based specification for timing consistency, denoted by $\mathcal{S}_T(D)$. For a time Petri net N, $\mathcal{S}_T(D)$ requires that whenever a scenario described by D occurs in a run of N, the corresponding run segment must satisfy all the timing constraints in C. For example, Figure 1 depicts a timing consistency specification for the time Petri net in Figure 2, which requires that after a train has passed, a new train can come after at least 100 time units, and that from the time one train is arriving to the time the next train is arriving, the barrier stay up for at least half of this period.

The satisfaction problem of a time Petri net N for a scenario-based specification $\mathcal{S}_T(D)$ is defined formally as follows. Let $D = (P, E, M, L, V, C)$ and

$$\rho = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$$

be a run of N. For any subsequence ρ_1 of ρ which is of the form

$$\rho_1 = s_i \xrightarrow{(t_i,\delta_i)} s_{i+1} \xrightarrow{(t_{i+1},\delta_{i+1})} \ldots \xrightarrow{(t_{j-1},\delta_{j-1})} s_j \xrightarrow{(t_j,\delta_j)} s_{j+1} \quad (0 \le i < j < n+1)\,,$$

since each transition t_k is labelled with an event $\varphi(t_k)$ $(i \le k \le j)$, we get a sequence τ of events: $\tau = \varphi(t_i)\hat{}\varphi(t_{i+1})\hat{}\ldots\hat{}\varphi(t_j)$. By removing any $\varphi(t_k)$ $(i \le k \le j)$ from τ which is not corresponding to the sending or receiving for a message in M, we get an event sequence $\tau_1 = e_0\hat{}e_2\hat{}\ldots\hat{}e_m$ $(m \le j - i)$. If τ_1 is a message trail of D, $\varphi(t_i) = e_0$, and $\varphi(t_j) = e_m$, then we say that ρ_1 is an *image* of D in ρ. If ρ_1 is an image of D, then there is a trace $f_0\hat{}f_1\hat{}\ldots\hat{}f_m$ of D which is corresponding to τ_1, and we can give a function

$$\theta : \{f_0, f_1, \ldots, f_m\} \to \{t_i, t_{i+1}, \ldots, t_j\}$$

which map each f_k $(0 \leq k \leq m)$ in an incremental order to t_l $(i \leq l \leq j)$ such that $\phi(f_k) = \varphi(t_l)$, that is, $\theta(f_0) = t_i$, $\theta(f_m) = t_j$, and if $\theta(f_a) = t_p$ and $\theta(f_b) = t_q$ $(a < b)$, then $p < q$. We define that the image ρ_1 of D satisfies $\mathcal{S}_T(D)$ if $\delta_i, \delta_{i+1}, \ldots, \delta_j$ satisfy all the timing constraints in C, i.e. for any timing constraint $\sum_{k=0}^{n} c_k(g_k - g'_k) \sim c$ in C, $c_0\lambda_0 + c_1\lambda_1 + \ldots + c_n\lambda_n \sim c$ where for each k $(0 \leq k \leq n)$, if $\theta(g_k) = t_a$ and $\theta(g'_k) = t_b$ $(i \leq a, b \leq j)$, then

$$\lambda_k = \begin{cases} \delta_{b+1} + \delta_{b+2} + \ldots + \delta_a & \text{if } a > b \\ -(\delta_{a+1} + \delta_{a+2} + \ldots + \delta_b) & \text{if } a < b \end{cases} .$$

We define that the run ρ of N satisfies $\mathcal{S}_T(D)$ if any image of D in ρ satisfies $\mathcal{S}_T(D)$, and that N satisfies $\mathcal{S}_T(D)$ if any run of N satisfies $\mathcal{S}_T(D)$.

4.2 Integer Time Verification Approach

According to the above definition, for solving the satisfaction problem of a time Petri net N for a scenario-based specification $\mathcal{S}_T(D)$, we need to check all the runs of N. We know that for a time Petri net, its runs could be infinite and the number of its runs could be infinite. So we attempt to solve the problem based on a finite set of finite runs. In the following we present an integer time verification approach to solving the problem. A similar approach has been used by us to check time Petri nets for linear duration properties [17].

For a time Petri net N, a run ρ of N of the form

$$\rho = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n$$

is an *integral run* if all δ_is occurred in its combined steps are integers. It follows that any state $s = (\mu, c)$ occurring in an integral run satisfies $c(t)$ is an integer for any $t \in enabled(\mu)$, which is called *integral state.*

Theorem 1. A time Petri net N satisfies a scenario-based specification $\mathcal{S}_T(D)$ if and only if any integral run of N satisfies $\mathcal{S}_T(D)$. □

The proof of this theorem is presented in the appendix. According to the above theorem, when we check a time Petri net N for a scenario-based specification $\mathcal{S}_T(D)$, we only need to consider the integral runs of N.

Since according to Definition 5 the upper bounds of the time intervals associated to transitions are finite, the number of the integral states in a time Petri net is finite. Therefore, for a time Petri net $N = (P, T, F, Eft, Lft, \mu_0)$, we can construct a *reachability graph* $G = (V, E)$ as follows, where V is a set of nodes and E is a set of edges:

1. The initial state (μ_0, c_0) of N is in the set V, which is called *initial node*;
2. Let $s = (\mu, c)$ be in the set V, and κ is the minimal value of the set $\{Lft(t) \mid t \in enabled(\mu)\}$. Then for any transition $t \in enabled(\mu)$, for any integer $\delta \geq 0$ such that $Eft(t) \leq c(t) + \delta \leq \kappa$, $s' = fire(s, (t, \delta))$ is in V, and $s \xrightarrow{(t,\delta)} s'$ is in the set E.

For a time Petri net N, a *path* in its reachability graph $G = (V, E)$ is a sequence of states, transitions, and delays $s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n$ such that s_0 is the initial node, $s_i \in V$ for every i $(0 \leq i \leq n)$, and $s_i \xrightarrow{(t_i,\delta_i)} s_{i+1} \in E$ for every i $(0 \leq i < n)$. It follows that any integral run of N is a path in G, and any path in G is an integral run of N. So we can solve the problem of checking a time Petri net N for a scenario-based specification $\mathcal{S}_T(D)$ by checking if every path in the reachability graph G of N satisfies $\mathcal{S}_T(D)$.

4.3 Algorithm for Timing Consistency Checking

Since for a time Petri net whose reachability graph is G, a path in G could be infinite and the number of paths in G could be infinite, we need to solve the problem based on a finite set of finite paths in G as follows.

First, for a time Petri net N, we define *loops* in its reachability graph G. Let ϱ be a path in G of the form $\varrho = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$. If all s_i $(0 \leq i \leq n)$ are distinct and there are s_k $(0 \leq k < n)$ such that $s_k = s_{n+1}$, then we say that the subsequence

$$\varrho_1 = s_k \xrightarrow{(t_k,\delta_k)} s_{k+1} \xrightarrow{(t_{k+1},\delta_{k+1})} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_k$$

is a *loop* in G, and $\delta_k + \delta_{k+1} + \ldots + \delta_n$ is the *elapsed time* on ϱ_1, denoted by $\zeta(\varrho_1)$. For a given MSC $D = (P, E, M, L, V, C)$, if there is t_i $(k \leq i \leq n)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E)$, then we say that the loop ϱ_1 is *related to D*.

Then, for a node s in the reachability graph G of a time Petri net, for a MSC D, we define recursively the set $\Theta(s, D)$ of the loops which are not related to D as follows:

- any loop ϱ in G from s to itself which is not related to D is in $\Theta(s, D)$;
- for any loop in $\Theta(s, D)$ of the form $s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n$, any loop ϱ in G from s_i $(0 \leq i < n)$ to itself which is not related to D is in $\Theta(s, D)$.

Let N be a time Petri net with its reachability graph G. Now for a given scenario-based specification $\mathcal{S}_T(D)$ where $D = (P, E, M, L, V, C)$, we introduce the *violable points* in an image of D in a path in G. Let ϱ be a path in G, and ϱ_1 is an image of D in ϱ of the form

$$\varrho_1 = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{m-1},\delta_{m-1})} s_m \xrightarrow{(t_m,\delta_m)} s_{m+1} \,.$$

We have defined that ϱ_1 satisfies $\mathcal{S}_T(D)$ if $\delta_0, \delta_1, \ldots, \delta_m$ satisfy all the timing constraints in C, i.e. for any timing constraint $\sum_{k=0}^{n} c_k(g_k - g'_k) \sim c$ in C,

$c_0\lambda_0 + c_1\lambda_1 + \ldots + c_n\lambda_n \sim c$ where for each k $(0 \leq k \leq n)$, if $\theta(g_k) = t_a$ and $\theta(g'_k) = t_b$ $(0 \leq a, b \leq m)$, then

$$\lambda_k = \begin{cases} \delta_{b+1} + \delta_{b+2} + \ldots + \delta_a & \text{if } a > b \\ -(\delta_{a+1} + \delta_{a+2} + \ldots + \delta_b) & \text{if } a < b \end{cases} .$$

We say that s_i $(0 \leq i \leq m)$ is a *violable point* in ϱ_1 if the following condition holds:

- $\varphi(t_i) \neq \phi(e)$ $(e \in E)$,
- there is a loop $\varrho' \in \Theta(s_i, D)$ whose elapsed time is greater than zero $(\zeta(\varrho') > 0)$, and
- δ_i occurs in λ_k $(0 \leq k \leq n)$ and $c_k\lambda_k > 0$ (in this case, $c_k\lambda_k$ becomes larger while δ_i becomes larger).

Last, for a time Petri net N with its reachability graph G, we define the finite set $\Delta(N, \mathcal{S}_T)$ of the finite paths in G which we need to check for a given scenario-based specification $\mathcal{S}_T(D)$ where $D = (P, E, M, L, V, C)$. $\Delta(N, \mathcal{S}_T)$ is the set of the paths in G which are of the form

$$s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{k-1},\delta_{k-1})} s_k \xrightarrow{(t_k,\delta_k)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$$

where all s_i $(0 \leq i \leq k)$ are distinct, $s_k \xrightarrow{(t_k,\delta_k)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$ is an image of D, and for any s_i and s_j $(k < i < j < n)$, if there is not any t_l $(i \leq l \leq j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E)$ then $s_i \neq s_j$.

Theorem 2. A time Petri net N satisfies a scenario-based specification $\mathcal{S}_T(D)$ if and only if any path ϱ in $\Delta(N, \mathcal{S}_T(D))$ satisfies $\mathcal{S}_T(D)$ and no violable point occurs in the image of D in ϱ. □

The proof of this theorem is presented in the appendix. For a timing Petri net N, for a scenario-based specification $\mathcal{S}_T(D)$, a path ϱ in the reachability graph of N is a *prefix* for $\Delta(N, \mathcal{S}_T(D))$ if it may be extended into a path which is in $\Delta(N, \mathcal{S}_T(D))$, i.e. there could be a sequence ϱ_1 of states, transitions, and delays such that $\varrho \xrightarrow{(t,\delta)} \varrho_1$ is in $\Delta(N, \mathcal{S}_T(D))$. Based on Theorem 2, we can develop an algorithm to check if a time Petri net N satisfies a scenario-based specification $\mathcal{S}_T(D)$ (cf. Figure 3). The algorithm traverses the reachability graph G of N in a depth first manner starting from the initial node. The path in G that we have so far traversed is stored in the list variable *currentpath*. The boolean variable *is_no_scenario* indicates if there is a scenario described by D occurring in N $(\Delta(N, \mathcal{S}_T(D)) \neq \emptyset)$. The set variable *loopset* is used to store all loops in G. The algorithm consists of two steps which are implemented by depth first search. In the first search, we traverse G for getting all the loops in G, which are used for checking if no violable point occurs in the image of D in any path in $\Delta(N, \mathcal{S}_T(D)$. Then we start a new depth first search to find out all the paths in $\Delta(N, \mathcal{S}_T(D))$ and to check them for $\mathcal{S}_T(D)$. For each new node we discover, we first check if it is such that the path corresponding to *currentpath* is in $\Delta(N, \mathcal{S}_T(D))$.

```
is_no_scenario :=true;
  currentpath := $\langle(\mu_0, c_0)\rangle$; loopset := $\emptyset$;
  repeat
    node := the last node of currentpath;
    if node has no new successive node
    then delete the last node of currentpath
    else
      begin
        node := a new successive node of node;
        if node has occurred in currentpath (we find out a loop $\varrho$)
        then put $\varrho$ into loopset
        else append node to currentpath;
      end
  until currentpath = $\langle\rangle$;

  currentpath := $\langle(\mu_0, c_0)\rangle$;
  repeat
    node := the last node of currentpath;
    if node has no new successive node
    then delete the last node of currentpath
    else
      begin
        node := a new successive node of node;
        if node is such that the path $\varrho$ corresponding to currentpath
           is in $\Delta(N, \mathcal{S}_T(D))$
        then
          begin
            check if $\varrho$ satisfies $\mathcal{S}_T(D)$;
            if no, return false;
            is_no_scenario :=false;
            check if no violable point occurs in the image of $D$ in $\varrho$;
            if no, return false;
          end
        if node is such that currentpath is corresponding to
           a prefix for $\Delta(N, \mathcal{S}_T(D))$
        then append node to currentpath;
      end
  until currentpath = $\langle\rangle$;
if is_no_scenario then return "No scenario of $D$ occurs"
else return true.
```

Fig. 3. Algorithm for timing consistency checking

If yes, then we first check the path for $\mathcal{S}_T(D)$ and assign *is_no_scenario* with **false**. Then we check if no violable point occurs in the image of D in the path. If the new node is such that *currentpath* is not corresponding to a prefix for $\Delta(N, \mathcal{S}_T(D))$, then the algorithm backtracks, otherwise the algorithm adds the new node to *currentpath*. The algorithm terminates because there is only a finite number of the paths in $\Delta(N, \mathcal{S}_T(D))$. Since the algorithm is based on depth first search method, its complexity is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_T(D))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_T(D))$.

The algorithm presented above has been implemented in a tool prototype. On a PentiumM/1.50GHz/512MB PC, the tool runs comfortably for several case studies including the railroad crossing system. The solution we give is based on investigating only the integer time state spaces of time Petri nets. But even for the integer time state spaces of time Petri nets, their sizes are often much large in the problems of practical interest so that more optimization and abstraction techniques are needed.

5 Related Work and Conclusion

To our knowledge, there has been few literature on consistency checking of time Petri nets for scenario-based specifications expressed by MSCs. A work closed to our own is described in [14] to verify whether the timed state machines in a UML model interact according to time-annotated UML collaboration diagrams, in which timed state machines are compiled into timed automata [16] and a collaboration diagram with time intervals is translated into an observer automaton, and the model checker UPPAAL [15] for timed automata is called for the verification, which is based on checking the automata inclusion. Compared to that work, the timing constraints considered in our work are more general and expressive than the timer, time intervals, and timing marks adopted in the existing works, which can be used to describe the relation among multiple separations in time between events. We know that for a clock constraint in a timed automaton, its corresponding timing constraint is about just the separation in time between two events. For describing timing constraints about the relation among multiple separations in time between events, we need to compare multiple clocks in a timed automaton, which will result in that the corresponding model checking problems are undecidable [16]. Thus, the scenario-based specifications expressed by MSCs considered in this paper cannot be verified by transferring to timed automata.

There have been a number of work on checking time Petri nets for the temporal logic based properties [11-13]. Compared to those works, on one hand, the problems considered in those works are to check if the behavior of time Petri nets satisfy the given temporal order of events specified by the temporal logics, while the problem we concern is to check if the behavior of time Petri nets satisfy not only the the given temporal order of events, but also the given timing constraints. On the other hand, the scenario-based specifications considered in this paper are a class of the original artifacts in software development processes, and often come directly from the requirements provided by the customers and domain experts. We know that it is not easy to use formal verification techniques directly in industry because the modelling languages in the verification tools are too formal and theoretical to master easily. For industry, it is much more acceptable to adopt MSCs as a specification language instead of the temporal logics in formal verification tools.

In this paper, since the specifications we concern usually come from the scenario-based requirements provided directly by the customers, which is incomplete, we just use bMSCs to describe the scenario-based specifications. For describing the more complete scenario-based specifications, we need to consider hMSC, which is one of our next works.

References

1. ITU-T. Recommendation Z.120. ITU - Telecommunication Standardization Sector, Geneva, Switzerland, May 1996.
2. J. Rumbaugh and I. Jacobson and G. Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
3. B. Berthomieu and M. Diaz. Modelling and verification of time dependent systems using time Petri nets. In *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.
4. Olaf Kluge. Modelling a Railway Crossing with Message Sequence Chatrs and Petri Nets. In H.Ehrig et al.(Eds.): *Petri Technology for Communication-Based Systems - Advance in Petri Nets*, LNCS 2472, Springer, 2003, pp.197-218.
5. van der Aalst, W.M.P. Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. In *Systems Analysis - Modelling - Simulation*, Vol.34, No.3, pages 335-367. 1999.
6. Uwe Rueppel, Udo F. Meissner, and Steffen Greb. A Petri Net based Method for Distributed Process Modelling in Structural Engineering. In *Proc. International Conference on Computing in Civil and Building Engineering*, 2004.
7. R. Alur, G.J. Holzmann, D. Peled. An Analyzer for Message Sequence Charts. In *Software-Concepts and Tools* (1996) 17: 70-77.
8. Hanene Ben-Abdallah and Stefan Leue. Timing Constraints in Message Sequence Chart Specifications. In *Proceedings of FORTE/PSTV'97*, Chapman & Hall, 1997.
9. Doron A. Peled. *Software Reliability Methods.* Springer, 2001.
10. Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Comparing Different Approaches for Specifying and Verifying Real-Time Systems. In *Proc.* 10_{th} *IEEE Workshop on Real-Time Operating Syatems abd Software.* New York, 1993. pp.122-129.
11. Andrea Bobbio, Andras Horvath. Model Checking Time Petri Nets using NuSMV. In *Proccedings of the Fifth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS2001)*, 2001.
12. Furfaro A. and Nigro L. Model Checking Time Petri Nets: A Translation Approach based on Uppaal and a Case Study. In *Proceedings of IASTED International Conference on Software Engineering(SE 2005)*, Innsbruck, Austria, Acta Press, 2005.
13. Tomohiro Yoneda, Hikaru Ryuba. CTL Model Checking of Time Petri Nets using Geometric Regions. In *IEICE Trans. INF. & SYST.*, Vol.E99-D, No.3, 1998, pp.1-11.
14. Alexander Knapp, Stephan Merz, and Christopher Rauh. Modelchecking Timed UML State Machines and Collaborations. In W. Damm and E.-R. Olderog (Eds.): *FTRTFT2002*, LNCS 2469, Springer, 2002, pp.395-414.
15. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. In *International Journal of Software Tools for Technology Transfer*, 1(1-2): 134-152, 1997.
16. R. Alur and D. David. A theory of timed automata. In *Theoretical Computer Science* 126 (1994). pp.183-235.
17. Xuandong Li and Johan Lilius. Checking Time Petri Nets for Linear Duration Properties. In Peter Bucbolz, manuel Silva (Eds.), Petri Nets and Performance Models, IEEE Computer Society Press, 1999. pp.218-226.

A Proofs of Theorems

Theorem 1. A time Petri net N satisfies a scenario-based specification $\mathcal{S}_T(D)$ if and only if any integral run of N satisfies $\mathcal{S}_T(D)$.

Proof. Let $D = (P, E, M, L, V, C)$, and ρ be a run of N of the form

$$s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{k-1},\delta_{k-1})} s_k \xrightarrow{(t_k,\delta_k)} \ldots \xrightarrow{(t_{m-1},\delta_{m-1})} s_m \xrightarrow{(t_m,\delta_m)} s_{m+1}$$

where $s_k \xrightarrow{(t_k,\delta_k)} \ldots \xrightarrow{(t_{m-1},\delta_{m-1})} s_m \xrightarrow{(t_m,\delta_m)} s_{m+1}$ is an image of D. For a timing constraint $\xi \in C$ of the from $\sum_{i=0}^{n} c_i(g_i - g'_i) \sim c$, let

$$\beta(\rho, \xi) = c_0\lambda_0 + c_1\lambda_1 + \ldots + c_n\lambda_n$$

where for each i $(0 \le i \le n)$, if $\theta(g_i) = t_a$ and $\theta(g'_i) = t_b$ $(k \le a, b \le m)$, then

$$\lambda_i = \begin{cases} \delta_{b+1} + \delta_{b+2} + \ldots + \delta_a & \text{if } a > b \\ -(\delta_{a+1} + \delta_{a+2} + \ldots + \delta_b) & \text{if } a < b \end{cases} .$$

The theorem follows immediately from the following claim: there is a run ρ' of N of the form

$$s'_0 \xrightarrow{(t_0,\delta'_0)} s'_1 \xrightarrow{(t_1,\delta'_1)} \ldots \xrightarrow{(t_{k-1},\delta'_{k-1})} s'_k \xrightarrow{(t_k,\delta'_k)} \ldots \xrightarrow{(t_{m-1},\delta'_{m-1})} s'_m \xrightarrow{(t_m,\delta'_m)} s'_{m+1}$$

such that it is an integral run of N and that $\beta(\rho, \xi) \le \beta(\rho', \xi)$. This claim can be proved as follows.

Let $\alpha_i = \delta_0 + \delta_1 + \ldots + \delta_i$ $(0 \le i \le m)$. It is clear that if each α_i $(0 \le i \le m)$ is an integer, then ρ is an integral run. Let $frac(\rho)$ be the set containing all fractions of α_i $(0 \le i \le m)$, 0, and 1, i.e.

$$frac(\rho) = \left\{ \gamma_i \;\middle|\; \begin{array}{l} 0 \le \gamma_i \le 1, 0 \le i \le m, \\ \text{and } \alpha_i - \gamma_i \text{ is an integer} \end{array} \right\} \cup \{0, 1\} .$$

Let $rank(\rho)$ be the number of the elements in $frac(\rho)$. Notice that if $rank(\rho) = 2$, then ρ is an integral run. In the following, we show that if $rank(\rho) > 2$, we can construct a run ρ_1 of the form

$$s''_0 \xrightarrow{(t_0,\delta''_0)} s''_1 \xrightarrow{(t_1,\delta''_1)} \ldots \xrightarrow{(t_{k-1},\delta''_{k-1})} s''_k \xrightarrow{(t_k,\delta''_k)} \ldots \xrightarrow{(t_{m-1},\delta''_{m-1})} s''_m \xrightarrow{(t_m,\delta''_m)} s''_{m+1}$$

such that $rank(\rho_1) = rank(\rho) - 1$ and $\beta(\rho_1, \xi) \ge \beta(\rho, \xi)$. By applying this step repeatedly, we can get a run ρ' which is an integral run of satisfying $rank(\rho') = 2$ and $\beta(\rho', \xi) \ge \beta(\rho, \xi)$ so that the claim is proved. Let

$$frac(\rho) = \{\gamma_0, \gamma_1, \ldots, \gamma_l\} \; (\gamma_0 = 0, \gamma_l = 1, \gamma_i < \gamma_{i+1} \; (0 \le i \le l - 1)) ,$$

and $index(\gamma_1) = \{i \mid 0 \le i \le m \text{ and } \delta_i - \gamma_1 \text{ is an integer}\}$. Let α'_i and α''_i defined as

$$\alpha'_i = \begin{cases} \alpha_i - \gamma_1 & \text{if } i \in index(\gamma_1) \\ \alpha_i & \text{if } i \notin index(\gamma_1) \end{cases} , \quad \alpha''_i = \begin{cases} \alpha_i - \gamma_1 + \gamma_2 & \text{if } i \in index(\gamma_1) \\ \alpha_i & \text{if } i \notin index(\gamma_1) \end{cases} .$$

Let $\delta_0^I = \alpha'_0$ and $\delta_0^\Pi = \alpha''_0$. For each i $(1 \leq i \leq m)$, let $\delta_i^I = \alpha'_i - \alpha'_{i-1}$ and $\delta_i^\Pi = \alpha''_i - \alpha''_{i-1}$. Let

$$\rho'_1 = s_0^I \xrightarrow{(t_0,\delta_0^I)} s_1^I \xrightarrow{(t_1,\delta_1^I)} \dots \xrightarrow{(t_{k-1},\delta_{k-1}^I)} s_k^I \xrightarrow{(t_k,\delta_k^I)} \dots \xrightarrow{(t_{m-1},\delta_{m-1}^I)} s_m^I \xrightarrow{(t_m,\delta_m^I)} s_{m+1}^I$$

$$\rho''_1 = s_0^\Pi \xrightarrow{(t_0,\delta_0^\Pi)} s_1^\Pi \xrightarrow{(t_1,\delta_1^\Pi)} \dots \xrightarrow{(t_{k-1},\delta_{k-1}^\Pi)} s_k^\Pi \xrightarrow{(t_k,\delta_k^\Pi)} \dots \xrightarrow{(t_{m-1},\delta_{m-1}^\Pi)} s_m^\Pi \xrightarrow{(t_m,\delta_m^\Pi)} s_{m+1}^\Pi$$

It follows that $rank(\rho'_1) = rank(\rho) - 1$ and $rank(\rho''_1) = rank(\rho) - 1$, and either $\beta(\rho'_1, \xi) \geq \beta(\rho, \xi)$ or $\beta(\rho''_1, \xi) \geq \beta(\rho, \xi)$. Suppose $N = (P, T, F, Eft, Lft, \mu_0)$. Since $Eft(t)$ and $Lft(t)$ are a natural number for any $t \in T$, ρ'_1 and ρ''_1 are a run of N. Let $\rho_1 = \rho'_1$ when $\beta(\rho'_1, \xi) \geq \beta(\rho, \xi)$, and $\rho_1 = \rho''_1$ when $\beta(\rho''_1, \xi) \geq \beta(\rho, \xi)$. By applying the above step repeatedly, the claim can be proved. □

Theorem 2. A time Petri net N satisfies a scenario-based specification $\mathcal{S}_T(D)$ if and only if any path ϱ in $\Delta(N, \mathcal{S}_T(D))$ satisfies $\mathcal{S}_T(D)$ and no violable point occurs in the image of D in ϱ.

Proof. It is clear that the half of the claim holds: if N satisfies $\mathcal{S}_T(D)$, then any path ϱ in $\Delta(N, \mathcal{S}_T(D))$ satisfies $\mathcal{S}_T(D)$ and no violable point occurs in the image of D in ϱ. The reason is that for a path ϱ in $\Delta(N, \mathcal{S}_T(D))$, if there is a violable point s in the image of D in ρ, then we can construct a path ϱ' from ρ whose image of D does not satisfy $\mathcal{S}_T(D)$ by repeating a loop $\varrho_1 \in \Theta(s, D)$ $(\zeta(\varrho_1) > 0)$ many times such that $\zeta(\varrho_1)$ becomes large enough to violate the related timing constraint enforced to D. The other half of claim can be proved as follows. Let $D = (P, E, M, L, V, C)$. Suppose that there is a path ϱ of the form

$$s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \dots \xrightarrow{(t_{k-1},\delta_{k-1})} s_k \xrightarrow{(t_k,\delta_k)} \dots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$$

where $s_k \xrightarrow{(t_k,\delta_k)} \dots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} s_{n+1}$ is an image of D which does not satisfy $\mathcal{S}_T(D)$. Since

- for any s_i and s_j $(0 \leq i < j < k)$ such that $s_i = s_j$, by removing the subsequence $s_i \xrightarrow{t_i,\delta_i} s_{i+1} \xrightarrow{t_{i+1},\delta_{i+1}} \dots \xrightarrow{t_{j-2},\delta_{j-2}} s_{j-1} \xrightarrow{t_{j-1},\delta_{j-1}}$ from ϱ we can get a run of N, and
- for any s_i and s_j $(k < i < j < n)$ such that $s_i = s_j$ and that there is not any $t_l (i \leq l \leq j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E)$, by removing the subsequence $s_i \xrightarrow{t_i,\delta_i} s_{i+1} \xrightarrow{t_{i+1},\delta_{i+1}} \dots \xrightarrow{t_{j-2}} s_{j-1} \xrightarrow{t_{j-1},\delta_{j-1}}$ from ϱ we can get a run of N,

we can construct a run ϱ' from ϱ which is in $\Delta(N, \mathcal{S}_T(D))$. Since there is no violable point in the image of D in ϱ', the sequences removing from ϱ in the process of constructing ϱ' do not related to any timing constraint in C. It follows that the image of D in ϱ' does not satisfy $\mathcal{S}_T(D)$, which results in a contradiction. Thus, the claim holds. □