Consistency Checking of Concurrent Models for Scenario-Based Specifications*

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

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. Scenario-based specifications such as message sequence charts (MSCs) offer an intuitive and visual way of describing design requirements. As one powerful formalism, Petri nets can model concurrency constraints in a natural way, and are often used in modelling system specifications and designs. Since there are gaps between MSC models and Petri net models, keeping consistency between these two kinds of models is important for the success of software development. In this paper, we use Petri nets to model concurrent systems, and consider the problem of checking Petri nets for scenario-based specifications expressed by message sequence charts. We develop the algorithms to solve the following two verification problems: the existential consistency checking problem, which means that a scenario described by a given MSC must happen during a Petri net runs, or any forbidden scenario described by a given MSC never happens during a Petri net run; and the mandatory consistency checking problem, which means that if a reference scenario described by the given MSCs occurs during a Petri net run, it must adhere to a scenario described by the other given MSC.

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. They are playing an increasingly important role in specification and design of systems. Such specifications focus on message exchanges among communicating entities in real-time and distributed systems.

* Supported by the National Natural Science Foundation of China (No.60425204, No.60233020, and No.60273036), the National Grand Fundamental Research 973 Program of China (No.2002CB312001), and by Jiangsu Province Research Foundation (No.BK2004080, No.BK2003408).

A. Prinz, R. Reed, and J. Reed (Eds.): SDL 2005, LNCS 3530, pp. 298–312, 2005.

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 specifications of the communication behavior of real-time systems, in particular telecommunication switching systems.

Petri Nets [2] is a formal and graphically appealing language, which is appropriate for modelling systems with concurrency and resource sharing. There are plenty of applications of Petri Nets in modelling system specifications and designs.

We often use MSCs and Petri nets together in a software project [3-5]. Usually, MSCs and Petri nets are used in different software development steps. Even when used in the same step, such as requirements analysis, MSCs are usually used to describe the requirements provided directly by the customers, while Petri nets are used to model the workflow synthesized by the domain and technical experts. Since there are gaps between MSC models and Petri net models, keeping consistency between these two kinds of models is important for the success of software development.

In this paper, we consider the problem of checking concurrent system designs modelled by Petri nets for scenario-based specification expressed by MSCs. We develop the algorithms to solve the following two verification problems: the existential consistency checking problem, which means that a scenario described by a given MSC must happen during a Petri net run, or any forbidden scenario described by a given MSC never happens during a Petri net run; and the mandatory consistency checking problem, which means that if a reference scenario described by the given MSCs occurs during a Petri net run, it must adhere to a scenario described by the other given MSC.

The paper is organized as follows. In the next section, we introduce MSCs, and use them to represent scenario-based specifications. In section 3, we review the definition and some basic properties of Petri nets. The solutions are given in Section 4 and 5 respectively to the existential and mandatory consistency checking of Petri nets for scenario-based specifications expressed by MSCs, and the detailed proofs of the theorems are omitted in these two sections because of space consideration. The related works and some conclusions are given in the last section.

2 Message Sequence Charts and Scenario-Based Specifications

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 instances that run in parallel and exchange messages in a one-to-one, asynchronous fashion. Instances usually represent process instances, and so are called processes in this paper. An hMSC graphically combines references to bM-

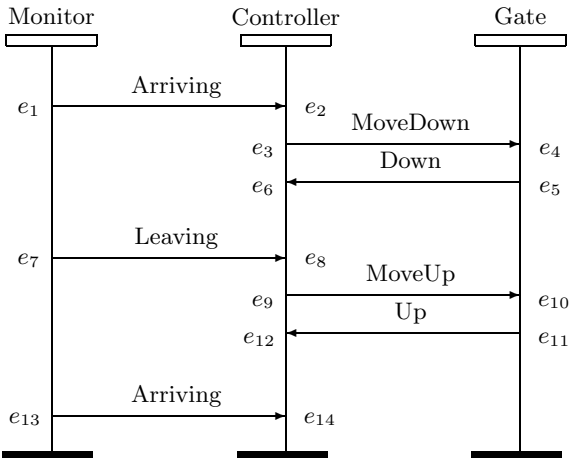

Fig. 1. An MSC describing the railroad crossing system

SCs to describe parallel, sequence, iterating, and non-deterministic execution of the bMSCs. In this paper we just consider bMSCs which are used to represent scenario-based specifications. For example, an MSC is depicted in fig. 1, which describes a scenario about the well-known example of the railroad crossing system in [6]. This system operates a gate at a railroad crossing, in which there are a railroad crossing monitor and a gate controller for controlling the gate. When the monitor detects that a train is arriving, it sends a message to the controller to move down the gate. After the train leaves the crossing, the monitor sends a message to controller to open the gate.

The semantics of an MSC essentially consists of sequences (of traces) of messages that are sent and received among the concurrent processes in the MSC. The order of communication events (message sent or received) in a trace is deduced from the visual partial order determined by the flow from top to bottom within each process in the MSC along with a causal dependency between the events of sending and receiving a message [1, 7, 8, 9]. In accordance with [7], 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 send event e_1 and its corresponding receive event e_2.
- **Controlability:** The event e_1 appears above the event e_2 on the same process line, and e_2 is a send event. This order reflects the fact that a send event can wait for other events to occur. On the other hand, we sometimes have less control on the order in which receive events occur.
- **Fifo order:** The receive event e_1 appears above the receive event e_2 on the same process line, and the corresponding send events e_1' and e_2' appear on a mutual process line where e_1' is above e_2'.

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

Definition 1. An MSC is a five-tuple $D = (P, E, M, L, V)$ 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. For any message $g \in M$, let $g!$ and $g?$ represent the send and the receive for g respectively. For any $e \in E$, it is corresponding to a send or receive for a message g, denoted by $\phi(e) = g!$ or $\phi(e) = g?$.
- $L : E \to P$ is labelling function which maps each event $e \in E$ to a process $L(e) \in P$.
- V is a finite set whose elements are of the form (e, e') where e and e' are in E and $e' \neq e$, which represents a visual order displayed in D.

□

We use *event sequences* to represent the *traces* of MSCs which are corresponding to the 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 $D = (P, E, M, L, V)$ be an 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 $\{e_0, e_1, \ldots, e_m\} = E$ 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 for any e_i and e_j, if $(e_i, e_j) \in V$, then $0 \leq i < j \leq m$. □

Corresponding to the sends or receives for messages, we can transform the traces of an MSC into the *message trails* of the MSC.

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

Notice that for an 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 sends or receives. For example, the events e_1 and e_{13} are distinct in the MSC depicted in fig. 1, but $\phi(e_1) = \phi(e_{13}) = Arriving!$.

3 Petri Nets

The Petri nets we consider in this paper are classical 1-safe systems.

Definition 4. A Petri net is a four-tuple, $N = (P, T, F, \mu_0)$, 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*;
- $\mu_0 \subset P$ is the *initial marking* of the net.

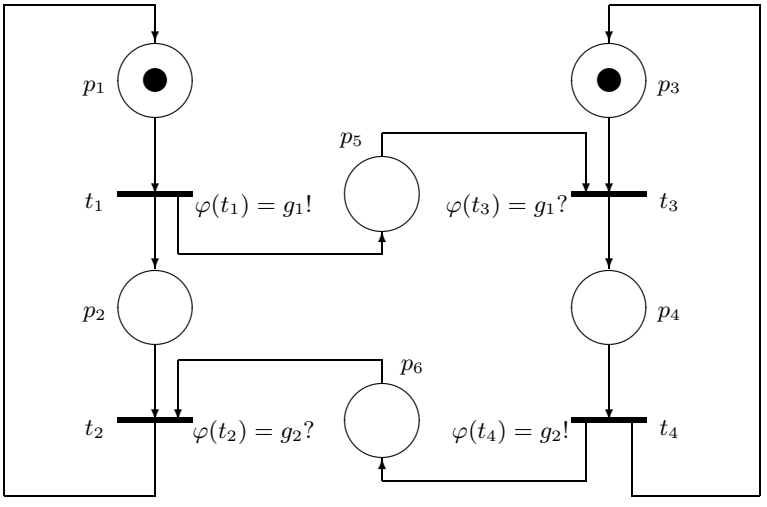

Fig. 2. A Petri net

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. □

As a tool used for modelling systems, the transitions of Petri nets represent the potential events in the systems. Since in this paper we consider the problem of checking Petri nets for scenario-based specifications expressed by MSCs, for any Petri net we consider in this paper, each transition t is labelled with an event which may be corresponding to a message send or receive in a MSC, which is denoted by $\varphi(t)$. That is, for an MSC $D = (P, E, M, L, V)$, for a transition t of a Petri net, there may be a message $g \in M$ and an event $e \in E$ such that $\varphi(t) = g! = \phi(e)$ or $\varphi(t) = g? = \phi(e)$. For example, a Petri net is depicted in fig. 2.

For the firing of a transition to be possible, two conditions must be satisfied.

Definition 5. A transition t may fire from marking μ if and only if the following two conditions hold: (1) $t \in$ *enabled*(μ), and (2) $(\mu - {}^\bullet t) \cap t^\bullet = \emptyset$. □

The first condition is the normal firing condition for Petri nets. The second condition requires *contact-freeness*. The new state is then calculated as follows.

Definition 6. When transition t fires from marking μ, the new marking μ' is given as follows: $\mu' = (\mu - {}^\bullet t) \cup t^\bullet$. □

Note that since we assume contact-freeness, a self-loop will not be enabled. The behaviour of a Petri net is described in term of *runs*.

Definition 7. A *run* of a Petri net is a finite or infinite sequence of markings and transitions

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \ldots$$

such that μ_0 is the initial marking of the net, $t_i \in enabled(\mu_i)$ for any i $(i \geq 0)$, and that $\mu_i = (\mu_{i-1} -^{\bullet} t_{i-1}) \cup t_{i-1}^{\bullet}$ for any i $(i \geq 1)$. □

4 Existential Consistency Checking

In this section, we consider the existential consistency checking of Petri nets for scenario-based specifications represented by MSCs. The existential consistency checking problem is to check if a scenario described by a given MSC must happen during a Petri net run, or that any forbidden scenario described by a given MSC never happens during a Petri net run, which is depicted in fig. 3.

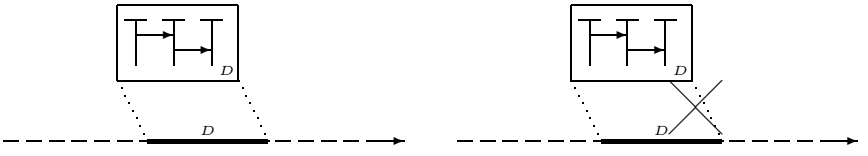

Fig. 3. Existential Consistency Checking

Now we define formally the existential consistency checking problem. Let $D = (P, E, M, L, V)$ be an MSC, N be a Petri net, and σ be a run of N of the form $\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$. For any subsequence σ_1 in σ of the form $\sigma_1 = \mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1}$ $(0 \leq i < j \leq n)$, since each transition t_k is labelled with an event $\varphi(t_k)$ $(i \leq k \leq j)$, we get a sequence τ of events: $\tau = \varphi(t_i)^\frown\varphi(t_{i+1})^\frown \ldots ^\frown\varphi(t_j)$. By removing any $\varphi(t_k)$ $(i \leq k \leq j)$ from τ which is not corresponding to the send or receive for a message in M, we get an event sequence $\tau_1 = e_0{}^\frown e_1{}^\frown \ldots {}^\frown e_m$ $(m \leq j - i + 1)$. If τ_1 is a message trail of D, then we say that a scenario described by D *occurs* in the run σ, and that σ_1 is an *image* of D in σ. If τ_1 is a message trail of D, $\varphi(t_i) = e_0$, and $\varphi(t_j) = e_m$, then we say that σ_1 is an *exact image* of D. For a Petri net N, for an MSC D, the existential consistency checking problem is to check if there is a run of N where a scenario described by D occurs.

We know that for a Petri net N, there could be infinite runs, and the number of the runs could be infinite. In the following we show how to solve the problem based on the investigation of a finite set of finite runs.

For any Petri net N, for any MSC $D = (P, E, M, L, V)$, let $\Delta(N, D)$ be the set of the runs of N which are of the form

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where

- all μ_i $(0 \leq i \leq m)$ are distinct;
- $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an exact image of D; and
- for any μ_i and μ_j $(m < i < j < n)$, if there is not any $t_k (i \leq k \leq j)$ such that $\varphi(t_k) = \phi(e)$ $(e \in E)$ then $\mu_i \neq \mu_j$.

It is clear that the number of the runs in $\Delta(N, D)$ is finite, and that each run in $\Delta(N, D)$ is finite.

Theorem 1. Let N be a Petri net, and D be an MSC. Then there is a run of N where a scenario described by D occurs if and only if $\Delta(N, D) \neq \emptyset$. □

```
currentpath := ⟨μ0⟩;
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 run corresponding to currentpath
              is in Δ(N, D)
           then return true;
           if node is such that the run corresponding to currentpath
              is a prefix for Δ(N, D)
           then append node to currentpath;
        end
until currentpath = ⟨⟩;
return false.
```

Fig. 4. Algorithm for existential consistency checking

For a Petri net N, for an MSC D, a run σ of N is a *prefix* for $\Delta(N, D)$ if it may be extended into a run which is in $\Delta(N, D)$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, D)$. Based on the above theorem, we can develop an algorithm to check the existential consistency of a Petri net $N = (P, T, F, \mu_0)$ for an MSC D (see fig. 4). The algorithm traverses the state space of N in a depth first manner starting from the initial node μ_0. The path in the state space that we have so far traversed is stored in the list variable *currentpath*. For each new marking that we discover, we first check whether it is such that the run corresponding to *currentpath* is in $\Delta(N, D)$. If yes, then it means that a scenario described by D must happen during N runs, and we are done. Then we check if the new marking that we discover is such that the run corresponding to *currentpath* is a prefix for $\Delta(N, D)$. If yes, then we add the new marking to the current path and start the search from it, otherwise

we backtrack. If no run is discovered during the search which is in $\Delta(N, D)$, then it means that any forbidden scenario described by D never happens during N runs. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, D)$ and to the size of the longest prefix for $\Delta(N, D)$.

5 Mandatory Consistency Checking

In this section, we consider the mandatory consistency checking of Petri nets for scenario-based specifications represented by MSCs. The mandatory consistency requires that if a reference scenario described by the given MSCs occurs during a Petri net run, it must adhere to a scenario described by the other given MSC. We consider the following three kinds of the mandatory consistency which are depicted in fig. 5:

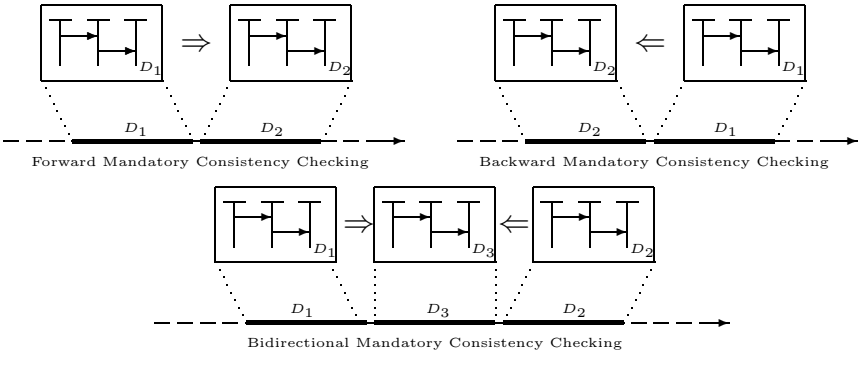

Fig. 5. Mandatory Consistency Checking

- **forward mandatory consistency:** if a reference scenario described by a given MSC D_1 occurs during a Petri net run, then a scenario described by the other given MSC D_2 must follow immediately;
- **backward mandatory consistency:** if a reference scenario described by a given MSC D_1 occurs during a Petri net run, then it must follow immediately from a scenario described by the other given MSC D_2; and
- **bidirectional mandatory consistency:** if a reference scenario described by a given MSC D_1 occurs during a Petri net run and a reference scenario described by another given MSC D_2 follows, then in between these two scenarios, a scenario described by the third given MSC D_3 must occur exactly.

5.1 Forward Mandatory Consistency Checking

For the forward mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_F(D_1, D_2)$, consists of two given MSCs D_1 and D_2, which requires

that if a scenario described by D_1 occurs in a run of a Petri net, then a scenario described by D_2 follows immediately (see fig. 5).

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ is defined formally as follows. Let $D_2 = (P_2, E_2, M_2, L_2, V_2)$. N satisfies $\mathcal{S}_F(D_1, D_2)$ if any run σ of N of the form

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$$

satisfies the following condition:

– if there is a subsequence σ_1 in σ of the form

$$\sigma_1 = \mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \ (0 \le i < j \le n)$$

which is an exact image of D_1, then for any subsequence σ_2 in σ of the form

$$\sigma_2 = \mu_{j+1} \xrightarrow{t_{j+1}} \mu_{j+2} \xrightarrow{t_{j+2}} \cdots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1} \ (j < k \le n)$$

where the number of μ_l $(j < l \le k)$ satisfying $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ is $|E_2|$, it is an image of D_2.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. For any Petri net N, for any scenario-based specification $\mathcal{S}_F(D_1, D_2)$ where $D_1 = (P_1, E_1, M_1, L_1, V_1)$ and $D_2 = (P_2, E_2, M_2, L_2, V_2)$, let $\Delta(N, \mathcal{S}_F(D_1, D_2))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where

– all μ_i $(0 \le i \le k)$ are distinct;
– $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1}$ is an exact image of D_1;
– for any μ_i and μ_j $(k < i < j < m)$, if there is not any $t_l (i \le l \le j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_1)$ then $\mu_i \ne \mu_j$;
– the number of μ_i $(m < i \le n)$ satisfying $\varphi(t_i) = \phi(e)$ $(e \in E_2)$ is $|E_2|$;
– there are $e \in E_2$ such that $\varphi(t_n) = \phi(e)$; and
– for any μ_i and μ_j $(m < i < j < n)$, if there is not any $t_l (i \le l \le j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ then $\mu_i \ne \mu_j$.

For any $\sigma \in \Delta(N, \mathcal{S}_F(D_1, D_2))$ of the above form, we call the subsequence $\mu_{m+1} \xrightarrow{t_{m+1}} \mu_{m+2} \xrightarrow{t_{m+2}} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ by the *last segment* of σ.

Theorem 2. A Petri net N satisfies a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$, its last segment is an image of D_2. □

For a Petri net N, for a scenario-based specification $\mathcal{S}_F(D_1, D_2)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_F(D_1, D_2))$ if it may be extended into a run which is in

```
currentpath := ⟨μ0⟩;
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 run corresponding to currentpath
              is in Δ(N, S_F(D1, D2))
           then
             begin
               check if the run corresponding to currentpath is such that
               its last segment is an image of D2;
               if no then return false;
             end;
           if node is such that the run corresponding to currentpath
              is a prefix for Δ(N, S_F(D1, D2))
           then append node to currentpath;
        end
until currentpath = ⟨⟩;
return true.
```

Fig. 6. Algorithm for forward mandatory consistency checking

$\Delta(N, \mathcal{S}_F(D_1, D_2))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma\hat{}\sigma_1$ is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$. Based on Theorem 2, we can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ (see fig. 6). The algorithm traverses the state space of N in a depth first manner starting from the initial node μ_0. The path in the state space that we have so far traversed is stored in the list variable *currentpath*. For each new marking that we discover, we first check whether it is such that the run corresponding to *currentpath* is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$. If so, we check if the run corresponding to *currentpath* is such that its last segment is an image of D_2. If no, then it means that N does not satisfy $\mathcal{S}_F(D_1, D_2)$, and we are done. Then we check if the new marking that we discover is such that the run corresponding to *currentpath* is a prefix for $\Delta(N, \mathcal{S}_F(D_1, D_2))$. If yes, then we add the new marking to the current path and start the search from it, otherwise we backtrack. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_F(D_1, D_2))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_F(D_1, D_2))$.

5.2 Backward Mandatory Consistency Checking

For the backward mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_B(D_1, D_2)$, consists of two given MSCs D_1 and D_2, which requires that if a scenario described by D_1 occurs in a run of a Petri net, then it must follow immediately from a scenario described by D_2 (see fig. 5).

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ is defined formally as follows. N satisfies $\mathcal{S}_B(D_1, D_2)$ if any run σ of N of the form $\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ satisfies the following condition:

– if there is a subsequence σ_1 in σ of the form

$$\sigma_1 = \mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \ (0 \leq i < j \leq n)$$

which is an exact image of D_1, then there is a subsequence σ_2 in σ of the form

$$\sigma_2 = \mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \cdots \xrightarrow{t_{i-2}} \mu_{i-1} \xrightarrow{t_{i-1}} \mu_i \ (0 \leq k < i)$$

which is an image of D_2.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. For any Petri net N, for any scenario-based specification $\mathcal{S}_B(D_1, D_2)$ where $D_1 = (P_1, E_1, M_1, L_1, V_1)$ and $D_2 = (P_2, E_2, M_2, L_2, V_2)$, let $\Delta(N, \mathcal{S}_B(D_1, D_2))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where

– $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an exact image of D_1;
– for any μ_i and μ_j $(m < i < j < n)$, if there is not any $t_l (i \leq l \leq j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_1)$ then $\mu_i \neq \mu_j$;
– for any μ_i and μ_j $(0 \leq i < j < m)$, if there is not any $t_l (i \leq l \leq j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ then $\mu_i \neq \mu_j$; and
– if there is k $(0 \leq k < m)$ such that the number of μ_l $(k \leq l < m)$ satisfying $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ is $|E_2|$, then all μ_i $(0 \leq i \leq k)$ are distinct.

For any $\sigma \in \Delta(N, \mathcal{S}_B(D_1, D_2))$ of the above form, we call the subsequences of the form $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{m-1}} \mu_m$ $(0 \leq i < m)$ by the *front segments* of σ.

Theorem 3. A Petri net N satisfies a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ if and only if for any run σ of N which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, there is a front segment of σ which is an image of D_2. □

For a Petri net N, for a scenario-based specification $\mathcal{S}_B(D_1, D_2)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_B(D_1, D_2))$ if it may be extended into a run which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$. Based on Theorem 3, we can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ (see fig. 7). The structure of the algorithm is the same as the one of the algorithm depicted in fig. 6. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_B(D_1, D_2))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_B(D_1, D_2))$.

currentpath := $\langle \mu_0 \rangle$;
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 run corresponding to *currentpath*
      is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$
    then
      begin
        check if the run corresponding to *currentpath* is such that
        a front segment is an image of D_2;
        if no **then return false**;
      end;
    if *node* is such that the run corresponding to *currentpath*
      is a prefix for $\Delta(N, \mathcal{S}_B(D_1, D_2))$
    then append *node* to *currentpath*;
  end
until *currentpath* = $\langle\rangle$;
return true.

Fig. 7. Algorithm for backward mandatory consistency checking

5.3 Bidirectional Mandatory Consistency Checking

For the bidirectional mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_D(D_1, D_2, D_3)$, consists of three given MSCs D_1, D_2, and D_3, which requires that if a scenario described by D_1 occurs in a run of a Petri net and a scenario described by D_2 follows, then the run segment between these two scenarios must be exactly corresponding to a scenario described by D_3 (see fig. 5).

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ is defined formally as follows. N satisfies $\mathcal{S}_D(D_1, D_2, D_3)$ if for any run σ of N of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \cdots \xrightarrow{t_{l-1}} \mu_l \xrightarrow{t_l} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \cdots \xrightarrow{t_{n-1}} \mu_n$$

where all μ_i $(l \leq i \leq m)$ are distinct, $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \cdots \xrightarrow{t_{l-1}} \mu_l$ is an exact image of D_1, and $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_{n-1}} \mu_n$ is an exact image of D_2, the following condition holds:

- if there is no subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} \mu_j$ $(l \leq i < j \leq m)$ which is an image of D_1 or D_2, then $\mu_l \xrightarrow{t_l} \mu_{l+1} \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_{m-1}} \mu_m$ is an image of D_3.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. For a Petri net N, for a scenario-based specification

currentpath := $\langle \mu_0 \rangle$;
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 run corresponding to *currentpath*
      is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$
    then
      begin
        check if the run corresponding to *currentpath* is such that
        its middle segment is an image of D_3;
        if no **then return false**;
      end;
    if *node* is such that the run corresponding to *currentpath*
      is a prefix for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$
    then append *node* to *currentpath*;
  end
until *currentpath* = $\langle\rangle$;
return true.

Fig. 8. Algorithm for bidirectional mandatory consistency checking

$\mathcal{S}_D(D_1, D_2, D_3)$ where $D_1 = (P_1, E_1, M_1, L_1, V_1)$ and $D_2 = (P_2, E_2, M_2, L_2, V_2)$, let $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \cdots \xrightarrow{t_{l-1}} \mu_l \xrightarrow{t_l} \cdots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \cdots \xrightarrow{t_{n-1}} \mu_n$$

where

- all μ_i $(0 \leq i \leq k)$ are distinct;
- all μ_i $(l \leq i \leq m)$ are distinct;
- $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \cdots \xrightarrow{t_{l-1}} \mu_l$ is an exact image of D_1;
- for any μ_i and μ_j $(k < i < j < l)$, if there is not any $t_a (i \leq a \leq j)$ such that $\varphi(t_a) = \phi(e)$ $(e \in E_1)$ then $\mu_i \neq \mu_j$;
- $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_{n-1}} \mu_n$ is an exact image of D_2;
- for any μ_i and μ_j $(m < i < j < n)$, if there is not any $t_a (i \leq a \leq j)$ such that $\varphi(t_a) = \phi(e)$ $(e \in E_2)$ then $\mu_i \neq \mu_j$; and
- there is no subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} \mu_j$ $(l \leq i < j \leq m)$ which is an image of D_1 or D_2.

For any $\sigma \in \Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ of the above form, we call the subsequence of the form $\mu_l \xrightarrow{t_l} \mu_{l+1} \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_{m-2}} \mu_{m-1} \xrightarrow{t_{m-1}} \mu_m$ by the *middle segment* of σ.

Theorem 4. Let N be a Petri net. N satisfies a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, its middle segment is an image of D_3. □

For a Petri net N, for a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ if it may be extended into a run which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$. Based on Theorem 4, we can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ (see fig. 8). The structure of the algorithm is the same as the one of the algorithm depicted in fig. 6. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ and to the size of the longest run in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$.

6 Related Work and Conclusion

In this paper, we solve the consistency checking problems of concurrent system designs modelled by Petri nets for scenario-based specifications expressed by MSCs. The algorithms we present can be used to check if a scenario described by a given MSC must happen during a Petri net run, that any forbidden scenario described by a given MSC never happens during a Petri net run, and if a Petri net satisfies a mandatory consistency specification expressed by MSCs which requires that if a reference scenario described by the given MSCs occurs during the Petri net run, it must adhere to a scenario described by the other given MSC.

To our knowledge, there has been a lack of publication on consistency checking of Petri nets for scenario-based specifications expressed by MSCs. There has been work on checking the state-transition graphs for the properties expressed by temporal logics (CTL, LTL)[10]. It is well known that the state-transition graphs and the Petri nets considered in this paper can be interchangeable, and even there exists an automatic translation of UML statecharts and sequence diagrams [11] into Generalized Stochastic Petri Nets [12], which means that theoretically the problems considered in this paper can be solved by converting to the corresponding formalisms. However, on the one hand, converting from one formalism to the other often leads to a significant enlargement of the state space. On the other hand, the specifications expressed by MSCs are much more acceptable for the industry than the ones expressed by temporal logic. Work close to our own is described in [13] where a tool, HUGO, is designed to check if the interactions expressed by an UML collaboration diagram can indeed be realized by a set of UML state machines in which state machines are compiled into SPIN [14] input model and collaboration diagrams are translated into sets of *Büchi* automata, and SPIN is called for the verification. Compared to that work, the specifications expressed by MSCs considered in this paper are more expressive, and our approach leads to direct and efficient implementation. Live sequence charts [15, 16] is a more expressive formalism to describe the mandatory consistency specifications considered in this paper, but it is less popular in the industry.

The algorithms presented in this paper have been implemented in a tool prototype. The tool is implemented in Java, and its graphical interface allows the users to construct, edit, and analyze MSCs and Petri nets interactively. The tool interface can also read .mdl files of UML sequence diagrams in Rational Rose.

Based on the work in this paper, we are solving the timing consistency checking problem of time Petri nets [17] for scenario-based specifications expressed by MSCs with timing constraints, which requires that if a scenario described by a given MSC occurs during a time Petri net run, the timing constraints enforced to the MSC must be satisfied.

References

1. ITU-T. Recommendation Z.120. ITU - Telecommunication Standardization Sector, Geneva, Switzerland, May 1996.
2. J.L. Peterson. *Petri Nets Theory and the Modeling of Systems*. Prentice-Hall, N.J., 1981.
3. Olaf Kluge. Modelling a Railway Crossing with Message Sequence Charts 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.
4. 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.
5. 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.
6. 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 Systems and Software*. New York, 1993. pp.122-129.
7. Doron A. Peled. *Software Reliability Methods*. Springer, 2001.
8. Rajeev Alur, Gerard J. Holzmann, Doron Peled. An Analyzer for Message Sequence Charts. In *Software-Concepts and Tools* (1996) 17: 70-77.
9. P.B. Ladkin and S. Leue. Interpreting Message Flow Graphs. In *Formal Aspects of Computing*, 7(5):473-509, 1995.
10. Edmund M. Clarke, Jr. Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.
11. J. Rumbaugh and I. Jacobson and G. Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
12. Simona Bernardi, Susanna Donatelli, and José Merseguer. From UML sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the third international workshop on Software and performance*, ACM Press, 2002, pp.35-45.
13. Timm Schafer, Alexander Knapp, and Stephan Merz. Model Checking UML State Machines and Collaborations. In *Electronic Notes in Theoretical Computer Science* 55, No.3, 2001.
14. G.J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*, Addison-Wesley, 2003.
15. Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. In *Formal Methods in System Design*, 19(1):45-80, 2001.
16. Yves Bontemps and Patrick Heymans. Turning High-Level Live Sequence Charts into Automata. In *Proceedings of Workshop: Scenarios and State-Machines*, 2002.
17. 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.