**Technical Report No. NJU-SEG-2012-IJ-001**

# Timing analysis of MSC specifications with asynchronous concatenation

Minxue Pan, Xuandong Li

Postprint Version. Originally Published in:
Software Tools Technology Transfer 2012
Int J Softw Tools Technol Transfer (2012) 14:639–651

Int J Softw Tools Technol Transfer (2012) 14:639–651
DOI 10.1007/s10009-012-0239-9

MTM

# Timing analysis of MSC specifications with asynchronous concatenation

Minxue Pan · Xuandong Li

Published online: 14 June 2012

**Abstract** Message Sequence Chart (MSC) is a graphical and textual language for describing the interactions between system components, and MSC specifications (MSSs) are a combination of a set of basic MSCs (bMSCs) and a High-level MSC that describes potentially iterating and branching system behavior by specifying the compositions of basic MSCs, which offer an intuitive and visual way of specifying design requirements. With concurrent, timing, and asynchronous properties, MSSs are amenable to errors, and their analysis is important and difficult. This paper deals with timing analysis of MSC specifications with asynchronous concatenation. For an MSC specification, we require that for any loop, its first node be flexible in execution time and its any associated external timing constraint be enforced on the entire loop. Such an MSC specification is called a *flexible loop-closed MSC specification* (FLMSS). We show that for FLMSSs, the *reachability analysis* and *bounded delay analysis* problems can be solved efficiently by linear programming. The solutions have been implemented into our tool TASS and evaluated by experiments.

**Keywords** Message Sequence Charts · Asynchronous concatenation · Timing consistency analysis

M. Pan · X. Li (✉)
State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing, Jiangsu 210093,
People's Republic of China
e-mail: lxd@nju.edu.cn

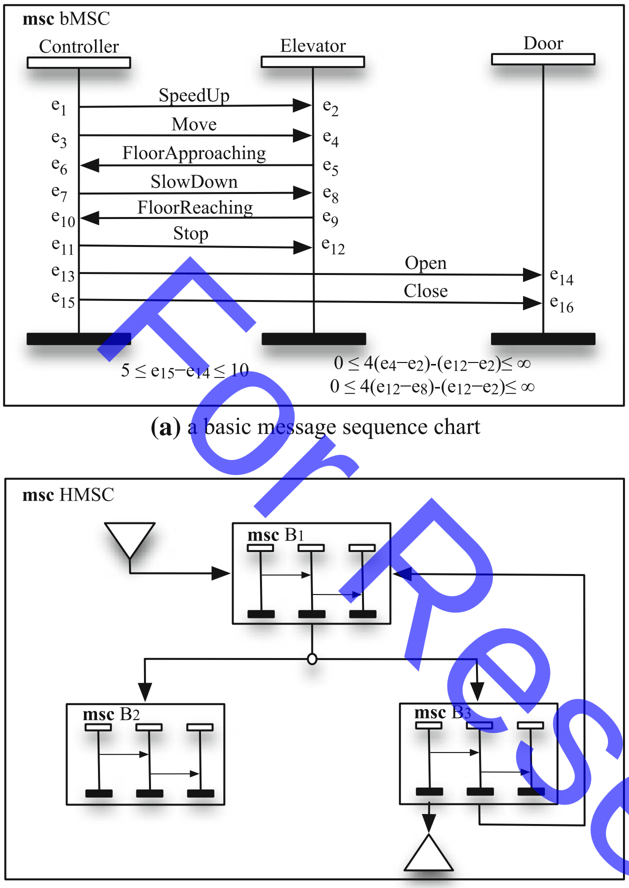
M. Pan
e-mail: panmx@seg.nju.edu.cn

M. Pan · X. Li
Department of Computer Science and Technology, Nanjing University,
Nanjing, Jiangsu 210093, People's Republic of China

## 1 Introduction

Message Sequence Chart (MSC) [1] is a graphical and textual language for describing the interactions between system components. MSC specifications (MSSs) are a combination of a set of basic MSCs (bMSCs) and a High-level MSC (HMSC). In an MSC specification, as illustrated in Fig. 1, each bMSC focuses on the temporal order of message flows and describes exactly one scenario without alternatives or loops, and the HMSC describes sequential, iterating and non-deterministic executions of scenarios depicted by bMSCs, which focuses on the overview of more complicated system behavior [1]. MSC specifications can describe systems in a hierarchical fashion and therefore can make complicated systems more comprehensible, and offer an intuitive and visual way of specifying design requirements. With concurrent, timing, and asynchronous properties, MSC specifications are amenable to errors, and their analysis is important and difficult. Since MSC specifications may be used at early stages of design, any error revealed during their analysis has a high payoff. In this paper, we are focused on timing analysis of MSC specifications.

For an MSC specification, its behavioral semantics depends on the interpretation of bMSC concatenation. For two bMSC $B_1$ and $B_2$, the *synchronous concatenation* of $B_1$ and $B_2$ requires that any event in $B_2$ happens after all the events in $B_1$, while the *asynchronous concatenation* has no such restriction. While the synchronous concatenation is easier to design and analyze, the asynchronous concatenation is more natural to model practical systems. For timing analysis of MSC specifications, most existing work is conducted on synchronous concatenation. The problems on asynchronous concatenation are more difficult. Even for the untimed MSC specifications, some analysis problems are undecidable [2].



**(a)** a basic message sequence chart

**(b)** a High-level message sequence chart

**Fig. 1** MSC specifications

In this paper, we consider timing analysis of MSC specifications with asynchronous concatenation, and focus on two problems: the *reachability analysis* and the *bounded delay analysis*. Reachability analysis is to check if a bMSC scenario is reachable in the behavior of an MSC specification, which is useful to assure the feasibility of the specification. Bounded delay analysis is to check if all behavior of an MSC specification satisfies that the time distance between two given events is within a given time interval, which is important for real-time and distributed systems since many safety properties involve time bounds, and many scheduling methods require information on time bounds, too. For MSC specifications with asynchronous concatenation, these two timing analysis problems are difficult, and to our knowledge there is no literature on them. In this paper, for giving efficient solutions we focus on a class of MSC specifications. For an MSC specification, we require that for any loop, its first node be flexible in execution time and its any associated external timing constraint be enforced on the entire loop. Such an MSC specification is called a *flexible loop-closed MSC specification* (FLMSS). For FLMSSs, we first reduce the problems on single finite behavior into linear programming problems, and then investigate the finite behavior one by one, which forms the efficient solutions for these two problems. The solutions have been implemented into our tool TASS and evaluated by experiments.

The paper is organized as follows. In the next section, we define MSC specifications formally and introduce FLMSSs. Section 3 gives the linear programming based solutions to reachability analysis and bounded delay analysis problems. Section 4 conducts a case study and evaluates the experiment results. The last section discusses the related work and draws the conclusion.

## 2 MSC specifications

The ITU Recommendation Z.120 advocates the use of structured design: to model a system, simple scenarios can be described by bMSCs; while more complete specifications can be formed by means of High-level MSCs which combines bMSCs [1].

### 2.1 Basic MSCs and timing constraints

A bMSC describes the message flow between system instances. In a bMSC, the vertical lines in the chart correspond to instances, and messages exchanged between those instances are represented by arrows. The sending and receiving of messages are corresponding to events, respectively. Figure 1a depicts a bMSC example.

For specifying real-time systems, timing constraints are introduced into MSC specifications, which specify the relations of time distances between events. In real-time systems, various functions and mechanisms can become the causes of the time distances, such as the network delay of transmitting messages or the CPU time of processing messages. It is common to abstract away these causes from the timing constraints to achieve the succinctness of the specifications. The timing constraints in ITU Recommendation Z.120 such as time points and time intervals [1], and the mechanisms in previous literature such as delays intervals [3,4] and timing marks [5–8] are just suitable to describe simple timing constraints which are only related to one time distance between a pair of events. For the elevator example [9] depicted in Fig. 1, we can use time intervals to describe simple timing constraints such as the elevator door should be kept open for 5–10 time units. However, in practical problems we often need to describe more complex timing constraints which are about the relation among multiple time distances between events. Therefore, we employ more expressive timing constraints in this paper. We use event names to represent event occurrence time, and linear inequalities on event names to represent the timing constraints. A timing constraint is of the form

$$a \le c_0(e_0 - e'_0) + c_1(e_1 - e'_1) + \cdots + c_n(e_n - e'_n) \le b$$

where $e_i$ and $e'_i$ $(0 \le i \le n)$ are event names which represent the occurrence time of $e_i$ and $e'_i$, $a, b$ and $c_0, c_1, \ldots, c_n$ are real numbers ($b$ may be $\infty$). For the elevator example, to avoid the discomfort caused by sudden acceleration, we require the time that the elevator speeds up or slows down should take more than one-fourth of the whole time that the elevator moves, which forms two timing constraints $0 \le 4(e_4 - e_2) - (e_{12} - e_2) \le \infty$ and $0 \le 4(e_{12} - e_8) - (e_{12} - e_2) \le \infty$. It is clear that using the existing mechanisms in MSCs we cannot describe such timing constraints.

The semantics of a bMSC essentially consists of the sequences (traces) of the message sending and receiving events. The order of 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 instance in the bMSCs along with a causal dependency between the events of sending and receiving a message. In accordance with [10, 11], without losing generality, we assume that for a pair of events $e$ and $e'$ in a bMSC, $e$ precedes $e'$ (denoted by $e \prec e'$) in the following cases:

- **Causality:** A sending event $e$ and its corresponding receiving event $e'$.
- **Controllability:** The event $e$ appears above the event $e'$ on the same instance axis, and $e'$ is a sending event.
- **Fifo order:** The receiving event $e$ appears above the receiving event $e'$ on the same instance axis, and the corresponding sending events $e_1$ and $e'_1$ appear on a mutual instance axis where $e_1$ is above $e'_1$.

In accordance with [11], we formally define bMSCs as follows.

**Definition 1** A basic MSC $B$ is a tuple $B = (I, E, M, L, V, C)$ where

- $I$ is a finite set of instances.
- $E$ is a finite set of events corresponding to sending a message and receiving a message. There are two special events $\epsilon$ and $\varpi$ in $E$ which represent the start and end of $B$, respectively.
- $M$ is a finite set of messages whose elements are a pair $(e, e')$ where $e, e' \in E$ are corresponding to the sending and the receiving for a message, respectively.
- $L : E \to I$ is a labeling function which maps each event $e \in E$ to an instance $L(e) \in I$ which is the sender (receiver) while $e$ corresponds to sending (receiving) a message.
- $V$ is a finite set whose elements are a pair $(e, e')$ $(e, e' \in E)$ such that $e \prec e'$.
- $C$ is a finite set of timing constraints. □

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

**Definition 2** Let $B = (I, E, M, L, V, C)$ be a bMSC. An event sequence $e_0 \to e_1 \to \cdots \to e_m$ is a *trace* of $B$ if and only if the following conditions hold:

- $e_0 = \epsilon$ and $e_m = \varpi$.
- $e_0, e_1, \ldots, e_m$ is a permutation of the events in $E$.
- $e_0, e_1, \ldots, e_m$ satisfies the visual order defined by $V$, i.e. for any $e_i$ and $e_j$, if $(e_i, e_j) \in V$, then $0 \le i < j \le m$.

□

We use *timed event sequences* to represent the behavior of bMSCs. A *timed event sequence* is of the form $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_m, t_m)$ where $e_i$ is an event and $t_i$ is a nonnegative real numbers for any $i$ $(0 \le i \le m)$. According to Definition 2, $e_0 = \epsilon$, so let $t_0$ be 0. The timed event sequence describes that $e_1$ takes place $t_1$ time units after $e_0$ takes place, then $e_2$ takes place $t_2$ time units after $e_1$ takes place, so on and so forth, at last $e_m = \varpi$ takes place $t_m$ time units after $e_{m-1}$ takes place. It follows that for any $i$ $(0 \le i \le m)$, the occurrence time of $e_i$ is $\sum_{j=0}^{i} t_j$.

**Definition 3** Let $B = (I, E, M, L, V, C)$ be a bMSC. A timed event sequence $\sigma = (e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_m, t_m)$ is a behavior of $B$ if and only if the following conditions hold:

- $e_0 \to e_1 \to \cdots \to e_m$ is a trace of $B$.
- $t_0, t_1, \ldots, t_m$ satisfy the timing constraints in $C$, i.e. for any timing constraint $a \le \sum_{i=0}^{n} c_i(f_i - f'_i) \le b$ in $C$, $a \le c_0\delta_0 + c_1\delta_1 + \cdots + c_n\delta_n \le b$ where for each $i$ $(0 \le i \le n)$, if $f_i = e_j$ and $f'_i = e_k$, then

$$\delta_i = \begin{cases} t_{k+1} + t_{k+2} + \cdots + t_j & \text{if } j > k \\ -(t_{j+1} + t_{j+2} + \cdots + t_k) & \text{if } j < k \end{cases}$$

Let $\mathcal{L}(B)$ denote the set of the timed event sequences representing the behavior of $B$. □

## 2.2 Definition of MSC specifications

While a bMSC describes a simple scenario, an HMSC can describe multiple scenarios and complete system specifications. An HMSC provides a means to graphically define how a set of bMSCs can be combined to describe potentially iterating and branching system behavior, which forms an MSC specification.

**Definition 4** An MSC specification $S = (U, N, succ, ref, T)$ where

- $U$ is a finite set of bMSCs satisfying that for any $B = (I, E, M, L, V, C)$ and $B' = (I', E', M', L', V', C')$ in $U$, if $B \neq B'$, then $E \cap E' = \emptyset$.
- $N = \{\top\} \cup IM \cup \{\bot\}$ is a finite set of nodes partitioned into the three sets: the singleton-set of *start* node, the set of *intermediate* nodes, and the singleton-set of *end* node, respectively.
- $succ \subseteq N \times N$ is the relation which reflects the connectivity of the nodes in $N$ (it is required that any node in $N$ be reachable from the start node).
- $ref : IM \mapsto U$ is a function that maps each intermediate node to a bMSC in $U$.
- $T$ is a finite set of timing constraints of the form $a \le e - e' \le b$ where $e$ and $e'$ occur in different bMSCs in $U$ and $0 \le a \le b$ ($b$ may be $\infty$), which are used to describe the timing constraints enforced between two events in different bMSCs in $U$. □

For an MSS $S = (U, N, succ, ref, T)$, a *path segment* is a sequence of intermediate nodes $v_0 \to v_1 \to \cdots \to v_n$ satisfying $(v_{i-1}, v_i) \in succ$ for any $i$ $(0 < i \le n)$. A *path* is a path segment $v_0 \to v_1 \to \cdots \to v_n$ such that $(\top, v_0) \in succ$ and $(v_n, \bot) \in succ$.

We interpret the timing constraints in an MSS by *local semantics*: select one path at one time and analyze its timing requirements, independently of other paths that may branch out of the selected one. As advocated by the ITU Recommendation Z.120 [1], the concatenation of bMSCs is interpreted by *asynchronous semantics*. The asynchronous concatenation of two bMSCs corresponds to concatenating two bMSCs instance by instance, which produces a new bMSC.

**Definition 5** Let $B_1 = (I_1, E_1, M_1, L_1, V_1, C_1)$ and $B_2 = (I_2, E_2, M_2, L_2, V_2, C_2)$ be two bMSCs $(E_1 \cap E_2 = \emptyset)$. The asynchronous concatenation of $B_1$ and $B_2$, denoted as $B_1 \circ B_2$, is a bMSC $B = (I, E, M, L, V, C)$ which is defined by

- $I = I_1 \cup I_2$.
- $E = E_1 \cup E_2$.
- $M = M_1 \cup M_2$.
- For $e \in E_1, L(e) = L_1(e)$, and for $e \in E_2, L(e) = L_2(e)$.
- $V = V_1 \cup V_2 \cup V_3 \cup V_4$ where
  $V_3 = \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, L_1(e_1) = L_2(e_2), e_2$ is a sending event$\}$,
  $V_4 = \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, L_1(e_1) = L_2(e_2), e_1, e_2$ are receiving events whose corresponding sending events appear on mutual instance axis.$\}$.
- $C = C_1 \cup C_2 \cup C_3$ where
  $C_3 = \{\epsilon_2 - \varpi_1 \le 0 \mid \epsilon_2$ is the start event of $B_2$, $\varpi_1$ is the end event of $B_1\}$. □

According to the above definition, every path in an MSS corresponds to a bMSC, and thus the behavior of an MSS is interpreted by the behavior of bMSCs.

**Definition 6** Let $S = (U, N, succ, ref, T)$ be an MSS. For any path segment $\rho = v_0 \to v_1 \to \cdots \to v_m$ in $S$, let $B$ be the bMSC obtained by concatenating $ref(v_i)$ $(0 \le i \le m)$ one by one, and $\mathcal{L}(\rho)$ be the set of timed event sequences which are in $\mathcal{L}(B)$ and satisfy the timing constraints in $T$. A timed event sequence $\sigma$ is a behavior of $S$ if and only if there is a path $\rho$ in $S$ such that $\sigma \in \mathcal{L}(\rho)$. □

## 2.3 Flexible loop-closed MSC specifications

For timing analysis of MSC specifications with asynchronous concatenation, the problems are difficult. It has been known in [2] that even for the MSC specifications without timing constraints, the model checking problem is undecidable. For giving efficient solutions, we enforce two restricting conditions on MSC specifications. For describing these two conditions, we first need to define *loops* in MSC specifications as follows.

For an MSS $S = (U, N, succ, ref, T)$, a path segment is called *simple* if all its nodes are distinct. Let $v_0 \to v_1 \to \cdots \to v_n$ be a simple path segment in $S$ such that $(\top, v_0) \in succ$. If there is $v_i$ $(0 \le i \le n)$ such that $(v_n, v_i) \in succ$, then the sequence $v_i \to v_{i+1} \to \cdots \to v_n \to v_i$ is a *loop*, and $v_i$ is the *loop-start node* of the loop.

Then we introduce the *loop-closed condition* for MSC specifications. For an MSS $S = (U, N, succ, ref, T)$, let $\rho = v_0 \to v_1 \to \cdots \to v_n$ be a path segment. For a timing constraint $a \le e - e' \le b$ in $T$, if $e$ occurs in $ref(v_i)$, $e'$ occurs in $ref(v_j)$ $(0 \le j < i \le n)$, and $e, e'$ do not occur in any $ref(v_k)$ $(j < k < i)$, then we say this timing constraint *combines* nodes $v_i$ and $v_j$ in $\rho$ (in this case, the occurrence time of $e$ in $ref(v_i)$ and the occurrence time of $e'$ in $ref(v_j)$ must satisfy this timing constraint). The *loop-closed condition* requires that any timing constraint in $T$ should not combine any two nodes which are inside and outside of a loop, respectively, i.e. any timing constraint in $T$ of the form $a \le e - e' \le b$ must satisfy:

- for any loop $v_0 \to v_1 \to \cdots \to v_m$, if $e$ occurs in $ref(v_i)$ $(0 \le i < m)$ and $e'$ does not occur in any $ref(v_j)$ $(0 \le j < i)$, then there is no simple path segment $v'_0 \to v'_1 \to \cdots \to v'_n$ such that $v'_n = v_0, e'$ occurs in $ref(v'_0)$, and that $e$ does not occur in any $ref(v'_k)$ $(0 \le k \le n)$; and
- for any loop $v_0 \to v_1 \to \cdots \to v_m$, if $e'$ occurs in $ref(v_i)$ $(0 \le i < m)$ and $e$ does not occur in any

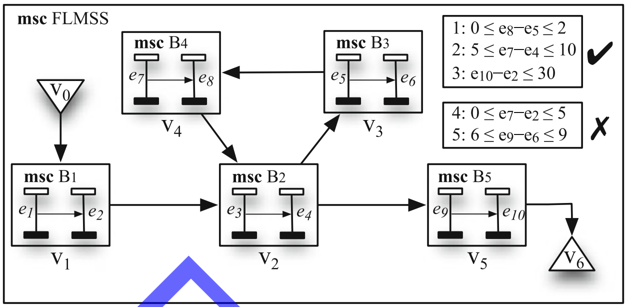


**Fig. 2** An illustrative flexible loop-closed MSC specification

$ref(v_j)$ $(i < j \leq m)$, then there is no simple path segment $v'_0 \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_n$ such that $v'_0 = v_0$, $e$ occurs in $ref(v'_n)$, and that $e'$ does not occur in any $ref(v'_k)$ $(0 \leq k \leq n)$.

For example, in the MSS depicted in Fig. 2, for the loop $v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_2$, the timing constraints 1, 2 and 3 satisfy the loop-closed condition, while the timing constraints 4 and 5 do not satisfy the loop-closed condition because timing constraint 4 $(0 \leq e_7 - e_2 \leq 5)$ combines node $v_4$ (inside the loop) and $v_1$ (outside the loop), and timing constraint 5 $(6 \leq e_9 - e_6 \leq 9)$ combines node $v_5$ (outside the loop) and $v_3$ (inside the loop). The loop-closed condition implies that if there is an external timing constraint associated with a loop then it must be enforced on the entire loop, i.e. for any loop, its any associated external timing constraint must be enforced on the entire loop.

The other condition enforced on MSC specifications is the *flexible condition*. For a bMSC $B = (I, E, M, L, V, C)$, we say $B$ is *flexible* if its execution time is flexible, i.e. there is no positive number $z$ such that for any behavior of $B$ of the form $(e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \cdots \rightarrow (e_m, t_m)$, $\sum_{j=1}^{m} t_j \leq z$. For an MSS $S = (U, N, succ, ref, T)$, the *flexible condition* requires that for any path $\rho$ in $S$, for any loop-start node $v$ in $\rho$, $ref(v)$ is a flexible bMSC. For example, the MSS depicted in Fig. 2 satisfies the flexible condition because $v_2$ is the loop-start node and the bMSC to which $v_2$ refers is a flexible bMSC. Note that for any loop in an MSS whose loop-start node refers to a flexible bMSC, though the execution of the loop-start node can take indeterminate amount of time, the execution time of the whole loop can still be constrained by enforcing timing constraints over the loop. For example, in Fig. 2 timing constraint 3 combines two node before and after the loop, and therefore constrains the execution time of the loop.

**Definition 7** An FLMSS is an MSS which satisfies the loop-closed condition and the flexible condition. □

The algorithm to check if an MSS is flexible loop-closed is presented in the appendix. The loop-closed condition and the flexible condition are rational for many real systems. For example, in many control systems the repetition of a control process often starts from the same control conditions. Also in many cases, a repeated control process may include several procedures which may take indeterminate amounts of time, such as preparing the supplies or calibrating the machines, while the entire process is required to be finished in a given time interval. These can be modeled by flexible bMSCs and loops constrained by timing constraints, which indicates the rationality of the flexible condition.

## 3 Timing analysis of MSC specifications

In this section, we give the solutions to *reachability analysis* and *bounded delay analysis* problems for MSC specifications.

### 3.1 Reachability analysis

*Reachability analysis* is to check if a given node of an MSS is *reachable* along a behavior of the MSS. Let $S = (U, N, succ, ref, T)$ be an MSS. For a given node $v \in N$, the reachability analysis checks if there is a path $\rho$ passing through $v$ which is of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_m$ such that $v_i = v$ $(0 \leq i \leq m)$ and that $\mathcal{L}(\rho) \neq \emptyset$.

Let $S = (U, N, succ, ref, T)$ be an MSS, and $\rho$ be a path in $S$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_m$ where $ref(v_i) = (I_i, E_i, M_i, L_i, V_i, C_i)$ for any $i$ $(0 \leq i \leq m)$. Since there could be $v_i$ and $v_j$ $(0 \leq i < j \leq m)$ such that $ref(v_i) = ref(v_j)$, by renaming, let $E_i \cap E_j = \emptyset$ for any $i, j$ $(0 \leq i < j \leq m)$. By concatenating $ref(v_i)$ $(0 \leq i \leq m)$ one by one, we can obtain a bMSC $B = (I, E, M, L, V, C)$. Let $E = \{e_0, e_1, \ldots e_n\}$, and $t_i$ represent the occurrence time of $e_i$ for any $i$ $(0 \leq i \leq n)$ in a timed event sequence in $\mathcal{L}(\rho)$. Then a group of linear inequalities on $t_0, t_1, \ldots, t_n$, denoted by $lp(\rho)$, can be constructed as follows:

- for any $t_i$ and $t_j$ $(0 \leq i < j \leq n)$, if $(e_i, e_j) \in V$, then $t_i - t_j \leq 0$, and
- $t_0, t_1, \ldots, t_n$ must satisfy all the timing constraints in $C$ and $T$, and the corresponding linear inequalities are given according to Definition 3.

Since $\mathcal{L}(\rho) \neq \emptyset$ if and only if $lp(\rho)$ has a solution, we can reduce the reachability analysis problem for a node $v$ of $S$ into a linear programming problem as follows: check if there is a path $\rho$ in $S$ passing through $v$ such that $lp(\rho) \neq \emptyset$. It is clear that in the worst case, we need to check all the paths in $S$ which pass through $v$. Since the number of paths of $S$ could be infinite, and the length of a path of $S$ could be infinite, we attempt to solve the problem based on a finite set of the finite paths of $S$.

Let $S = (U, N, succ, ref, T)$ be an MSS, and $v$ be a node in $N$. Let $\Delta(S, v)$ be a set of the paths in $S$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_m$ where $v_i = v$ $(0 \leq i \leq m)$, all $v_j$ $(0 \leq j \leq i)$ are distinct, and all $v_k$ $(i \leq k \leq m)$ are distinct. Intuitively, the node $v$ divides each path in $\Delta(S, v)$ into two simple path segments, which implies that $\Delta(S, v)$ is finite and each path in $\Delta(S, v)$ is finite because $N$ is finite. A path segment $\rho$ in $S$ is a *prefix* for $\Delta(S, v)$ if it may be extended into a path which is in $\Delta(S, v)$, i.e. there could be a path segment $\rho_1$ in $S$ such that $\rho \rightarrow \rho_1$ is in $\Delta(S, v)$. The following theorem tells us that if $S$ is flexible loop-closed, we just need to check each path in $\Delta(S, v)$ for reachability.

**Theorem 1** *Let $S = (U, N, succ, ref, T)$ be an FLMSS, and $v$ be a node in $N$. Then, $v$ is reachable if and only if there is a path $\rho \in \Delta(S, v)$ such that $\mathcal{L}(\rho) \neq \emptyset$.* □

The proof of the theorem is presented in the appendix. Based on the above theorem, we can develop an algorithm to check if a node $v$ in an MSS $S$ is reachable (cf. Fig. 3). In the algorithm, first we check if $S$ is flexible loop-closed, and assign the result to the boolean variable $flexible\_loop\_closed$. Then, the algorithm traverses the state space of the nodes of $S$ in a depth first manner starting from the start node $\top$. The path in the state space that we have so far traversed is stored in the list variable $currentpath$. For each successive node $node$ of the last node of $currentpath$, we first check whether the path segment $\rho$ corresponding to the concatenation of $currentpath$ and $node$ is in $\Delta(S, v)$. If yes, then we check if $\mathcal{L}(\rho) \neq \emptyset$ by linear programming, and return **true** when $\mathcal{L}(\rho) \neq \emptyset$. If the path segment corresponding to the concatenation of $currentpath$ and $node$ is a prefix for $\Delta(S, v)$, then we add $node$ to the current path and start the search from it, otherwise we search the other successive nodes. The algorithm backtracks when all the successive nodes of the last node of $currentpath$ are explored. After finishing the depth first search, we return **false** when $S$ is flexible loop-closed, and **undecided** when $S$ is not flexible loop-closed. Notice that the algorithm can answer **true** for some MSSs which are not flexible loop-closed, but not all. It is thus a decision procedure for the FLMSSs, and a semi-decisions procedure for the general MSSs.

```
check if $S$ is flexible loop-closed;
if yes then $flexible\_loop\_closed$ :=true;
else $flexible\_loop\_closed$ :=false;
$currentpath := \langle \top \rangle$;
repeat
    $node :=$ the last node of $currentpath$;
    if all successive nodes of $node$ are explored through
      $currentpath$
    then delete the last node of $currentpath$;
    else begin /*explore an unexplored successive node*/
        $node :=$ a successive node of $node$ not explored
              through $currentpath$;
        if the path segment $\rho$ corresponding to the con-
          catenation of $currentpath$ and $node$ is in $\Delta(S, v)$
        then begin check if $\mathcal{L}(\rho) \neq \emptyset$;
              if yes then return true;
           end
        if the path segment corresponding to the con-
          catenation of $currentpath$ and $node$ is a prefix
          for $\Delta(S, v)$
        then append $node$ to $currentpath$;
        end
until $currentpath = \langle \rangle$;
if $flexible\_loop\_closed$ then return false;
else return undecided.
```

**Fig. 3** Algorithm for reachability analysis

### 3.2 Bounded delay analysis

The *bounded delay analysis* is to check if the time distance between the two given events in any behavior of an MSS is not smaller or greater than a given real number, which is called the *minimal bounded delay analysis* or the *maximal bounded delay analysis*, respectively.

For an MSS $S$, a *bounded delay specification* consists of two events $e, e'$ and a real number $d$ ($e$ and $e'$ occur in different node of $S$), denoted by $\mathcal{S}_B(e, e', d)$, which can be a *minimal (or maximal) bounded delay specification* $\mathcal{S}_B^m(e, e', d)$ [or $\mathcal{S}_B^M(e, e', d)$], and requires that the time distance between $e$ and $e'$ in any behavior of $S$ is not smaller (or greater) than $d$.

Let $S = (U, N, succ, ref, T)$ be an MSS, $\mathcal{S}_B^m(e, e', d)$ [or $\mathcal{S}_B^M(e, e', d)$] be a bounded delay specification, and $\sigma$ be a behavior of $S$ of the form

$$(e_0, t_0) \rightarrow \cdots \rightarrow (e_i, t_i) \rightarrow \cdots \rightarrow (e_j, t_j) \rightarrow \cdots \rightarrow (e_n, t_n).$$

If for any $i$ and $j$ $(0 \leq i < j \leq n)$ such that $e_i = e'$ and $e_j = e$, and that $e_k \neq e \wedge e_k \neq e'$ for any $k$ $(i < k < j)$, $t_{i+1} + t_{i+2} + \cdots + t_j \geq (\leq)\ d$, then we say that $\sigma$ satisfies $\mathcal{S}_B^m(e, e', d)$ [or $\mathcal{S}_B^M(e, e', d)$]. We define that a path $\rho$ in $S$ satisfies $\mathcal{S}_B(e, e', d)$ if any $\sigma \in \mathcal{L}(\rho)$ satisfies $\mathcal{S}_B(e, e', d)$, and that $S$ satisfies $\mathcal{S}_B(e, e', d)$ if any path in $S$ satisfies $\mathcal{S}_B(e, e', d)$.

Let $S = (U, N, succ, ref, T)$ be an MSS, $\mathcal{S}_B^m(e, e', d)$ [or $\mathcal{S}_B^M(e, e', d)$] be a bounded delay specification, and $\rho$ be a path in $S$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_m$ where $ref(v_i) = (I_i, E_i, M_i, L_i, V_i, C_i)$ for any $i$ $(0 \leq i \leq m)$. Since there could be $v_i$ and $v_j$ $(0 \leq i < j \leq m)$ such that $ref(v_i) = ref(v_j)$, by renaming, let $E_i \cap E_j = \emptyset$ for any $i, j$ $(0 \leq i < j \leq m)$. By concatenating $ref(v_i)$ $(0 \leq i \leq m)$ one by one, we can obtain a bMSC $B = (I, E, M, L, V, C)$. Let $E = \{e_0, e_1, \ldots e_n\}$, and $t_i$ represent the occurrence time of $e_i$ for any $i$ $(0 \leq i \leq n)$ in a timed event sequence in $\mathcal{L}(\rho)$. By linear programming, we can check $\rho$ for $\mathcal{S}_B^m(e, e', d)$ [or $\mathcal{S}_B^M(e, e', d)$] as follows: for any $e_i = e'$ and $e_j = e$ $(0 \leq i, j \leq n, i \neq j)$, find the minimum (or maximum) value of the function $t_j - t_i$ subject to the linear constraint $lp(\rho)$ and $t_j - t_i \geq 0$, and check whether it is not smaller (or greater) than $d$ and whether in the corresponding timed event sequence $e$ and $e'$ do not occur in between $e_i$ and $e_j$. For all the paths in $S$, we attempt to solve the problem based on a finite set of the finite paths in $S$.

Let $S = (U, N, succ, ref, T)$ be an MSS, and $\mathcal{S}_B(e, e', d)$ be a bounded delay specification. Let $\Delta(S, \mathcal{S}_B(e, e', d))$ be the set of the paths in $S$ of the form

$$v_0 \to \cdots \to v_i \to v_{i+1} \to \cdots \to v_j \to v_{j+1} \to \cdots \to v_m$$

where

- either $e'$ occurs in $ref(v_i)$ and $e$ occurs in $ref(v_j)$, or $e$ occurs in $ref(v_i)$ and $e'$ occurs in $ref(v_j)$ $(0 \le i \le j \le m)$; and
- all $v_l$ $(0 \le l < i)$ are distinct, all $v_k$ $(i < k < j)$ are distinct, and all $v_p$ $(j < p \le m)$ are distinct.

Intuitively, each path in $\Delta(S, \mathcal{S}_B(e, e', d))$ is separated by $e'$ and $e$ into three simple path segments. For an MSS $S$, for a bounded delay specification $\mathcal{S}_B(e, e', d)$, a path segment $\rho$ in $S$ is a *prefix* for $\Delta(S, \mathcal{S}_B(e, e', d))$ if it may be extended into a path which is in $\Delta(S, \mathcal{S}_B(e, e', d))$, i.e. there could be a path segment $\rho_1$ in $S$ such that $\rho \to \rho_1$ is in $\Delta(S, \mathcal{S}_B(e, e', d))$.

For an FLMSS $S$, the problem of checking $S$ for a bounded delay specification $\mathcal{S}_B(e, e', d)$ can be solved by checking each path in $\Delta(S, \mathcal{S}_B(e, e', d))$, which is supported by the following theorem.

**Theorem 2** *Let $S$ be an FLMSS, and $\mathcal{S}_B(e, e', d)$ be a bounded delay specification. Then, $S$ satisfies $\mathcal{S}_B(e, e', d)$ if and only if any path in $\Delta(S, \mathcal{S}_B(e, e', d))$ satisfies $\mathcal{S}_B(e, e', d)$.* □

The proof of the theorem is presented in the appendix. Based on Theorem 2, we can develop an algorithm to check if an MSS $S$ satisfies a bounded delay specification (cf. Fig. 4). The structure of the algorithm is similar to the algorithm depicted in Fig. 3. Since the algorithm can answer **false** for some MSSs which are not flexible loop-closed, but not all, it is thus a decision procedure for the FLMSSs, and a semi-decision procedure for the general MSSs.

```
check if $S$ is flexible loop-closed;
if yes then $flexible\_loop\_closed$ :=true;
else $flexible\_loop\_closed$ :=false;
$currentpath := \langle\top\rangle$;
repeat
    $node :=$ the last node of $currentpath$;
    if all successive nodes of $node$ are explored through
       $currentpath$
    then delete the last node of $currentpath$;
    else begin /*explore an unexplored successive node*/
           $node :=$ a successive node of $node$ not explored
                  through $currentpath$;
           if the path segment $\rho$ corresponding to the concatenation of $currentpath$ and $node$ is in
              $\Delta(S, \mathcal{S}_B(e, e', d))$
           then begin check if $\rho$ satisfies $\mathcal{S}_B(e, e', d)$;
                  if no then return false;
                end
           if the path segment corresponding to the concatenation of $currentpath$ and $node$ is a prefix
              for $\Delta(S, \mathcal{S}_B(e, e', d))$
           then append $node$ to $currentpath$;
         end
until $currentpath = \langle\rangle$;
if $flexible\_loop\_closed$ then return true;
else return undecided.
```

**Fig. 4** Algorithm for bounded delay analysis

### 3.3 Complexity of Algorithms

The complexity of the algorithms presented in this section consists of two parts: one is from the searching the node state space of an MSS, and the other includes the number and size of the linear programs we need to solve in the algorithms.

Let $S = (U, N, succ, ref, T)$ be an MSS, and $m$ be the number of the nodes in $S$, i.e. $m = |N|$. For the node state space search, the complexity is not greater than $(m!)^2$ and $(m!)^3$ for the reachability analysis and the bounded delay analysis, respectively. For the linear program solving, the number of linear programs we need to solve is the number of the paths in the sets $\Delta(G, v)$ or $\Delta(G, \mathcal{S}_B(e, e', d))$, which is not greater than $(m!)^2$ and $(m!)^3$, respectively. It is known that a linear program can be solved in polynomial time[12], e.g. the algorithm in [12] requires $O(n^{3.5}L)$ arithmetic operations on $O(L)$ bit numbers under the worst case, where $n$ is the number of variables and $L$ is the number of bits in the input. Therefore, the upper bounds of the algorithms are $O((m!)^2 \cdot p(n, L))$ and $O((m!)^3 \cdot p(n, L))$ for the reachability analysis and the bounded delay analysis, respectively, where $p(n, L)$ is in polynomial complexity which can vary depending on the different polynomial linear programming algorithms, and $m, n$ and $L$ have the same meanings as above.

The complexity of the algorithms presented above are based on the worst case, which rarely appears in practice. First and foremost, for the number of the linear programs which need solving to reach the numbers in the above analysis, the MSS, which is essentially a directed graph, must be complete. In real world applications, this is hardly the case. In practical use, most of the edges are one directional and even the number of loops are relatively small, which results in quite a small number of paths compared to the one in the worst case. Secondly, since one event is corresponding to one variable in the linear programs, the size of the linear programs we need to solve in the algorithms is proportional to the maximal number of the events occurring in a path in the sets, and to the maximal bits which encode the timing constraints in a path in the sets. As mentioned above, in our algorithms the size of the longest path is only $2m$ for reachability analysis and $3m$ for bounded delay analysis, which assures the size of the linear programs are manageable.

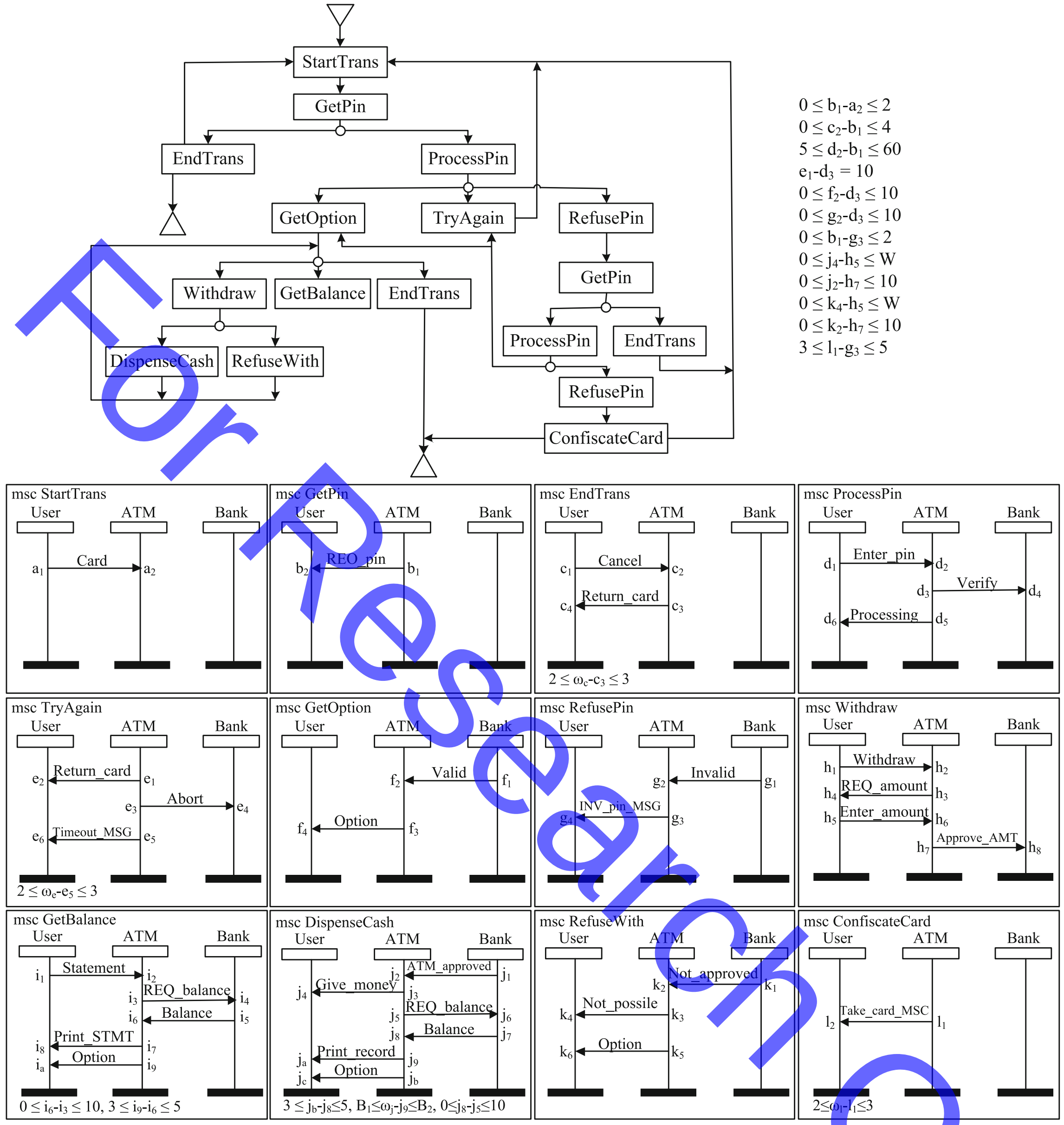


**Fig. 5** MSC specifications for the ATM system

## 4 Implementation and evaluation

The solutions presented in the above section have been implemented into our tool TASS [13]. TASS is a prototype tool based on our previous work [11,13], which is a timing analyzer of scenario-based specifications expressed by UML interaction models, and adopts the synchronous concatenation for scenario compositions. TASS is written in Java as an Eclipse plugin [14], and can be downloaded from its website [15]. We have extended TASS by accepting MSC specifications, and implementing the timing analysis algorithms for MSC specifications with asynchronous concatenation presented in this paper. The linear programming software package integrated in TASS is from OR-Objects of DRA Systems [16] which is a free collection of Java classes for developing operations research, scientific and engineering applications.

We evaluate the performance of TASS on the well-known example of automatic teller machine (ATM) system [4], shown in Fig. 5. The ATM system consists of the three components: the customers (User), the ATM controller (ATM), and a host computer in a bank (Bank). Initially, the ATM controller waits to receive the customer's bank card and requests a pin number in [0, 2] seconds after receiving a card ($0 \leq b_1 - a_2 \leq 2$, bMSCs StartTrans and GetPin). Then, it either receives a request to cancel the transaction

**Table 1** Sample results of timing analysis of the ATM MSS

| Case | Problem | | | |
|---|---|---|---|---|
| | Reachability | | $\mathcal{S}^m_B(j_4, a_1, 20)$ | |
| | Result | Time | Result | Time |
| $w = 4, B_1 = 0, B_2 = \infty, T_1 = 0.5, T_2 = 2$ | No | 2.142 s | a | a |
| $w = 6, B_1 = 0.5, B_2 = 1, T_1 = 0.5, T_2 = 2$ | Yes | 30 ms | No | 31 ms |
| $w = 6, B_1 = 1, B_2 = 2, T_1 = 0.5, T_2 = 2$ | Yes | 29 ms | No | 30 ms |
| $w = 9, B_1 = 2, B_2 = 3, T_1 = 1, T_2 = 3$ | Yes | 30 ms | Yes | 3.775 s |

[a] The verification problem disappears because the node is unreachable

within [0,4] seconds ($0 \leq c_2 - b_1 \leq 4$, bMSC EndTrans), or receives the customer's pin number with [5,60] seconds ($5 \leq d_2 - b_1 \leq 60$, bMSC ProcessPin). If the ATM receives a request to cancel the transaction, it returns the customer's card and takes [2, 3] seconds to return to its initial state ($2 \leq \varpi_c - c_3 \leq 3$, bMSC EndTrans). The ATM expects a reply from the bank within 10 s, which form the following timing constraints:

$$0 \leq f_2 - d_3 \leq 10,\ 0 \leq g_2 - d_3 \leq 10,\ 0 \leq j_2 - h_7 \leq 10$$
$$0 \leq k_2 - h_7 \leq 10,\ 0 \leq i_6 - i_3 \leq 10,\quad 0 \leq j_8 - j_5 \leq 10.$$

If no reply from the bank is received in a delay of 10 s, the card is returned, an appropriate message is then displayed, and the ATM takes [2,3] seconds to return to its initial state ($e_1 - d_3 = 10,\ 2 \leq \varpi_e - e_5 \leq 3$, bMSC TryAgain). Our specification also describes the following constraints: a customer expects a withdraw request to be processed within $[0, W]$ seconds relative to the time of entering an amount ($0 \leq j_4 - h_5 \leq W,\ 0 \leq k_4 - h_5 \leq W$); the ATM takes $[B_1, B_2]$ seconds for book-keeping after dispensing cash ($B_1 \leq \varpi_j - j_9 \leq B_2$, bMSC DispenseCash); the ATM takes $[3, 5]$ seconds to print a receipt after receiving the balance information from the bank ($3 \leq j_b - j_8 \leq 5,\ 3 \leq i_9 - i_6 \leq 5$, bMSC DispenseCash, bMSC GetBalance); and in the case of refusing pin number, at the first time the ATM takes $[0, 2]$ seconds to request a pin number again after sending the information for the invalid pin number ($0 \leq b_1 - g_3 \leq 2$), and at the second time it takes $[3, 5]$ seconds to confiscate the card and inform the customer ($3 \leq l_1 - g_3 \leq 5$, bMSC ConfiscateCard). Each ATM-customer communication takes at least $T_1$ seconds, and each ATM-bank communication takes at least $T_2$ seconds, which we do not explicitly represent in the chart. The ATM MSS can be regarded as an FLMSS because the ATM system requirements are compatible with asynchoronous concatenation, and the loop-closed condition and flexible condition are satisfied.

On an HP laptop (Intel Core 2 Duo CPU 2.2 GHz/ 2 GB RAM), TASS is used for solving the following timing analysis problems:

- Reachability analysis: we check if the node bMSC DispenseCash is reachable in the specification.
- Bound delay analysis: for the security consideration it is necessary to record the process for withdrawing money by the camera embedded in the ATM. We require that every process for withdrawing money takes enough time for recording, which forms a minimal bounded delay specification $\mathcal{S}^m_B(j_4, a_1, 20)$.

Given the various values of $W$, $B_1$, $B_2$, $T_1$, and $T_2$, TASS reports the corresponding sample results, which is depicted in Table 1. As we expect, the experiment results are satisfactory since we only assigned 50M memory to the Java Virtual Machine, and all the analysis tasks finished in split seconds including the time to check whether the MSS is flexible loop-closed. There are two reasons for such good performance. One is that our algorithm is to recursively traverse directly on the structure of the MSS and check each relevant path in a depth-first manner, one by one. No matter how big the whole model state is, it only cares about the currently visiting path and therefore consumes little memory. The other reason is that our algorithm takes advantage of the characteristics of FLMSSs. For the FLMSSs which contain loops, only paths which contain up to two simple path segments need to be checked for reachability according to Theorem 1, and those which contain up to three simple path segments need to be checked for bounded delay specifications according to Theorem 2.

## 5 Related work and conclusion

MSC specifications and similar formalisms are increasingly being used by designers for specifying requirements. In addition such specifications are naturally compatible with object-oriented design methods, and are being supported by modern software engineering methodologies such as UML [5,6]. They are playing an increasingly important role in the design of software systems.

The bMSCs and simple UML sequence diagrams describe exactly one scenario. In [3], the problem of checking bMSCs with delay intervals for timing consistency is reduced to computing negative cost cycles and shortest distances in a weighted directed graph using temporal constraint network techniques. In [7], the same techniques are used for timing consistency analysis of a class of simple UML sequence diagrams in which all timing constraints are of the form $a \leq e_1 - e_2 \leq b$. In [8] a case study is investigated for UML sequence diagram-based verification in which a simple sequence diagram is transformed to a set of timed automata and then checked for a timed automata-based implementation by a model checking [17] tool. In [18], MSCs with timed stamps at events are interpreted as event clock communicating finite-state machines, which later are used to construct global timed automata and the model checking problem is then reduced to checking timed automata.

To specify a complete system, specifications with multiple scenarios are necessary. The simple property of time consistency has been extensively studied for these specifications. In [19] and [20], the problems of checking compositions of UML sequence diagrams for timing consistency and timing inconsistency are considered. The problem of checking MSC specifications for timing consistency is considered in [4], which just gives a sufficient condition for timing consistency. In [21] the strong and weak time consistency are studied for MSC specifications in which the timing constraints can be absolute or relative time constraints. The reachability analysis, *constraint conformance analysis*, and bounded delay analysis problems for scenario-based specifications expressed by UML models are considered in [11], which are conducted on synchronous compositional semantics (synchronous concatenation).

In [2], the model checking problem of MSC specifications with asynchronous concatenation is shown undecidable, due to unbounded drift between instances. To translate the MSC specifications to automata, a restricted class called bounded MSC-graphs is proposed in which the distance that instances can drift is bounded, or in other words, the number of the pending messages is bounded. Theoretically, under asynchronous semantics, to translate MSC specifications with timing constraints to timed automata requires a similar restriction on the structure of the MSC specifications. For example, in [22], MSC specifications with timing constraints are used as behavioral specifications and timed automata are regarded as system models. The MSC specifications called locally synchronized message sequence graphs, which require that all the communication channels are bounded, are transformed to MSC event clock automata, which can be translated into equivalent timed automata and used for checking the automata which model the system. The MSC specifications considered in this paper do not have any restriction on their structures, and therefore cannot be transformed into timed automata. Furthermore, the timed automata-based approach requires the timing constraints be only related to one time distance between a pair of events. If the timing constraints considered in this paper are allowed, which are about the relations among multiple time distances between pairs of events, we have to compare multiple clocks in the corresponding timed automata, which will result in that the corresponding model checking problems are undecidable [23].

In this paper, we give the linear programming-based solutions to timing analysis of MSC specifications with asynchronous concatenation. For FLMSSs, which satisfy that for any loop, its first node is flexible in execution time and its any associated external timing constraint is enforced on the entire loop, we develop the efficient algorithms for the reachability analysis and bounded delay analysis problems. These algorithms are also a semi-decision procedure for the general MSC specifications.

**Acknowledgments** Thanks to the anonymous reviewers and editors for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No.90818022, No.61021062), the National Grand Fundamental Research 973 Program of China (No.2009CB320702), and by the National 863 High-Tech Programme of China (No.2011AA010103, No.2012AA011205).

## Appendix A: Algorithm to check if an MSS is flexible loop-closed

Let $S = (U, N, succ, ref, T)$ be an MSS. According to the definition, for checking if the flexible condition is held for $S$, we just need to check all the bMSCs corresponding to the loop-start nodes for their execution time. The loop-start nodes can be obtained from the set $loopset$ which records all the loops in $S$, as depicted in Fig. 6. So we only need to consider the problem of checking if a bMSC is flexible, which is equivalent to the problem to check if there is an up-bound on the time distance between the start event and the end event of the bMSC. This can be solved by linear programming. For a bMSC $B = (I, E, M, L, V, C)$. Let $E = \{e_0, e_1, \dots e_n\}$, $e_0$ be the start event, $e_n$ be the end event, and $t_i$ represent the occurrence time of $e_i$ for any $i$ $(0 \leq i \leq n)$. Then, the problem of checking if the time distance between $e_0$ and $e_n$ has a up-bound in any $\sigma \in \mathcal{L}(B)$ can be solved as follows: check if the maximal value of the objective function $t_n - t_0$ regarding to the linear program $lp(B)$ is bounded.

Let $S = (U, N, succ, ref, T)$ be an MSS. According to the definition, for checking if the loop-closed condition is held for $S$, we just need to traverse all the path segments in $S$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_n$ where $(\top, v_0) \in succ$, $v_i$ $(0 \leq i \leq n)$ is a loop-start node, all $v_j$ $(0 \leq j \leq i)$ are distinct, and all $v_k$ $(i < k \leq n)$ are distinct,, which are called *checked path segments*. A path segment $\rho$ is a *prefix* for a checked path segment if it may be extended into a checked path segment, i.e. there could be a path segment $\rho_1$ such that $\rho \rightarrow \rho_1$ becomes a checked path segment.

The following presents an algorithm to check if an MSS $S$ is flexible loop-closed (cf. Fig. 6). The algorithm is based

```
$currentpath := \langle v_I \rangle$; $loopset := \emptyset$;
repeat
    $node$ := the last node of $currentpath$;
    if all successive nodes of $node$ are explored through
       $currentpath$
    then delete the last node of $currentpath$;
    else begin /*explore an unexplored successive node*/
        $node$ := a successive node of $node$ not explored
                  through $currentpath$;
        if $node$ is in $currentpath$ /*a loop is discovered*/
        then begin
              put the loop into $loopset$;
              check if the loop-start node is flexible;
              if no then return false;
           end
        else append $node$ to $currentpath$;
    end
until $currentpath = \langle\rangle$.
```

```
$currentpath := \langle \top \rangle$;
repeat
    $node$ := the last node of $currentpath$;
    if all successive nodes of $node$ are explored through
       $currentpath$
    then delete the last node of $currentpath$;
    else begin /*explore an unexplored successive node*/
        $node$ := a successive node of $node$ not explored
                  through $currentpath$;
        if the path segment $\rho$ corresponding to
           $currentpath$ is a checked path segment
        then begin
              check if the loop-closed condition
              is satisfied;
              if no then return false;
           end
        if the path segment corresponding to the conca-
           tenation of $currentpath$ and $node$ is a prefix
        then append $node$ to $currentpath$;
    end
until $currentpath = \langle\rangle$;
return true.
```

**Fig. 6** Algorithm to check if an MSS is flexible loop-closed

on depth-first search method. The main data structure in the algorithm includes a list *currentpath* of nodes which is used to record the current paths, and a set *loopset* of loops which records all the loops in $S$. The algorithm consists of two main steps. First, by a depth-first search the algorithm finds out all loops in $S$ and checks if any loop-start node is flexible. Then it traverses all the checked paths segments in $S$ to check if $S$ is loop-closed. The complexity of the algorithm is proportional to the number of the prefixes and the size of the longest prefix for loop-closed condition in $S$.

## Appendix B: Proofs of Theorems

**Theorem 1** *Let $S = (U, N, succ, ref, T)$ be an FLMSS, and $v$ be a node in $N$. Then, $v$ is reachable if and only if there is a path $\rho \in \Delta(S, v)$ such that $\mathcal{L}(\rho) \neq \emptyset$.*

*Proof* It is clear that one half of the claim holds: if there is $\rho \in \Delta(S, v)$ such that $\mathcal{L}(\rho) \neq \emptyset$, then $v$ is reachable. The other half of the claim can be proved as follows. Suppose that $v$ is reachable. Then there is a path $\rho$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_m$ such that $v_i = v$ $(0 \leq i \leq m)$, and a timed event sequence $\sigma \in \mathcal{L}(\rho)$ of the form $(e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \cdots \rightarrow (e_n, t_n)$. If all $v_j$ $(0 \leq j \leq i)$ are distinct and all $v_k$ $(i \leq k \leq m)$ are distinct, then $\rho \in \Delta(S, v)$ and we are done. Otherwise, there are $v_{p'}$ and $v_{q'}$ $(0 \leq p' < q' \leq i)$ such that $v_{p'} \rightarrow v_{p'+1} \rightarrow \cdots \rightarrow v_{q'}$ is loop $(v_{p'} = v_{q'})$, and (or) there are $v_{p''}$ and $v_{q''}$ $(i \leq p'' < q'' \leq m)$ such that $v_{p''} \rightarrow v_{p''+1} \rightarrow \cdots \rightarrow v_{q''}$ is loop $(v_{p''} = v_{q''})$. Suppose that the rightmost loop in $\rho$ is $v_p \rightarrow v_{p+1} \rightarrow \cdots \rightarrow v_q$. By removing the subsequences $v_{p+1} \rightarrow v_{p+2} \rightarrow \cdots \rightarrow v_q$ from $\rho$, we can get a path $\rho'$, and in the following we will show by reconstructing $\sigma$ we can get $\sigma'$ such that $\sigma' \in \mathcal{L}(\rho')$.

First, we get $\sigma'$ from $\sigma$ by removing all the events in the bMSCs corresponding to the nodes in the subsequence $v_{p+1} \rightarrow v_{p+2} \rightarrow \cdots \rightarrow v_q$, without changing the time distances between the rest events in $\sigma$, i.e. for any timed event $(e_f, t_f)$ in $\sigma$ $(0 \leq f \leq n)$, if $e_f$ occurs in $ref(v_l)$ $(p < l \leq q)$ and $(e_f, t_f)$ is not the last timed event in $\sigma$, then remove $(e_f, t_f)$ from $\sigma$, and change $(e_{f+1}, t_{f+1})$ to $(e_{f+1}, t_f + t_{f+1})$.

Secondly, to be consistent with asynchronous concatenation, we require that if after the above removal of nodes, $v_p$ is not the last node of $\rho'$, then the start event of bMSC $ref(v_{q+1})$ happen before or simultaneously with the end event of $ref(v_p)$. If this is the case in $\sigma'$, then the reconstruction is done. Otherwise, let $t$ be the time distance between these two events in $\sigma'$. Since $ref(v_p)$ is a flexible bMSC, we can always find a new behavior of $ref(v_p)$ of the form $\sigma_p = (e_u, t_u) \rightarrow (e_{u+1}, t_{u+1}) \rightarrow \cdots \rightarrow (e_v, t_v)$ whose execution time is $t$ time units longer to replace the timed events of $ref(v_p)$ in $\sigma'$, without changing the time distances between the events which are not in $ref(v_p)$, i.e.

- we remove the old behavior of $ref(v_p)$ from $\sigma'$, i.e. for any timed event $(e_f, t_f)$ in $\sigma'$ $(0 \leq f \leq n)$, if $e_f$ occurs in $ref(v_p)$ and $e_f$ is not the start event of $ref(v_p)$, then remove $(e_f, t_f)$, and change $(e_{f+1}, t_{f+1})$ to $(e_{f+1}, t_f + t_{f+1})$. The start event of $ref(v_p)$ is left in $\sigma'$ so that we know where to begin the insertion of the new behavior; and
- we insert the new behavior $\sigma_p$ of $ref(v_p)$ into $\sigma'$, i.e. remove the first timed event from $\sigma_p$ as it is already in $\sigma'$ and no longer needed. Then repeat the following process until $\sigma_p = \emptyset$. For the first timed event $(e_w, t_w)$ in $\sigma_p$ $(u \leq w \leq v)$, locate the timed event $(e_r, t_r)$ in $\sigma'$ $(0 \leq r \leq n)$ such that $e_r = e_{w-1}$ where $e_{w-1}$ is the event right before $e_w$ in the original $\sigma_p$. Since the new behavior $\sigma_p$ of $ref(v_p)$ is only $t$ time units longer than the old one, any event of $ref(v_{q+1})$ in $\sigma'$ does not happen

before any event in $\sigma_p$, and since there are at least two events in $ref(v_{q+1})$, we can always find two timed event $(e_s, t_s)$ and $(e_{s+1}, t_{s+1})$ $(r \leq s \leq n)$ in $\sigma'$ such that $t_{r+1} + t_{r+2} + \cdots + t_s \leq t_w$ and $t_{r+1} + t_{r+2} + \cdots + t_{s+1} > t_w$, insert $(e_w, t'_w)$ between $(e_s, t_s)$ and $(e_{s+1}, t_{s+1})$ where $t'_w = t_w - (t_{r+1} + t_{r+2} + \cdots + t_s)$, change $(e_{s+1}, t_{s+1})$ to $(e_{s+1}, t_{s+1} - t'_w)$, and remove $(e_w, t_w)$ from $\sigma_p$.

From the construction steps it is easy to see that the time distances between any two events which are not in $ref(v_p)$ are unchanged in $\sigma'$ after the reconstruction of $\sigma$. So all the timing constraints in the bMSCs which make up $\rho'$ are satisfied.

Since $S$ satisfies the loop-closed condition, any timing constraint in $T$ does not combine any two nodes which are inside and outside a loop, respectively. It follows that there is no timing constraint combines $v_p$ and the other node in $\rho'$, so no timing constraints are affected by the replacement of the timed event sequence of $ref(v_p)$. It also follows that all timing constraints which combine two nodes outside the removed loop remain unchanged after the removal of the loop nodes. Therefore, all timing constraints in $T$ are satisfied.

Plus it is obvious that $\sigma'$ meets the requirement of asynchronous concatenation. Hence $\sigma'$ is a behavior of $\rho'$, which indicates $\mathcal{L}(\rho') \neq \emptyset$.

By applying the above step repeatedly, we can get a path $\rho''$ of the form $v'_1 \to v'_2 \to \cdots \to v'_j \to v_i \to v'_{j+1} \cdots \to v'_k$ such that $\mathcal{L}(\rho'') \neq \emptyset$, all $v'_h$ $(0 \leq h \leq j)$ and $v_i$ are distinct, and that all $v'_h$ $(j < h \leq k)$ and $v_i$ are distinct. It follows that $\rho''$ is in $\Delta(S, v)$, from which the claim holds. □

**Theorem 2** *Let $S$ be an FLMSS, and $\mathcal{S}_B(e, e', d)$ be a bounded delay specification. Then, $S$ satisfies $\mathcal{S}_B(e, e', d)$ if and only if any path in $\Delta(S, \mathcal{S}_B(e, e', d))$ satisfies $\mathcal{S}_B(e, e', d)$.*

*Proof* It is clear that one half of the claim holds: if $S$ satisfies $\mathcal{S}_B(e, e', d)$, then any path in $\Delta(S, \mathcal{S}_B(e, e', d))$ satisfies $\mathcal{S}_B(e, e', d)$. The other half of the claim can be proved as follows. Suppose that any path in $\Delta(S, \mathcal{S}_B(e, e', d))$ satisfies $\mathcal{S}_B(e, e', d)$, and there is a path $\rho$ of $S$ such that a behavior $\sigma \in \mathcal{L}(\rho)$ does not satisfy $\mathcal{S}_B(e, e', d)$. Without losing generality, suppose that $\rho$ is of the form $v_0 \to v_1 \to \cdots \to v_m$, and $\sigma$ is of the form $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_k, t_k) \to \cdots \to (e_l, t_l) \to \cdots \to (e_n, t_n)$ where $e' = e_k$ and $e = e_l$ $(0 \leq k < l \leq n)$, and $e, e'$ do not appear in any $(e_h, t_h)$ $(k < h < l)$. The time distance between $e_l$ and $e_k$ is smaller (or greater) than $d$ for the bounded delay specification $\mathcal{S}^m_B(e, e', d)$ [or $\mathcal{S}^M_B(e, e', d)$]. In the following, we prove that we can construct a path $\rho'' \in \Delta(S, \mathcal{S}_B(e, e', d))$ such that there is a timed event sequence $\sigma'' \in \mathcal{L}(\rho'')$ which does not satisfies $\mathcal{S}_B(e, e', d)$, which results in a contradiction and implies that the claim holds. Suppose that $e'$ occurs in $ref(v_i)$ and $e$ occurs in $ref(v_j)$, or $e$ occurs in $ref(v_i)$ and $e'$ occurs in $ref(v_j)$ $(0 \leq i < j \leq m)$. Since $\rho \notin \Delta(S, \mathcal{S}_B(e, e', d))$,

- there are $v_{p'}$ and $v_{q'}$ $(0 \leq p' < q' < i)$ such that $v_{p'} \to v_{p'+1} \to \cdots \to v_{q'}$ is loop $(v_{p'} = v_{q'})$, and (or)
- there are $v_{p''}$ and $v_{q''}$ $(i < p'' < q'' < j)$ such that $v_{p''} \to v_{p''+1} \to \cdots \to v_{q''}$ is loop $(v_{p''} = v_{q''})$, and (or)
- there are $v_{p'''}$ and $v_{q'''}$ $(j < p''' < q''' \leq m)$ such that $v_{p'''} \to v_{p'''+1} \to \cdots \to v_{q'''}$ is loop $(v_{p'''} = v_{q'''})$.

Suppose that the rightmost loop in $\rho$ is $v_p \to v_{p+1} \to \cdots \to v_q$. By removing the subsequences $v_{p+1} \to v_{p+2} \to \cdots \to v_q$ from $\rho$, we can get a path $\rho'$, and using the same constructing steps used in the proofs of Theorem 1, we can get $\sigma'$ such that $\sigma' \in \mathcal{L}(\rho')$.

By applying the above step repeatedly, we can get a path $\rho'' \in \Delta(S, \mathcal{S}_B(e, e', d))$ such that there is a timed event sequence $\sigma'' \in \mathcal{L}(\rho'')$ which does not satisfy $\mathcal{S}_B(e, e', d)$, which results in a contradiction and implies that the claim holds. □

## References

1. ITU-T. Recommendation, Z120. Message Sequence Charts. International Telecommunication Union, Standardization Sector, Genève, Switzerland (2000)
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Proceedings of the 10th International Conference on Concurrency Theory (CONCUR '99), pp. 114–129. Springer, Berlin (1999)
3. Alur, R., Holzmann, G.J., Peled, D.: An analyzer for message sequence charts. In: Software-Concepts and Tools, vol. 17, pp. 70–77. Springer, Berlin (1996)
4. Ben-Abdallah, H., Leue, S.: Timing constraints in message sequence chart specifications. In: Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing and Verification (FORTE/PSTV '97), pp. 91–106. Chapman & Hall, Ltd, London (1998)
5. Rumbaugh, J., Jacobson, I., Booch, G. (eds.): The Unified Modeling Language reference manual. Addison-Wesley Longman Ltd., Essex (1999)
6. OMG. UML2.0 Superstructure Specification. http://www.uml.org, Oct. (2005)
7. Seemann, J., von Gudenberg, J.W.: Extension of uml sequence diagrams for real-time systems. In: Proceedings of the First International Workshop on The Unified Modeling Language (UML '98), pp. 240–252. Springer, Berlin (1999)
8. Firley, T., Huhn, M., Diethers, K., Gehrke, T., Goltz, U.: Braunschweig TU. Timed sequence diagrams and tool-based analysis—a case study. In: Proceedings of the Second International Conference on UML (UML '99), pp. 645–660. Springer, Berlin (1999)
9. Debbabi, M., Hassaïne, F., Jarraya, Y., Soeanu, A.: Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models. Springer, Berlin (2010)

10. Peled, D.A.: Message sequence charts. In: Peled, D.A., Gries, D., Schneider, F.B. (eds.) Software Reliability Methods, pp. 300–305. Springer, Berlin (2001)
11. Li, X., Pan, M., Bu, L., Wang, L., Zhao, J.: Timing analysis of scenario-based specifications using linear programming. In: Software Testing, Verification and Reliability, vol. 22, no.2, pp. 121–143. Wiley InterScience, New York (2012)
12. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing (STOC '84), pp. 302–311. ACM, New York (1984)
13. Pan, M., Bu, L., Li, X., TASS.: Timing analyzer of scenario-based specifications. In: Proceedings of the 21th International Conference on Computer Aided Verification (CAV2009), pp. 689–695. Springer, Berlin (2009)
14. Eclipse—The Eclipse Foundation open source community website. http://www.eclipse.org/ (2011). Accessed 26 Sep 2011
15. TASS: Timing Analyzer of Scenario-based Specifications. http://seg.nju.edu.cn/TASS/ (2011). Accessed 26 Sep 2011
16. OR-Objects. OR-Objects/index.html. http://1997.opsresearch.com/ (2011). Accessed 26 Sep 2011
17. Clarke, E., Grumberg, J., Peled, D.: Model Checking. The MIT Press, Cambridge (2000)
18. Akshay, S., Bollig, B., Gastin, P.: Automata and logics for timed message sequence charts. In: Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '07), pp. 290–302. Springer, Berlin (2007)
19. Li, X., Lilius, J.: Timing analysis of uml sequence diagrams. In: Proceedings of the Second International Conference on UML (UML '99), pp. 661–674. Springer, Berlin (1999)
20. Li, X., Lilius, J.: Checking compositions of uml sequence diagrams for timing inconsistency. In: Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC '00), pp. 154–161. IEEE Computer Society, New York (2000)
21. Zheng, T., Khendek, F.: Time consistency of MSC-2000 specifications. In: Computer Networks, vol. 42(3), pp. 303–322. Elsevier, Amsterdam (2003)
22. Akshay, S., Gastin, P., Mukund, M., Narayan Kumar, K.: Model checking time-constrained scenario-based specifications. In: Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10), pp. 204–215. Schloss Dagstuhl, Washington, DC (2010)
23. Alur, R., David, D.: A theory of timed automata. In: Theoretical Computer Science, vol. 126(2), pp. 183–235. Elsevier, Amsterdam (1994)