1000-9825/2001/12(10)1423-11 ©2001 Journal of Software 软件学报 Vol. 12, No. 10

A Tool for Analyzing UML Sequence Diagrams with Timing Constraints*

TAN Wen-kai, LI Xuan-dong, ZHENG Guo-liang

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093, *China*)

E-mail: wktan@seg.nju.edu.cn

http://www.nju.edu.cn

Received April 10, 2000; accepted April 17, 2001

Abstract: The Unified Modeling Language (UML) is a general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of software systems. UML sequence diagrams describe a collaboration of interacting objects, where the interactions are exchanges among communicating entities in real-time and distributed systems. Like any other aspect of the specification and design process, the specifications in UML sequence diagrams are susceptible to errors, and their analysis is necessary. A tool for timing analysis of UML sequence diagrams is described in this paper.

Key words: real-time systems; UML (unified modeling language); sequence diagrams; timing constraints

The Unified Modeling Language (UML) is a general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of software systems[1,2]. UML provides a number of diagrams to describe particular aspects of software artifacts. Among these diagrams, UML sequence diagrams describe behavioral aspects of systems. They describe a collaboration of interacting objects, where the interactions are exchanges among communicating entities in real-time and distributed systems. Like any other aspect of the specification and design process, the specifications in UML sequence diagrams are susceptible to errors, and their analysis is necessary. In this paper, we describe a tool for timing analysis of UML sequence diagrams.

The paper is organised as follows. In next section, we introduce the UML sequence diagrams and the compositions of UML sequence diagrams. The design and implementation of our analysis tool is presented in Section 2. In Section 3, we use an example of an automatic teller machine system to illustrate the presented timing analysis. The last section is the conclusion.

1 UML Sequence Diagrams with Timing Constraints and Their Compositions

A UML sequence diagram describes an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result. Its focus is on the temporal order of the message flow. A

* Project is supported by the National Natural Science Foundation of China under Grant Nos. 69703009; 60073031 (国家自然科学基金); the National High Technology Development 863 Program of China under Grant No. 863-306-ZT06-04-4 (国家863高科技发展计划)

TAN Wen-kai was born 1976. He is a M. S. student in the Department of Computer Science and Technology, Nanjing University. His research area is model checking, formal verification and real time systems. **LI Xuan-dong** was born in 1963. He is a professor and doctoral supervisor of the Department of Computer Science and Technology, Nanjing University. His research area is formal verification and software engineering. **ZHENG Guo-liang** was born in 1937. He is a professor and doctoral supervisor of the Department of Computer Science and Technology, Nanjing University. His research area is software engineering.

UML sequence diagram has two dimensions: the vertical dimension represents time, and the horizontal dimension represents different objects. Each object is assigned a column, the messages are shown as horizontal, labeled arrows. Here we consider simple UML sequence diagrams which describe exactly one scenario without any alternatives and loops. For example, a simple UML diagram is depicted in Fig. 1.

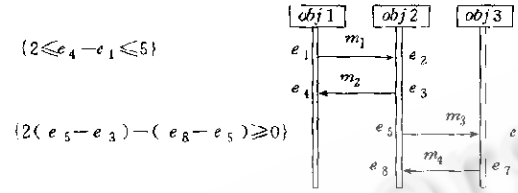


Fig. 1 A UML sequence diagram

1.1 UML sequence diagram with timing constraints

UML sequence diagrams only define the temporal order of the messages communicated in systems. They lack the precise timing constraints when messages are sent and received. In previous work[3~6], timing analysis of UML sequence diagrams and of message sequence charts has been restricted to timing constraints that just consist of time intervals on the occurrence time of events. Here we consider more general and expressive timing constraints.

By events we mean message sending, message receiving, creating an object, or deleting an object. Each event is given a name which represents its occurrence time. So, timing constraints can be described by boolean expressions on event names. Here we let any timing constraint be of the form

$$a \leqslant c_0(e_0 - e'_0) + c_1(e_1 - e'_1) + \ldots + c_n(e_n - e'_n) \leqslant b,$$

where $e_0, e'_0, e_1, e'_1, \ldots e_n, e'_n$ are event names, a, b and $c_0, c_1, \ldots, c_n$ are real numbers (b may be ∞). For analyzing UML sequence diagrams, we formalize UML sequence diagrams as follows.

Definition 1. A UML sequence diagram is a tuple $D=(O,E,V,C)$ where

O is a finite set of objects;

E is a finite set of events corresponding to sending a message, receiving a message, creating an object, and deleting an object;

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;

C is a set of boolean expressions, which represents the timing constraints enforced on D.

For example, the UML diagram depicted in Fig. 1 can be represented by the tuple (O,E,V,C) where

$O=\{obj_1, obj_2, obj_3\}$,

$E=\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$,

$V=\{(e_1,e_2),(e_2,e_3),(e_3,e_4),(e_4,e_5),(e_5,e_6),(e_5,e_7),(e_7,e_8)\}$, and

$C=\{2 \leqslant e_4 - e_1 \leqslant 5,\ 2(e_5 - e_3) - (e_8 - e_5) \geqslant 0\}$.

We use event sequences to represent the untimed behaviour of UML sequence diagrams. 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 $i (0 \leqslant i \leqslant m-1)$.

Definition 2. For any UML sequence diagram $D=(O,E,V,C)$, an event sequence $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$ is an untimed behaviour of D if and only if the following condition holds:

—all events in E occur in the sequence, and each event occurs only once; i.e. $\{e_0, e_1, \ldots, e_m\} = E$ and $e_i \neq e_j$ for any $i, j (i \neq j, 0 \leqslant i, j \leqslant m)$; and

—$e_0, e_1, \ldots, e_m$ satisfy the visual order defined by V, i.e. for any e_i and e_j, if $(e_i, e_j) \in V$, then $0 \leqslant i < j \leqslant m$.

We use timed event sequences to represent the behaviour of UML sequence diagrams. A timed event sequence is of the form $(e_0,t_0) \hat{} (e_1,t_1) \hat{} \ldots \hat{} (e_m,t_m)$ where e_i is an event and t_i is a nonnegative real number for any

$i(0\leqslant i\leqslant m)$, which describes that e_0 takes place t_0 time units after the system starts, then e_1 takes place t_1 time units after e_0 takes place, so on and so forth, at last e_m takes place t_m time units after e_{m-1} takes place. It follows that for any $i(0\leqslant i\leqslant m)$, the occurrence time of e_i is $\sum_{j=0}^{i} t_j$.

Let $\gamma(\sigma)=\sum_{i=0}^{m} t_i$ for any timed event sequence $\sigma=(e_0,t_0)^\frown(e_1,t_1)^\frown\ldots^\frown(e_m,t_m)$.

Definition 3. A timed event sequence $\sigma=(e_0,t_0)^\frown(e_1,t_1)^\frown\ldots^\frown(e_m,t_m)$ is a behaviour of a UML sequence diagram $D=(O,E,V,C)$ if and only if the following condition holds:

—$e_0^\frown e_1^\frown\ldots^\frown e_m$ represents an untimed behaviour of D, and

—$t_0,t_1,\ldots,t_m$ satisfy the timing constraints described by C, i.e. for any boolean expression $a\leqslant\sum_{i=0}^{n} c_i(f_i-f'_i)\leqslant b$ in C, $a\leqslant c_0\delta_0+c_1\delta_1+\ldots+c_n\delta_n\leqslant b$ where for each $i(0\leqslant i\leqslant n)$, if $f_i=e_j$

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

and $f'_i=e_k$, then

Let $L(D)$ denote the set of the timed event sequences representing the behaviour of D.

For describing timing constraints on the occurrence time of events, we introduce a special event ε which represents the start of system. For any UML sequence diagram $D=(O,E,V,C)$ such that $\varepsilon\in E$, any timed event sequence in $L(D)$ is of the form

$$(\varepsilon,0)^\frown(e_1,t_1)^\frown(e_2,t_2)^\frown\ldots^\frown(e_m,t_m).$$

1.2 Compositions of UML sequence diagrams

A simple UML sequence diagram describes exactly one scenario. For describing multiple scenarios and specifying real-time systems, we need to consider the compositions of UML sequence diagrams. At the moment the UML standard does not define compositions of UML sequence diagrams. Here we suggest to introduce high-level graphs, which are similar to High-Level MSCs defined in the message sequence chart standard[7], to describe compositions of UML sequence diagrams. For simplicity, "simple UML sequence diagram" is abbreviated to "SUD", while "composition of UML sequence diagrams" is abbreviated to "CUD".

The compositions of UML sequence diagrams can be described by a hierarchical graph. For example, Fig. 2 shows a composition of UML sequence diagrams, which describes a simple connection establishment protocol in a telecommunication system exampled in Ref. [4]. In Fig. 2, there are three simple UML diagrams (D_1,D_2,D_3) and a high-level graph (D_4) which describes the composition of these SUDs. D_1 describes a connection request, D_2 describes the successful establishment of the connection, and D_3 describes an unsuccessful establishment of the connection. obj_1 is a service provider, obj_2 is a local protocol machine, and obj_3 is a remote protocol machine. D_4 describes the composition of D_1,D_2, and D_3: the iterating branch describes a repeated request to establish the connection, while the non-iterating branch describes a successful connection establishment. For describing the timing constraints enforced on between two events in different UML sequence diagrams, a set of boolean expressions of the form $a\leqslant e-e'\leqslant b$ can be used as a complement.

Definition 4. A composition of UML sequence diagrams (CUD) is a tuple $S=(U,N,succ,ref,M)$ where

U is a finite set of SUDs satisfying that for any $D=(O,E,V,C)\in U$ and $D'=(O',E',V',C')\in U$, if $D\neq D'$, then $E\cap E'=\varnothing$;

$N=\{\top\}\cup I\cup\{\perp\}$ is a finite set of nodes partitioned into the three sets: singleton-set of start node, intermediate nodes, and singleton-set of end node, respectively;

$succ\subset N\times N$ is the relation which reflects the connectivity of the nodes in N such that any node in N is

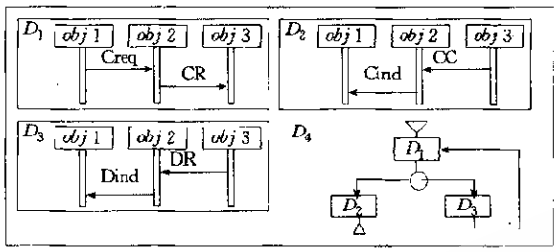


Fig. 2 A compositions of UML sequence diagrams

reachable from the start node and that the end node is reachable from any node in N;

$ref: I \mapsto B$ is a function that maps each intermediate node to a UML sequence diagram in U; and

M is a finite set of timing constraints of the form $a \leqslant e - e' \leqslant b$ where e and e' occur in different SUDs and $0 \leqslant a \leqslant b$ (b may be ∞).

For a CUD $S=(U,N,succ,ref,M)$, a path segment is a sequence of intermediate nodes $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_n$ satisfying that $(v_{i-1},v_i) \in succ$ for any $i(2 \leqslant i \leqslant n)$. A path segment is called simple if all its nodes are distinct. A path is a path segment $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_n$ such that $(\top, v_1) \in succ$ and $(v_n, \perp) \in succ$. A simple path is a path which is a simple path segment. For a simple path segment $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_n$, if there is $v_i (1 \leqslant i \leqslant n)$ such that $(v_n, v_i) \in succ$, then the sequence $v_i \hat{} v_{i+1} \hat{} \ldots \hat{} v_n$ is a loop and v_i is a loop-start node.

To avoid the ambiguity in interpreting timing constraints related to loops, timing constraints are not allowed to combine event occurrences inside and outside of a loop, i.e., for a CUD $S=(U,N,succ,ref,M)$, all timing constraints in M of the form $a \leqslant e - e' \leqslant b$ must satisfy the following condition:

—for any loop $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_m$, if e occurs in $ref(v_i)(1 \leqslant i \leqslant m)$ and e' does not occur in any $ref(v_j)(1 \leqslant j < i)$, then there is no simple path segment $v'_1 \hat{} v'_2 \hat{} \ldots \hat{} v'_n$ such that e' occurs in $ref(v'_i)$, e does not occur in any $ref(v'_k)(1 \leqslant k \leqslant n)$, and that $v'_n = v_1$;

—for any loop $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_m$, if e' occurs in $ref(v_i)(1 \leqslant i \leqslant m)$ and e does not occur in any $ref(v_j)(i < j \leqslant m)$, then there is no simple path segment $v'_1 \hat{} v'_2 \hat{} \ldots \hat{} v'_n$ such that $v_i = v'_1$, e occurs in $ref(v'_n)$, and that e' does not occur in any $ref(v'_k)$ $(1 \leqslant k \leqslant n)$; and

—for any loop $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_m$, if e occurs in $ref(v_i)$ and e' occurs in $ref(v_j)$, then $1 \leqslant j < i \leqslant m$.

We interpret the timing constraints in CUDs by local semantics: select one path at a time and analyze its timing requirements, independently of other paths that may branch out of the selected one. We define the behaviour of a CUD S as the timed event sequences which are the concatenation of the timed event sequences representing the behaviour of the UML sequence diagrams which make up S. We use $\hat{}$ to denote the concatenation of sequences.

Definition 5. For a CUD $S=(U,N,succ,ref,M)$, a timed event sequence $\sigma=(e_0,t_0) \hat{} (e_1,t_1) \hat{} \ldots \hat{} (e_n,t_n)$ represents a behaviour of S if and only if the following condition holds:

—there is a path $v_1 \hat{} v_2 \hat{} \ldots \hat{} v_m$ such that $\sigma = \sigma_1 \hat{} \sigma_2 \hat{} \ldots \hat{} \sigma_m$, where σ_i is a behaviour of $ref(v_i)$ for each i $(1 \leqslant i \leqslant m)$; and

—σ satisfies any timing constraint expressed by all boolean expressions in M, i.e., for any $a \leqslant (f - f') \leqslant b \in M$, for any i, $j(0 \leqslant i < j \leqslant n)$ such that $f' = e_i, f = e_j$, and that there is no k $(i < k < j)$ satisfying $f = e_k \vee f' = e_k$,

$$a \leqslant t_{i+1} + t_{i+2} + \ldots + t_j \leqslant b.$$

Definition 6. For any CUD $S=(U,N,succ,ref,M)$, for any path segment $\rho=v_1 \hat{} v_2 \hat{} \ldots \hat{} v_m$, let $L(\rho)$ be the set of all timed event sequences which are of the form $\sigma=(e_0,t_0) \hat{} (e_1,t_1) \hat{} \ldots \hat{} (e_n,t_n)$ and satisfy that

—$\sigma = \sigma_1 \hat{} \sigma_2 \hat{} \ldots \hat{} \sigma_m$, where σ_i is a behaviour of $ref(v_i)$ for each i $(1 \leqslant i \leqslant m)$; and

2.3 Algorithms in the tool

In the tool, we have implemented two algorithms corresponding to the two kinds of UML sequence diagrams.

2.3.1 Algorithm for checking timing consistency of simple UML sequence diagram

In section 1.1, we know a UML sequence diagram D is timing consistent if and only if $L(D)\neq\varnothing$. Let $D=(O,E,V,C)$ be a UML sequence diagram, $E=\{e_0,e_1,\ldots,e_m\}$, and t_i represent the occurrence time of e_i for any i $(0\leqslant i\leqslant m)$. By Definition 3, for any $(e-e')\in V$, $t_i-t_j\leqslant 0$ where $e=e_i$ and $e'=e_j$, and for any timing constraint in C

$$a\leqslant c_0(f_0-f'_0)+c_1(f_1-f'_1)+\ldots+c_n(f_n-f'_n)\leqslant b,$$

$t_0,t_1,\ldots,t_m$ must satisfy

$$a\leqslant c_0\delta_0+c_1\delta_1+\ldots+c_n\delta_n\leqslant b,$$

where for any i $(0\leqslant i\leqslant n)$, if $f_i=e_j$ and $f'_i=e_k$ then $\delta_i=t_j-t_k$, which form a group of linear inequalities on t_0, $t_1,\ldots,t_m$, denoted by $lp(D)$. Hence, the problem of checking D for timing consistency can be solved by checking whether the group $lp(D)$ of linear inequalities has no solution, which can be solved by linear programming.

2.3.2 Algorithm for checking timing consistency of compositions of UML sequence diagrams

For a CUD S, a path ρ is timing consistent if and only if $L(\rho)\neq\varnothing$. A CUD S is timing consistent if and only if all paths are timing consistent. Let $S=(U,N,succ,ref,M)$ be a CUD, and $\rho=v_1\hat{}\,v_2\hat{}\,\ldots\hat{}\,v_m$ be a finite path. Suppose that $ref(v_i)=(O',E',V',C')$ for any i $(1\leqslant i\leqslant m)$. From Definitions 3 and 6, it follows that all timed event sequence $\sigma\in L(\rho)$ are of the form $\sigma=\sigma_1\hat{}\,\sigma_2\hat{}\,\ldots\hat{}\,\sigma_m$ where $\sigma_i=(e_{i1},t_{i1})\hat{}\,(e_{i2},t_{i2})\hat{}\,\ldots\hat{}\,(e_{in_i},t_{in_i})$ for all i $(1\leqslant i\leqslant m)$. Furthermore, all t_{ij} $(1\leqslant i\leqslant m,\ 1\leqslant j\leqslant n_i)$ must satisfy all timing constraints in M and C_i $(1\leqslant i\leqslant m)$. This set of timing constraints forms a group of linear inequalities. Thus, we can check whether $L(\rho)=\varnothing$ by checking whether the group of linear inequalities has no solution, which can be solved by linear programming. So, for a finite path, we can reduce the problem into a linear programming problem. However we know that for a CUD, there could be infinite paths and the number of paths could be infinite. So we need a way to reduce the problem to finite set of finite paths.

Let $S=(U,N,succ,ref,M)$ be a CUD, and $\rho=v_1\hat{}\,v_2\hat{}\,\ldots\hat{}\,v_m$ be a loop. If the following condition holds:

—for any $ref(v_i)=(O',E',V',C')$ $(1\leqslant i\leqslant m)$, any timing constraints in C_i of the form $a\leqslant\sum_{j=0}^{n}c_j(e_j-e'_j)\leqslant b$ is such that $a\leqslant 0$ and $b\geqslant 0$; and

—for any timing constraint $a\leqslant e-e'\leqslant b\in M$, if e occurs in $ref(v_j)$ and e' occurs in $ref(v_k)$ $(1\leqslant k<j\leqslant m)$, then $a=0$ and $b\geqslant 0$;

then ρ is an unbounded loop; otherwise ρ is a bounded loop. Notice that for any loop ρ of the form $v_1\hat{}\,v_2\hat{}\,\ldots\hat{}\,v_m$,

—if ρ is an unbounded loop, then there is $\sigma\in L(\rho)$ such that $\sigma=(e_1,0)\hat{}\,(e_2,0)\hat{}\,\ldots\hat{}\,(e_n,0)$, i.e. there is $\sigma\in L(\rho)$ such that $\gamma(\sigma)=0$;

—if ρ is a bounded loop, then there is no $\sigma\in L(\rho)$ such that $\sigma=(e_1,0)\hat{}\,(e_2,0)\hat{}\,\ldots\hat{}\,(e_n,0)$, i.e. any $\sigma\in L(\rho)$ is such that $\gamma(\sigma)>0$.

Let $S=(U,N,succ,ref,M)$ be a CUD, and ρ be a bounded loop. For any timing constraints $a\leqslant e-e'\leqslant b$ in M where $b\neq\infty$, we say that it constrains ρ if and only if there is a path segment of the form

$$v_1\hat{}\,v_2\hat{}\,\ldots\hat{}\,v_{i-1}v_i\hat{}\,v_{i+1}\hat{}\,\ldots\hat{}\,v_m,$$

such that

ρ starts from v_i,

e' occurs in $ref(v_1)$ and e occurs in $ref(v_m)$,

e and e' do not occur in ρ and in any v_l $(2\leqslant l\leqslant m-1)$, and

all $v_j(1\leqslant j\leqslant i)$ are distinct and all $v_k(i\leqslant k\leqslant m)$ are distinct.

Theorem 1. A CUD $S=(U,N,succ,ref,M)$ is timing consistent if and only if the following condition holds:

—any loop ρ is such that $L(\rho)\neq\varnothing$,

—any simple path is timing consistent, and

—any timing constraint $a\leqslant e-e'\leqslant b$ in M such that $b\neq\infty$ dose not constrain any bounded loop.

Proof. The details of the proof are presented in Ref. [8]. □

```
Currentpath:=(|);loopset:=∅
    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 in currentpath then
                 begin
                   if the loop ρ is such that L(ρ)∅ return false
                   else put the loop into loopset;
                 end;
               if node≠⊥ and node is not in currentpath
               then append node to currentpath;
             end
    until currentpath=();
```

```
Currentpath:=(|);
    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,⊥)∈succ then
                 begin
                   if the path corresponding to currentpath is not timing consistent
                   then return false;
                 end;
               if node≠⊥ and node is not in currentpath
               then append node to currentpath;
             end
    until currentpath=();
```

```
check if any timing constraint a≤e-e'≤b in M such that b≠∞ does not constrain any bounded loop. The algorithm is shown in Fig. 5.
return true
```

Fig. 4 Algorithm for checking CUDs for timing consistency

Based on Theorem 1, for a given CUD $S=(U,N,succ,ref,M)$, we can give an algorithm to check its timing consistency (c. f. Figs. 4 and 5). The main data structures in the algorithm 2 include three lists of nodes *currentpath*, *currentpath*1, and *currentpath*2 which are used to record the current path and a set of loops which is used to record all loops. The algorithm consists of three steps. First, we find all loops and check if any loop ρ is such that $L(\rho)\neq\varnothing$. Then we traverse all simple paths and check if they are timing consistent. Last, we check if there are timing constraints $a\leqslant e-e'\leqslant b$ in M with $b\neq\infty$ that do not constrain any bounded loops. This step is done with a nested depth-first search. We start from the node that includes e' and look for a simple path segment ending at a loop-start node *node* from which a bounded loop excluding both e and e' starts. If we find one, we start a new

depth-first search to look for a simple path segment from *node* to a node that includes *e*.

```
for each timing constraint a≤e−e'≤b in M do
  begin
    node1:=the node including e': currentpath1:=⟨node⟩;
    repeat
        node:=the last node of currentpath1;
        if node has no new successive node
        then delete the last node of currentpath1
        else begin
              node:=a new successive node of node;
              if there is a bounded loop in loopset starting from node and
               e and e' do not occur in the loop
              then
                  currentpath2:=⟨node⟩
                  repeat
                    node:=the last node of currentpath2;
                    if node has no new successive node
                    then delete the last node of currentpath2
                    else begin
                          node:=a new successive node of node;
                          if e is in node then return false
                          if node≠⊥, e' is not in node, and
                            node is not in currentpath2;
                          end
                    until currentpath2=⟨⟩
              if node≠⊥,e is not in node, and node is not in currentpath1
              then append node to currentpath1;
    until currentpath1=⟨⟩;
  end
```

Fig. 5 Algorithm checking if any timing constraint does not constrain any bound loop

The algorithm is based on depth-first search. The space consumption is portional to the size of the longest path in the CUD. In the algorithm we need to solve a linear program when we check if a loop ρ is such that $L(\rho)\neq\varnothing$, and when we check if a path corresponding to *currentpath* is timing consistent. So the number of the linear programs we need to solve equals the number of all loops and simple paths in the CUD.

2.4 Implementation techniques

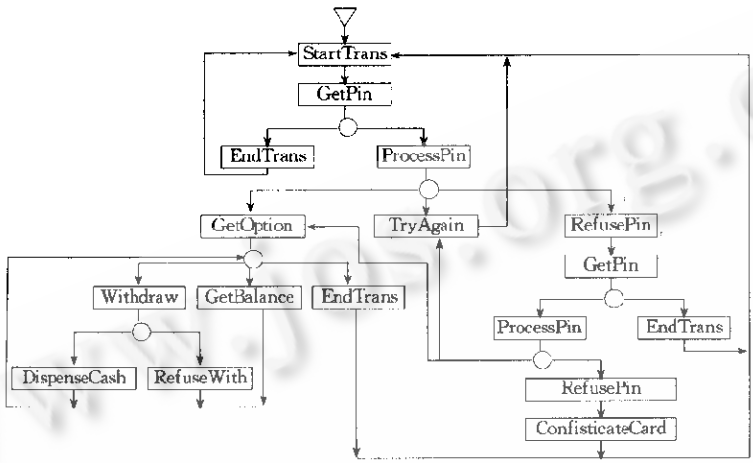
(1) Representation of events in sequence diagrams of Rational Rose. In the phase of making sequence diagrams into formal description defined as Definition 1, we need the time when messages are sent and received. Unfortunately, there is no such concept in sequence diagram of Rational Rose. Sequence diagrams only show the temporal order of messages happened, it doesn't concern the time when messages are sent and received. So we can't get event names needed in formalizing phase. We solve this problem by reusing the names of messages. In our implementation process, we use the name of a message to represent the event of sending the message and use the name of a message adding symbol' to represent the event of receiving the message.

(2) Analysis of representation of UML sequence diagrams in Rational Rose. Rational Rose has its own inner representation of the sequence diagram. We need draw the useful information from it. Fortunately, this inner representation is text format. We analyze its content and finally find the key words used to draw the tuple $D=(O,E,V,C)$ defined as Definition 1.

3 ATM Example

To illustrate the presented timing analysis, we consider an example of an automatic teller machine (ATM)

given in Ref. [4]. Figure 6 shows the high level graph of UML sequence diagrams and Fig. 7 shows the referenced SUDs of a CUD S describing the system. The ATM system consists of three components: potential customers that are represented by the object User, the ATM controller which is represented by the object ATM, and a host bank that is represented by the object Bank. Each one of the SUDs represents a scenario or 'use-case' of the system, and the CUD S specifies a successor relationship between the scenarios.



Constraints between two events in different UML sequence diagram:
$\{0 \leqslant \text{cancel}' \text{ req\_pin} \leqslant 4\}$, $\{5 \leqslant \text{enter\_pin}' - \text{req\_pin} \leqslant 60\}$,
$\{0 \leqslant \text{valid}' \text{-verify} \leqslant T_1\}$, $\{\text{return\_card-verify} \geqslant T_1\}$, $\{0 \leqslant \text{invalid}' \text{-verify} \leqslant T_1\}$,
$\{q_1 \leqslant \text{amt\_approved}' - \text{approve\_amt} \leqslant q_2\}$, $\{q_1 \leqslant \text{not\_approved}' \text{-approve\_amt} \leqslant q_2\}$,
$\{w_1 \leqslant \text{give\_money}' \text{-ent\_amount} \leqslant w_2\}$, $\{w_1 \leqslant \text{not\_possible}' \text{-ent\_mount} \leqslant w_2\}$

Fig. 6 High level graph of UML sequence diagrams for the ATM example

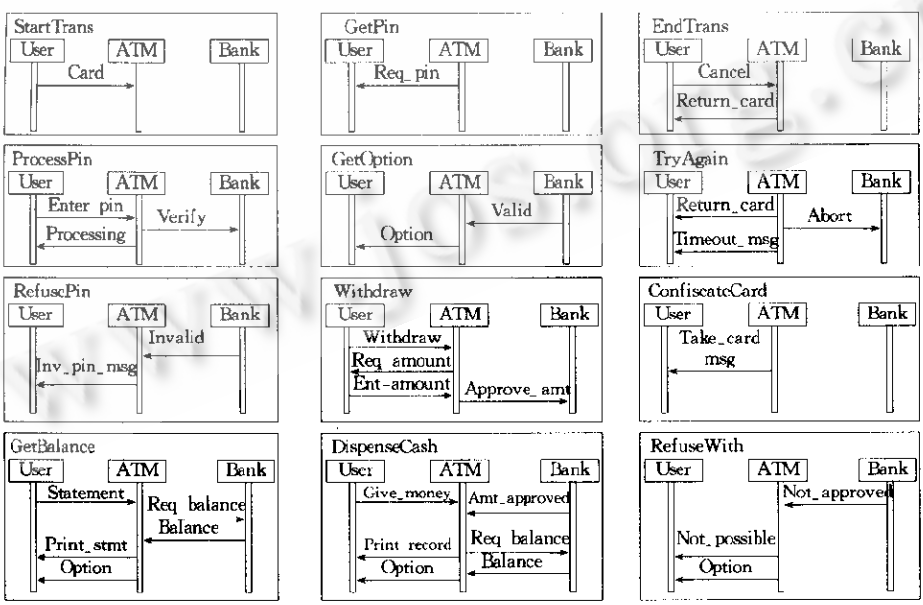
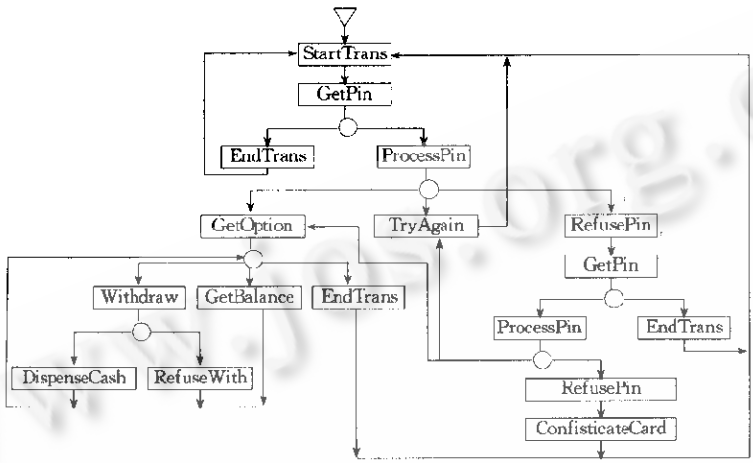


Fig. 7 SUDs in the ATM example

Initially, the ATM controller waits to receive a message that signals a customer has inserted his bank card. Once this message is received, the system then behaves in two possible ways: either the ATM controller receives a request to cancel the transaction within $[0,4]$ seconds (SUD EndTrans), or the ATM receives the customer's pin

given in Ref. [4]. Figure 6 shows the high level graph of UML sequence diagrams and Fig. 7 shows the referenced SUDs of a CUD S describing the system. The ATM system consists of three components: potential customers that are represented by the object User, the ATM controller which is represented by the object ATM, and a host bank that is represented by the object Bank. Each one of the SUDs represents a scenario or 'use-case' of the system, and the CUD S specifies a successor relationship between the scenarios.



Constraints between two events in different UML sequence diagram:
$\{0 \leqslant \text{cancel}' \text{ req\_pin} \leqslant 4\}$, $\{5 \leqslant \text{enter\_pin}' - \text{req\_pin} \leqslant 60\}$,
$\{0 \leqslant \text{valid}' \text{-verify} \leqslant T_1\}$, $\{\text{return\_card-verify} \geqslant T_1\}$, $\{0 \leqslant \text{invalid}' \text{-verify} \leqslant T_1\}$,
$\{q_1 \leqslant \text{amt\_approved}' - \text{approve\_amt} \leqslant q_2\}$, $\{q_1 \leqslant \text{not\_approved}' \text{-approve\_amt} \leqslant q_2\}$,
$\{w_1 \leqslant \text{give\_money}' \text{-ent\_amount} \leqslant w_2\}$, $\{w_1 \leqslant \text{not\_possible}' \text{-ent\_mount} \leqslant w_2\}$

Fig. 6 High level graph of UML sequence diagrams for the ATM example

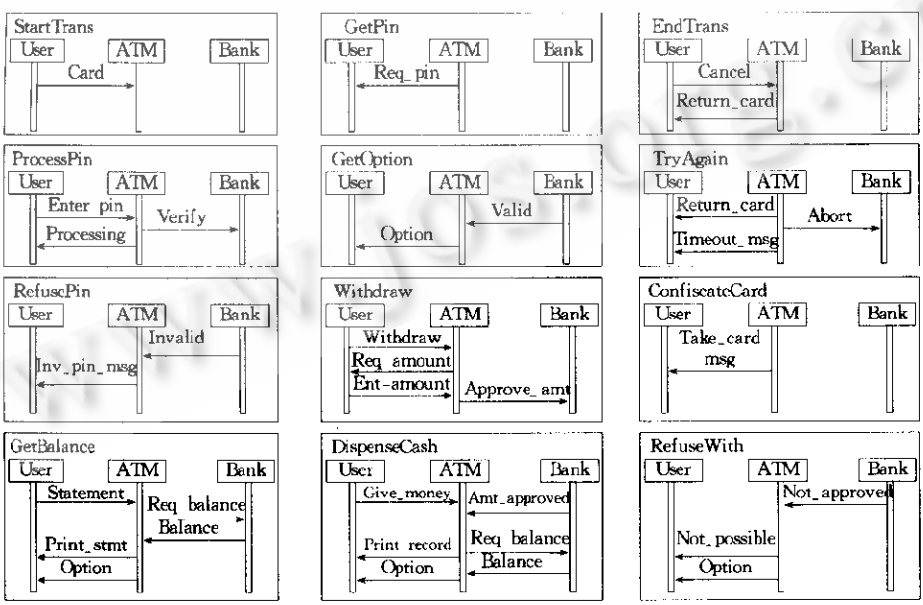


Fig. 7 SUDs in the ATM example

Initially, the ATM controller waits to receive a message that signals a customer has inserted his bank card. Once this message is received, the system then behaves in two possible ways: either the ATM controller receives a request to cancel the transaction within $[0,4]$ seconds (SUD EndTrans), or the ATM receives the customer's pin

number within [5,60] seconds (SUD ProcessPin), relative to the time the message 'Card' was received.

If the ATM receives a request to cancel the transaction (SUD EndTrans), it returns the customer's card and then returns to its initial state as described by the SUD StartTrans. If the ATM receives the customer's pin number, it sends a request to the bank to validate the pin number, signals the customer to wait as it is processing the request, and then waits for a reply from the bank. For performance reasons, the ATM constrains communication with the bank to take no longer than T_1 seconds. In this case, the timing constraints are described via the boolean expressions of $0 \leqslant$ valid'$-$verify$\leqslant T_1$, return_card-verify$\geqslant T_1$, and $0 \leqslant$ invalid'-verify$\leqslant T_1$ to corresponding the SUD of GetOption, TryAgain and RefusePin separately. The next behavior of the system is described as follows.

· In the SUD TryAgain, the ATM times-out on its wait for a validation reply from the bank. It therefore signals the customers to retry later, signals the bank to abort the validation request, returns the customer's card, and then returns to its initial state.

· In the SUD RefusePin, the ATM receives an invalid reply from the bank within the deadline. It signals the customer that the entered pin is invalid, and tries to get a new pin. In this CUD S, the customer tries to enter a valid pin at most twice, after which the ATM confiscates the customer's card-see the path from the SUD RefusePin to ConfiscateCard in Fig. 6.

· In the SUD GetOption, the ATM receives a valid reply from the bank within the deadline. It therefore signals the user a list of options. From this point on, the customer can make one or more withdrawals (SUD Withdraw), request one or more times their account balance (SUD GetBalance), or end the transaction (SUD EndTrans).

The ATM is expected to receive a message (either successfully as in the SUD DispenseCash, or unsuccessfully as in the SUD RefuseWith) from the Bank within $[q_1,q_2]$ seconds relative to the time it sends the request. The customer expects a withdrawal request to be processed (either successfully as in the SUD DispenseCash, or unsuccessfully as in the SUD RefuseWith) within $[w_1,w_2]$ seconds relative to the time they enter an amount.

In addition to the above timing constraints between two events in different UML sequence diagrams, we assume that each communication among User, ATM, and Bank has a delay of [1,2] seconds, and that each vertical line has a default delay of $[0,\infty]$. For simplicity, these timing constraints are not represented in Fig. 7.

We used our analysis tool to verify automatically the timing consistency of the CUD for $T_1=10$ and various values of q_1, q_2, w_1 and w_2. Table 1 shows sample results.

Table 1 Sample results of timing consistency analysis

Case #	(1)	(2)	(3)	(4)	(5)
$[q_1,q_2]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	[6,10]	[6,10]
$[w_1,w_2]$	$[0,\infty]$	[0,3]	[0,4]	[0,7]	[0,8]
Consistent?	Yes	No	Yes	No	Yes

For the case (1) in Table 1, the user does not impose any timing constraints on the system. This case in fact makes any value acceptable for the remaining variables. In the case (2), the user expects the ATM to process their withdrawal request within [0,3] seconds; such a deadline leads to timing inconsistencies in all sequential components that contain the SUD Withdraw followed by either DispenseCash or RefuseWith. In the case (3), the user loosens the time of ATM response to event "ent_amount" to [0,4], then the system is timing consistent. In the case (4) and (5), we examine the affects of the Bank response time. If the Bank needs [6,10] seconds to respond the ATM request 'approve_amt', then the user should loosen the time of ATM response to event 'ent_amount' more. In case (4), even if we loosen the time to [0,7], the system is still timing inconsistent. In case (5), when we loosen the time to [0,8], the system is timing consistent.

4 Conclusion

In this paper, we have presented the design and implementation of a tool for checking UML sequence diagrams for timing consistency. In Ref. [3], some algorithms for analyzing message sequence charts with interval delays are presented, and a corresponding tool is described. In Refs. [4,5], this timing analysis is extended to MSC specifications, which are compositions of message sequence charts. However, the problem of analyzing MSC specifications for timing consistency is not solved completely there because a sufficient condition for timing consistency is given, which is not enough to develop an algorithm to analyze MSC specifications for timing consistency. In Ref. [6], UML sequence diagrams are extended for real-time systems with loops, and timing consistency analysis is considered: timing constraints are of simple form $a \leqslant e - e' \leqslant b$, and are not over any loop. Instead in this paper more general and expressive timing constraints are considered, and timing constraints are allowed to be over loops so that the compositions of UML sequence diagrams are much more complicated to check.

An important topic for the future work is doing case studies in practical use with the developed tool. In the longer term, our intention is to extend the tool for analyzing more UML diagrams.

References:

[1] Grady, B., Rumbaugh, J., Ivarm, J. The Unified Modeling Language User Guide. Addison-Wesley, 1998.

[2] Rumbaugh, J., Jacoson, I., Booch, G. The Unified Modeling Language Reference Guide. Addison-Wesley, 1998.

[3] Alur, R., Holzmann, G. J., Peled, D. An analyzer for message sequence charts. Software-Concepts and Tools, 1996,17: 70~77.

[4] Ben-Abdallah, H., Leue, S. Expressing and analyzing timing constraints in message sequence char: specifications. Technical Report 97-04, Department of Electrical & Computer Engineering, University of Waterloo, 1997.

[5] Ben-Abdallah, H., Leue, S. Timing constraints in message sequence chart specifications. In: Formal Description Techniques X, Proceedings of the 10th International Conference on Formal Description Techniques FORTE/PSTV'97. Osaka, Japan, Chapman & Hall, 1997.

[6] Seemann, J., WvG, J. Extension of UML sequence diagrams for real-time systems. In: Proceedings International UML Workshop. Lecture Notes in Computer Science, Springer, 1998.

[7] ITU-T. Recommendation Z.120. ITU-Telecommunication Standardization Sector, Geneva, Switzerland, 1996.

[8] Li, Xuan-dong, Johan, Lilius. Timing Analysis of UML Sequence Diagrams. In: 《UML》'99-The Unified Modeling Language. Lecture Notes in Computer Science 1723, Springer, 1999. 661~674.

带时间约束的 UML 序列图的分析工具

谭文凯, 李宣东, 郑国梁

(南京大学 计算机科学与技术系,江苏 南京 210093)

摘要: 统一建模语言(UML)是一种多用途的可视化建模语言,可用于软件系统的规约、可视化的构造和建档. UML 序列图描述了交互对象间的协作,如在实时和分布式系统中通讯实体间的信息交互.与其他规约和设计过程类似,UML 序列图的规约也易出错,所以对它进行分析是很有必要的.描述了一个对带时间约束的 UML 序列图进行分析的工具.

关键词: 实时系统;UML;序列图;时间一致性

中图法分类号: TP311 **文献标识码:** A