Using *TIMED*TTCN-3 in Interoperability Testing for Real-time Communication Systems

Zhiliang Wang[1], Jianping Wu[1], Xia Yin[1], Xingang Shi[2] and Beihang Tian[1]

[1] Department of Computer Science and Technology, Tsinghua University
Beijing, P. R. China, 100084
{wzl, yxia, tbh}@csnet1.cs.tsinghua.edu.cn, jianping@cernet.edu.cn
[2] Network Research Center, Tsinghua University,
Beijing, P. R. China, 100084
shixg@cernet.edu.cn

Abstract. Interoperability testing is an important technique to ensure the quality of implementations of network communication software, and real-time protocol interoperability testing is an important issue in this area. *TIMED*TTCN-3 is a real-time extension of test specification language TTCN-3. In this paper, test notations for real-time interoperability testing are studied. Test behavior trees are constructed from specifications of system under test and then transformed to *TIMED*TTCN-3 test cases. We also investigate real-time TTCN and analyze the insufficiency of its capabilities in specifying time constraints. Possible extensions for real-time TTCN are given to specify real-time interoperability test cases. From the comparisons between the two real-time test notations, it can be concluded that *TIMED*TTCN-3 is more powerful and flexible than real-time TTCN and can be suitable for real-time interoperability testing.

1 Introduction

In order to ensure the quality of communication software, protocol test techniques are widely used. Conformance testing is the basic method of protocol testing, which can be used to test whether an implementation conforms to its protocol specification. As the complement of conformance testing, interoperability testing is often used to test whether two or more protocol implementations can communicate with each other correctly and inter-operate as a whole system to perform functions specified in protocol specifications. Interoperability testing is necessary because (1) It is difficult to perform exhaustive conformance testing, that is, a conformance test suite can hardly ensure 100% test coverage; (2) Many optional features may be contained in network protocols, and moreover vendors perhaps have their own extensions, so if two implementations implement different options, problems on interoperability will happen. Interoperability testing is also being performed by IETF and ETSI in the process of protocol design ([KD03]). Interoperability testing events have been organized by these organizations.

In the area of interoperability testing, [Hao97] proposed a TTCN-2 based framework for interoperability testing, [VBT01] presented a formal framework for interop-

erability testing and several interoperability relations were defined to guide test generation. Interoperability test generation is an important issue in this field. In most works of test generation, the basic idea is to model the interoperability system under test as a system of communicating finite state machines and generate test sequences for the composition of these machines ([RC90]). Based on this idea, a series of test generation techniques have been proposed ([KSK00, SKKJ03, TKS03, SKCK04, ETSY04]). Different from the above literatures, [HLSG04] proposed an efficient method only considering the specification of one protocol entity. By this method, there is no need to generate the composition machine and the state space explosion can be alleviated.

But in most real-life network protocols, not only the behaviors of input and output, but also their time of occurrence should be considered, that is, such protocols can be modeled by real-time systems. In order to test real-time systems, we should check if the I/O behaviors act under the specified time constraints. In the field of real-time testing, many methods of conformance testing have been proposed. In most of these works, Timed Automaton [AD94] or its variants have been used to specify real-time system. [SVD01, EDK02] converted timed automaton to grid automaton, and applied existing test generation methods for finite state machine (FSM) to it. But this method suffers from the state space explosion problem. [HNTC99] presented a test generation method of executability decision. [KJM03, KT04, LMN04, BB04] defined timed conformance relations, and proposed associated test generation methods. As far as we know, [WWY04] is the first work to study interoperability testing of real-time systems. An interoperability test generation method of time dependent protocols was presented in [WWY04].

In this paper, we focus on the problems of test notations suitable for interoperability testing of real-time communication systems, i.e., how to specify test cases. The *testing and test control notation version 3*(TTCN-3) ([TTCN3, GHRS03]) is a new test specification language standardized by ETSI (*European Telecommunications Standards Institute*), which is a new version and redesign of TTCN (*tree and tabular combined notation*) ([TTCN]). *TIMED*TTCN-3 ([DGN02]) is a real-time extension of TTCN-3. [DGN03] presented a method of generating *TIMED*TTCN-3 code from MSC test specifications; and [NDG04] used *TIMED*TTCN-3 in specifying real-time communication patterns. In this paper, we intend to use *TIMED*TTCN-3 in real-time interoperability testing. Following the test generation method presented in [WWY06], a parameterized test behavior tree will be generated from the formal model of system under test (SUT). Parameters in the test behavior tree are relative time intervals between IO events. In this paper, firstly the test behavior tree will be converted from the view of SUT to the view of test system, which is an intermediate notation of test cases. Then we give transformation rules to transform such a test behavior tree to a *TIMED*TTCN-3 test case.

We also investigate the real-time extension of TTCN – real-time TTCN ([WG99]) and intend to use real-time TTCN to specify test cases. But unfortunately, we find that real-time TTCN has no enough capabilities to specify timed interoperability test cases. We also give possible extensions of real-time TTCN on the syntactical and semantic levels to specify test cases. Based on the transformation results to

*TIMED*TTCN-3 and extended real-time TTCN test cases, we compare the two test notations mainly on the capabilities of specifying *hard* real-time requirements.

The rest of the paper is structured as follows. Section 2 gives the formal model Communicating Multi-port TIOA (CMpTIOA) to specify interoperability system under test; and as a working example, a simple real-time communication protocol system is specified by using this model. In Section 3, test architecture is given and test behavior trees will be generated. In Section 4, we give transformation rules from test behavior trees to *TIMED*TTCN-3 test cases. In Section 5, we investigate real-time TTCN and draw a comparisons between *TIMED*TTCN-3 and real-time TTCN. Conclusion and future work are given in Section 6.

2 Preliminaries

2.1 Multi-port TIOA

Timed Automaton ([AD94]) is a widely-used model of real-time system. TIOA (*Timed Input Output Automata* [EDK02]) is a variant of Timed Automaton, which distinguishes whether an action is an input or output. To specify an entity interacting with more than one other entities, we extend TIOA to Multi-port TIOA as follows.

Definition 1. *Multi-port Timed Input Output Automaton* (*MpTIOA*)

A Timed Input Output Automaton with n ports (for short, np-TIOA) is a 6-tuple *(L, I, O, l^0, C, T)*, where,

- L is a finite set of locations;
- It has n ports communicating with environment, which are denoted as P_1, P_2, …, P_n respectively;
- I is an n-tuple: $I=(I_1, I_2, ..., I_n)$, where $I_k(k=1,2,...,n)$ is the set of input action symbols of port P_k; $\overline{I}=I_1 \cup I_2 \cup \cdots \cup I_n$ is the set of input action symbols; An input action symbol occurring in port P_k can be denoted as $P_k?a$ ($a \in I_k$);
- O is an n-tuple: $O=(O_1, O_2, ...O_n)$, where $O_k(k=1,2,...,n)$ is the set of output action symbols of port P_k; $\overline{O}=O_1 \cup O_2 \cup \cdots \cup O_n$ is the set of output action symbols. An output action symbol occurring in port P_k can be denoted as $P_k!b$ ($b \in O_k$);
- $l^0 \in L$ is the initial location;
- C is a finite set of clocks $\{t_1, t_2, ..., t_{|C|}\}$, where, $|C|$ is the number of clocks;. $v_i \in \boldsymbol{R}^+$(non-negative real numbers) is the clock value of t_i; $\vec{v}=(v_1, v_2, \cdots, v_{|C|})$ denotes a clock valuation;
- T is a set of transitions: $(l,a,P,R,l') \in T$, where, $l,l' \in L$ are the source and destination locations; $a \in \overline{I} \cup \overline{O}$ is an input or output action symbol; P is the time constraint, which is a Boolean conjunction over linear inequalities $P(\vec{v})$; The subset $R \subseteq C$ specifies the clocks to be reset to 0. The transition $(l,a,P,R,l') \in T$ can be also denoted as $l \xrightarrow{a[P]/R} l'$; □

In the model, we assume that time constraints of transitions are all the format of $\wedge(v_i \sim d)$, where $\sim \in \{<, >, \leq, \geq, =\}$, and $d \in \boldsymbol{R}^+$. We distinguish two urgency types ([BST98]) of transitions implicitly: (1) ***Lazy***, for transitions with input actions, means that input actions may be not taken because they are controlled by environment (such a property is also called "***Unforced Inputs***"); and (2) ***Delayable***, for transitions with output actions, means that the corresponding output action must be taken during such transitions' enabling time.

The semantics of MpTIOA can be defined as a TIOTS (*Timed Input Output Transition System*) $(S, s_0, A_{in}, A_{out}, \rightarrow)$, where $A_{in} = \overline{I}$ and $A_{out} = \overline{O}$. We denote $Act = A_{in} \bigcup A_{out}$ as the set of all IO symbols. Its states are the pairs $s = (l, \vec{v})$, where $l \in L$ is a location, $\vec{v} = (v_1, v_2, \cdots, v_{|C|})$ is a clock valuation. S is the set of all possible states. $\rightarrow \subseteq S \times (Act \bigcup R^+) \times S$ is the set of transitions. There are two types of transitions: ***Timed transitions*** and ***Discrete transitions. Timed transitions*** model time progress, which are the form $(l, \vec{v}) \xrightarrow{d} (l, \vec{v}')$, where $d \in R^+$ is the delaying time, $\vec{v}' = \vec{v} + \vec{d} = \vec{v} + (d, d, \cdots, d)$, and in this period, no discrete transitions occur. ***Discrete transitions*** $(l, \vec{v}) \xrightarrow{a} (l', \vec{v}')$ correspond to execution of the transition (l,a,P,R,l') in MpTIOA, where P is satisfied by $\vec{v}$ ($P(\vec{v})$=***true***) and $\vec{v}'$ is obtained by updating $\vec{v}$ according to R.

2.2 Communicating Multi-port TIOA

To specify an interoperability system under test including two or more entities, we introduce a formal model Communicating MpTIOA (CMpTIOA). In the model, MpTIOA can model each single entity, and all these entities in the system can communicate with each others via channels between different MpTIOAs.

Definition 2. *Communicating Multi-port Timed Input Output Automata* (*CMpTIOA*)

A Communicating MpTIOA is composed of a set of MpTIOAs M and a set of channels Ch, where,

(1) $M=\{M_1, M_2, \ldots, M_m\}$ is a finite set of m MpTIOAs;

(2) $Ch = \{C_{ij} \mid i, j = 1, 2, \ldots, m \square\ i \neq j\}$ is a finite set of channels between MpTIOAs: $C_{ij} \square Ch$ represents the communicating channel from MpTIOA M_i to M_j. □

In the definition of CMpTIOA, channels behave like FIFO queues. Intuitively, the semantic of channels is that outputs of MpTIOA M_i can be transferred via channel C_{ij} to be inputs of M_j. In this paper, we assume that transfer time of actions in communicating channels can be neglected, that is, the channels are **lossless** and **non-delayed**.

Definition 3. *Port Mapping Relations of CMpTIOA*

Port Mapping Relations R of CMpTIOA M is an m-tuple: $R=(R_1, R_2, \ldots, R_m)$, where $R_k(k=1,2,\ldots,m)$ is the Port Mapping Relations of MpTIOA M_k; R_k is a set of Port Mapping Relations for all ports of M_k: $R_k=\{r_1, r_2, \ldots, r_n\}$, where n is the port number of M_k, and $r_i(i=1,2,\ldots, n)$ can be the format of 1) P_i -> M_j:$P_h(j \neq k)$, which means that the port P_i of M_k is connected to the port P_h of M_j via the channel C_{kj}; 2) P_i -> *env*,

which means that the port P_i of M_k is connected to the external environment of the system. □

According to the above definitions, we can get the abstract topology of the system under test. We furthermore denote the ports communicating with the external environment as "external ports"; and others as "internal ports". Inputs/outputs on external/internal ports are "external/internal inputs/outputs".

2.3 A simple real-time communication protocol

We specify a simple real-time communication protocol by using MpTIOA. Fig. 1 (a) shows the specification of such a protocol, which is a 2p-TIOA with two ports (*U* and *l*) and two clocks $\{t_1, t_2\}$. $I_U=\{A\}$, $O_U=\{B,C\}$, $I_l=O_l=\{a,b,c\}$. The initial location is '0'. The protocol can be specified informally as follows:

(1) Initiate a connection to a remote entity actively:

If an input 'A' is received from port *U* in the initial location 0, the protocol entity should initiate a connection to a remote entity actively; In this transition $(0,U?A,\textbf{\textit{true}},\{t_1,t_2\},1)$, the two local clocks t_1 and t_2 should be reset to 0. Within 2 time units, an output 'a' should be sent from port *l* to remote entity, and the clock t_1 should be reset to 0 (transition $(1,l!a,[t_1<2],\{t_1\},2)$). After that, three cases should be considered:

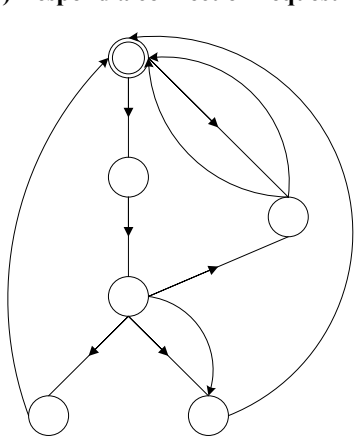
a) Receiving an input 'b' from port *l* in time, i.e., transition $(2,l?b,[t_1\geq 1,t_2<2],\{\},3)$, indicates that the connection can be established;

b) Receiving an input 'b' from port *l* too late, i.e., transition $(2,l?b,[t_2\geq 2],\{t_1\},4)$, indicates that the connection cannot be established;

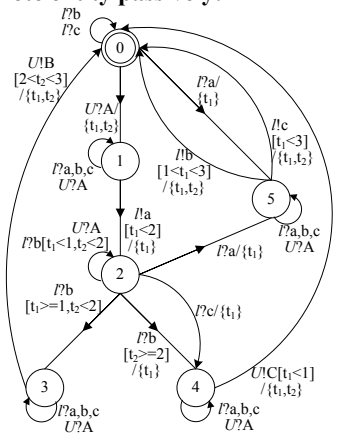
c) Receiving an input 'c' from port *l*, i.e., transition $(2,l?c,\textbf{\textit{true}},\{t_1\},4)$, indicates that the connection cannot be established.

If the connection can be established, an output 'B' should be sent to port *U*, i.e., transition $(3,U!B,[2<t_2<3],\{\},0)$; else, an output 'C' should be sent to port *U*, i.e., transition $(4,U!C,[t_1<1],\{\},0)$.

(2) Respond a connection request from a remote entity passively:



(a) Specification of a MpTIOA



(b) an input-complete specification

Fig. 1. A simple real-time protocol specified by using MpTIOA

a) Sending an output 'b' to port *l*, i.e., transition (5,*l*!*b*,[1<t_1<3],{},0);
b) Sending an output 'c' to port *l*, i.e., transition (5,*l*!*c*,[t_1<3],{},0).

To test interoperability, we make an assumption that both specifications and implementations are input-complete, that is, they can accept any inputs at any locations. To make a specification input-complete, some self-loop transitions can be added to it, which indicates that a specification ignores the unspecified input actions. Fig. 1(b) shows an input-complete specification after adding self-loops to Fig. 1(a).

Fig. 2 shows an example of a system under test specified by CMpTIOA, which is a real-time communication system containing two real-time protocol entities. M={M_1, M_2}, Ch={C_{12}, C_{21}}. The specifications of M_1 and M_2 are both the MpTIOA of Fig. 1(b). We use subscript 1, 2 on ports and actions to distinguish them. Port Mapping Relations are ({U_1->env, l_1->M_2:l_2}, {U_2->env, l_2->M_1:l_1}).

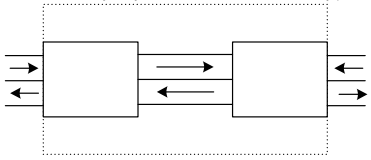


Fig. 2. An example of CMpTIOA

3 Test Behavior Tree

3.1 Test architecture

To test interoperability of protocol system, test architecture should be defined firstly. Fig. 3 shows test architecture that can be used to test SUT in Fig. 2. In the test architecture, there are two types of access points to SUT in the test system: PCO (Point of Control and Observation) and PO (Point of Observation). PCOs have capabilities of control and observation, which can either apply stimuli to or receive responses from SUT; and POs only have capabilities of monitoring the interactions of SUT. In Fig. 3, PCO1 and PCO2 are connected to the external ports U_1 and U_2 respectively, and only one PO is contained in test architecture to monitor the IO behaviors in channel C_{12} and C_{21}.

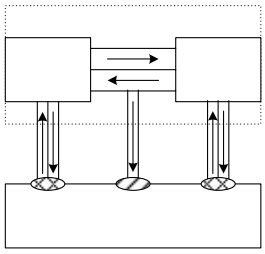


Fig. 3. Test architecture used to test SUT in Fig.2

3.2 Generating test behavior tree

In [WWY06], based on timed interoperability relations, a test generation method was presented. This method starts from the formal model of SUT, and as a result, a parameterized test behavior tree can be generated. Such a test behavior tree is just an intermediate notation. In this paper, we do not intend to introduce this method in detail.

Fig. 4 is a part of resulting parameterized test behavior tree. Leaf nodes of a test behavior tree are the verdict "**pass**" or "**fail**". For "**fail**" verdict, it is also necessary to indicate which implementation the fault is located in. The other internal nodes represent the tester's knowledge of the SUT's current global states, denoted as (s^1, s^2, ..., s^m), where $s^i = (l^i, \vec{v}^i)(i = 1,2,\ldots,m)$, representing local states of M_i. The root node is the initial global state $\boldsymbol{GS_0} = (s_0^1, s_0^2, \ldots, s_0^m)$ of SUT, where $s_0^i = (l_0^i, \vec{v}_0^i)(i = 1,2,\ldots,m)$. Edges between nodes are labeled as possible input/output events and their time constraints in SUT. Parameters $d_i(i=0,1,2,\ldots)$ in a test behavior tree represent relative time intervals between the two consecutive IO events. There are two types of parameters: **controllable parameters** are time intervals between an external input event and its last IO event in the tree, and their values should be set in test cases in advance, so such parameters are controllable for test system, e.g., d_0 in Fig. 4; **uncontrollable parameters** are time intervals between an internal or external output event and its last IO event, and their values are dependent on SUT and only can be retrieved on the process of test execution and cannot be set in advance, so such parameters are uncontrollable for test system, e.g., d_1, d_2, d_3 in Fig. 4.

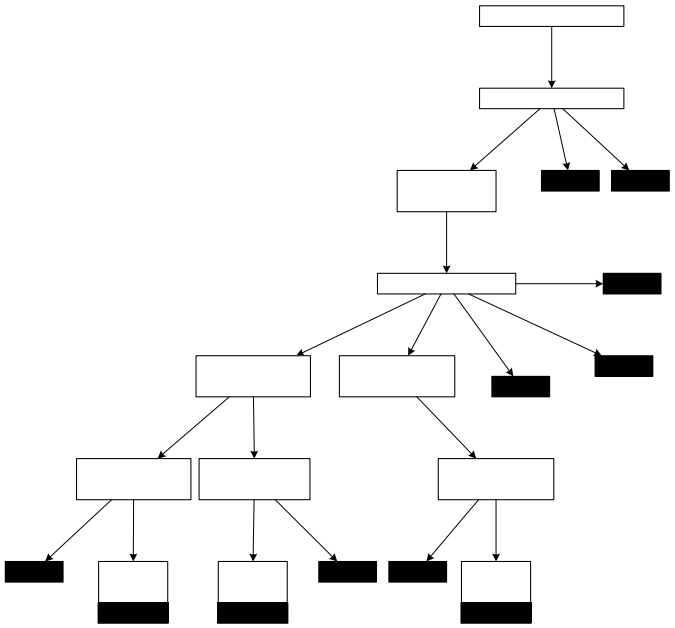


Fig. 4. A part of parameterized test behavior tree (from the view of SUT)

Parameterized test behavior tree in Fig. 4 is described from the view of SUT. To generate executable test case, at first, it should be converted to the test behavior tree which is described from the view of test system. The basic idea is to convert edges of the original tree to nodes of the resulting tree, and associate each IO event with one access point of test system; e.g., for test architecture of Fig. 3, IO events on the port U_1 of M_1 are associated with PCO1, events on U_2 of M_2 are associated with PCO2 and events on the two internal ports l_1 and l_2 are all associated with PO. On PCO1 and PCO2, input/output actions of SUT should be converted to sending/receiving test events of test system. On PO, all actions should be converted to receiving test events of test system.

Fig. 5 shows the resulting test behavior tree described from the view of test system: the root node represents the start point of the test; other internal nodes are labeled as test events and their time constraints; black leaf nodes represent pass verdicts, and gray leaf nodes represent fail verdicts. In the timing axis on the right of the tree, the global time for the same level of test events occurring are denoted as $T_i(i=0,1,2,\ldots)$, so relative time intervals between two consecutive events are $d_i=T_{i+1}-T_i\,(i=0,1,2,\ldots)$.

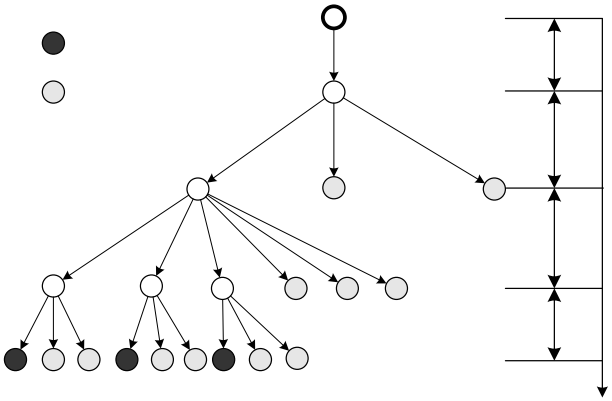


Fig. 5. Test behavior tree (from the view of test system)

We have proved in [WWY06], all time constraints on each test event can be represented by a conjunction over a set of linear inequalities on parameters (see **Lemma 1**), e.g., on the time point T_3, time constraints of node (1) are [$1<d_2<3$ and $d_2\geq 1$ and $d_1+d_2<2$] ([$1<d_2<3$ and $d_1+d_2<2$]).

Lemma 1. On the time point T_k, time constraints of the test event nodes can be represented by a conjunction over a set of linear inequalities on $d_i(i=0,1,\ldots,k-1)$. □

4 Transformation to *TIMED*TTCN-3 Codes

4.1 *TIMED*TTCN-3

*TIMED*TTCN-3 ([DGN02]) is a real-time extension of TTCN-3([TTCN3]). In *TIMED*TTCN-3, the concept of absolute time is introduced, so *TIMED*TTCN-3 provides a capability of testing *hard* real-time requirements. *TIMED*TTCN-3 (1) introduces a new verdict **conf** to indicate *functional pass* but *no-functional fail*; (2) introduces the concept of absolute time and provides mechanisms of retrieving the current local time and delaying the execution of a test component; (3) extends the TTCN-3 logging mechanism; (4) supports both online and offline evaluation.

(1

4.2 Transformation to *TIMED*TTCN-3 test cases

Now we consider how to convert a test behavior tree to a *TIMED*TTCN-3 test case. In this paper, for the sake of simplicity, only test architecture with one single main test component is considered. Firstly, not considering time constraints, a test behavior tree can be converted to a TTCN-3 test case easily: **Pre-order traversing method** can be used to covert a test behavior tree to dynamic behaviors of a TTCN-3 test case. In a test behavior tree, each node of sending event corresponds to a TTCN-3 statement of **send** operation, and each node of receiving event corresponds to a TTCN-3 statement of **receive** operation. The nodes of the same level in a test behavior tree can be represented by using **alt** statements of TTCN-3, which are called a set of *alternatives*. When reaching a leaf node of the test behavior tree, a verdict should be set by using a **setverdict** statement, and the test case will be stopped.

Now we consider time constraints in test behavior trees and give transformation rules from test behavior trees to *TIMED*TTCN-3 test cases.

(1) Get global time values of nodes in the test behavior tree:

In *TIMED*TTCN-3, the concept of absolute time is introduced. To get global time values of nodes, e.g., T_0, T_1, T_2 … in Fig. 5, **now** operations in *TIMED*TTCN-3 can be used. The **now** statements should be placed directly after the associated statements of **receive** operations that correspond to nodes of receiving events. For example, in Table 1, for the **receive** operation of line 9 corresponding to node (b) or (c) in Fig. 5, a **now** statement is placed directly in line 10 to get the global time and store it in a **float** variable T2.

(2) Get the real values of uncontrollable parameters:

In the test behavior tree, uncontrollable parameters are time intervals between an internal or external output event and its last IO event, which can be calculated only in the process of test execution. In *TIMED*TTCN-3, **assignment** statements can be used to get the real values of such parameters. These values can be calculated by expressions on global time values, in fact, $d_i=T_{i+1}-T_i$ (i=0,1,2,…). For example, an uncontrollable parameter d_1 can be calculated by an **assignment** statement in line 11.

Table 1 the *TIMED*TTCN-3 test case for the test behavior tree in Fig. 5

```
1  testcase testcase1() runs on simple_rtp {
2    var float T0,T1,T2,T3,T4; //global clock
3    var float d1,d2,d3; //time interval
4    T0 := self.now;
5    T1 := T0 + 1;
6    resume(T1) ;  //wait until T1 point
7    PCO1.send(A1);  //node (a)
8    alt {   //alt 1
9      []PO.receive(a) {
10       T2 := self.now;
11       d1 := T2 – T1;
12       if(d1 < 2) {     //node (b)
13         alt {   //alt 2
14           []PO.receive(b) {
15             T3 := self.now;
42           }
43           []PCO1.receive{ //node(12)
44             setverdict(fail);
45           }
46         }  //end of alt 4
47       } else { //node(4)
48         setverdict(conf);
49       }
50     }
51     []PO.receive(c) {
52       T3 := self.now;
53       d2 := T3 - T2;
54       if(d2 < 3) { //node (3)
55         alt {   //alt 5
56           [] PCO1.receive(C1) {
```

```
16        d2 := T3 – T2;
17        if((d2>1) and (d2<3)
            and ((d1+d2)<2)){  //node(1)
18          alt {   //alt 3
19            []PCO1.receive(B1) {
20              T4 := self.now;
21              d3 := T4 – T3;
22              if( ((d1+d2+d3)>2) and
                  ((d1+d2+d3)<3)){//node(7)
23                setverdict(pass);
24              } else {    //node (8)
25                setverdict(conf);
26              }
27            }
28            []PCO1.receive { //node(9)
29              setverdict(fail);
30            }
31          }  //end of alt 3
32        } else if((d2>1) and (d2<3)
            and ((d1+d2)>=2)){ //node(2)
33          alt {    //alt 4
34            []PCO1.receive(C1) {
35              T4 := self.now;
36              d3 := T4 – T3;
37              if(d3 < 1) { //node (10)
38                setverdict(pass);
39              } else { //node (11)
40                setverdict(conf);
41              }
42              T4 := self.now;
43              d3 := T4 - T3;
44              if(d3 < 1) { //node(12)
45                setverdict(pass);
46              } else { //node(13)
47                setverdict(conf);
48              }
49            }
50            [] PCO1.receive {//node (14)
51              setverdict(fail);
52            }
53          }    //end of alt 4
54        } else { //node (3)
55          setverdict(conf);
56        }
57        []PO.receive { //node (4)
58          setverdict(fail);
59        }
60      } //end of alt 2
61    } else {
62      setverdict(conf);
63    }
64  }
65  []PO.receive {
66    setverdict(fail);
67  }
68 } //end of alt 1
69 } //end of test case
```

(3) Implementation of controllable parameters:

In the test behavior tree, controllable parameters are time intervals between an external input event and its last IO event, which should be set in advance. To implement such parameters, **resume** statements can be used. For example, we set the controllable parameter d_0 in Fig. 5 to 1 time unit, so this parameter can be implemented by line 4~7 in Table 1, which means that after waiting 1 time unit from the time point T_0, PCO1 will send A1 to SUT.

(4) Online evaluations and verdicts setting:

Only online evaluation can be used to test *hard* real-time requirements in real-time interoperability testing. According to **Lemma 1**, Time constraints of the test event nodes on the time point T_k can be represented by a conjunction over a set of linear inequalities on parameters $d_i(i=0,1,\ldots,k\text{-}1)$. Until the time point T_k, values of all uncontrollable parameters have been calculated by transformation rule (2) and values of all controllable parameters have been set in advance, so during the testrun, Mathematical formulae on the values of $d_i(i=0,1,\ldots,k\text{-}1)$ can be used in online evaluations on the time point T_k. In the example of Table 1, the **if** statement of line 17 checks if the test event node (1) is reached by using the condition 1<d2<3 and d1+d2<2.

When verdict nodes of the test behavior tree are reached, verdicts should be set by using **setverdict** statements. Besides **pass** or **fail** verdicts, if functional requirements

are satisfied, i.e., received messages are correct, but non-functional requirements are violated, i.e., time constraints are not satisfied, a **conf** verdict should be set. In the example of Table 1, **setverdict** statement in line 23 set a **pass** verdict for node (7) in Fig. 5; and **setverdict** statement in line 25 set a **conf** verdict for node (8) in Fig. 5.

According to above transformation rules, the test behavior tree in Fig. 5 can be converted to the *TIMED*TTCN-3 test case with about 85 lines codes shown in Table 1.

5 Comparisons with Real-time TTCN

5.1 Real-time TTCN

Real-time TTCN ([WG99]) is a real-time extension of TTCN-2, which is a previous version of TTCN-3. Table 2 is an example of real-time TTCN behavior description. Real-time TTCN extends TTCN-2 both on the level of syntax and semantics. In real-time TTCN, an assumption is made that execution of each statement is instantaneous. On the syntactical level, real-time TTCN adds two columns in dynamic behavior description table: Time and Time options column. Time columns are used to define the earliest and latest execution times (EET and LET) to constrain relative time interval between the associated test event statement and a previous or earlier test event. In real-time TTCN, two types of methods for specifying EET and LET are defined: (1) Define the two values by using time expressions directly, which indicate relative time interval between the execution time of the associated statement and its previous statement (just the parent node in the test behavior tree). In the example of Table 2, line 1 is defined by two constants directly, and line 3 is defined by using a time name, which should be defined in Time Declaration table and be evaluated by an **assignment** statement before use (line 2). (2) Define the two values by using Labels, which indicate relative time interval between the execution time of the associated statement and its earlier statement that labeled as this Label. For example, in Table 2, line 5 defines time constraints (L1+WFN, L1+LET), which means that the time interval between the execution time of line 5 and line 1 are from WFN to LET. The two types of specifications are different from the starting time points for relative time interval. In real-time TTCN, entries in Time Options columns are combinations of M and N. On the semantics level, [WG99] defines its operational semantics and formal semantics based on timed transition system.

Table 2 An example of real-time TTCN behavior description ([WG99])

Test Case Dynamic Behaviour							
Nr	**La**	**Time**	**Time Options**	**Behaviour Description**	**C**	**V**	**Comments**
1	L1	2, 4	M	A ? DATA_ind			Time Label
2				(NoDur := 3)			Time Assignment
3		2, NoDur		A ! DATA_ack			
4				A ? DATA_ind			
5		L1+WFN,	M, N	B ? Alarm			

		L1+LET					

5.2 Problems in transformation from test behavior tree to real-time TTCN

We consider how to transform a test behavior tree to a real-time TTCN test case. Not considering time constraints, the method of transformation is similar to the one for TTCN-3. Now consider time constraints of each node in the test behavior tree of Fig. 5. For controllable parameters, three cases should be considered:

Case 1: Node (3) PO?c, $d_2[d_2<3]$:

In this case, d_2 is just the time interval between the execution time of this statement and its previous statement corresponding to its parent node in the test behavior tree, so the first method of specifying EET and LET in real-time TTCN can be used: $EET_{(3)} = 0$, $LET_{(3)} = 3$.

Case 2: Node (7) PCO1?B_1, $d_3[2<d_1+d_2+d_3<3]$:

In this case, $d_1+d_2+d_3 = T_4 - T_1$, is just the time interval between the execution time of this statement and the statement corresponding to node (a), so the second method of specifying EET and LET in real-time TTCN can be used: Label the statement corresponding to node (a) as L1, then $EET_{(7)}$ = L1+2, $LET_{(7)}$ = L1+3. See Table 3.

Table 3 real-time TTCN representation of Node (7) in Fig. 5

Test Case Dynamic Behaviour							
Nr	**La**	**Time**	**TOpt**	**Behaviour Description**	**C**	**V**	**Comments**
1	L1			PCO1!A1			Node (a)
2				……			
3		L1+2,L1+3		PCO1?B1		P	Node (7)
4				……			

Case 3: Node (2) PO?b, $d_2[1<d_2<3$ and $d_1+d_2\geq 2]$:

This case is most complicated. In this case, time constraints of the node are $1<d_2<3$ and $d_1+d_2\geq 2$: on the one hand, $1<d_2<3$ indicates the time interval between the execution time of this statement and its parent statement; on the other hand, $d_1+d_2=T_3 - T_1$, so $d_1+d_2\geq 2$ indicates the time interval between the execution time of this statement and the statement corresponding to node (a) must be not less than 2 time units. Thus neither the two methods of specifying EET and LET can satisfy the two real-time requirements at the same time. If real-time TTCN should be used in real-time interoperability testing, real-time TTCN must be extended to solve this problem.

For uncontrollable parameters, their values should be set in advance, so the relative time intervals corresponding to such parameters must be fixed values. In this case, EET and LET are the same. For example, in Table 4, line 1 represents the node (a) in the test behavior tree, so its EET and LET are both 1 time unit.

5.3 Possible extensions of real-time TTCN

In this section, we give a suggestion of possible extensions of real-time TTCN for interoperability testing of real-time communication system.

In the **Case** 3 of Section 5.2, time constraints can also be represented as max(2-d_1,1)<d_2<3, where the return value of the function **max()** is the maximal value of parameters. With different values of d_1, the values of max(2-d_1,1) are possibly different: in fact, when d_1<1, max(2-d_1,1)=2-d_1, so time constraints can be represented as 2-d_1<d_2<3; when d_1≥1, max(2-d_1,1)=1, so time constraints are 1<d_2<3. If the above various cases are distinguished in dynamic behavior descriptions, test cases will become very fussy. To avoid such a problem, a uniform syntax can be used. Here, the concept of absolute time must be introduced in real-time TTCN just like in *TIMED*TTCN-3. Possible extensions for real-time TTCN on the syntactical level can be as follows.

(1) Introduce a timestamp recording function **T**( ): for a lable **L**, **T**(**L**) returns the absolute time value of the execution time for the statement labeled as **L**;

(2) Introduce the third type of method for specifying EET and LET: the meanings of EET and LET are the same with the first type of specifying method, i.e., time constraints of the relative time interval between the execution time of the associated statement and the previous statement corresponding to its parent node; in the expressions of EET and LET, function **T**( ), **max**( ) and **min**( ) can be used.

Proposition 1. In real-time interoperability testing, all time constraints of each node in test behavior trees can be represented as EET and LET by the above syntactical extensions.

Proof. According to **Lemma 1**, on the time point T_k, time constraints of the test event nodes can be represented by a conjunction over a set of linear inequalities on d_i(i=0,1,…,k-1). So on the time point T_{k+1} (k=0,1,2…), time constraints of d_k can be reduced to the two following formats: (1) $d_k \le f(d_0,d_1,\cdots,d_{k-1})$ or (2) $d_k \ge g(d_0,d_1,\cdots,d_{k-1})$, here, $f(d_0,d_1,\cdots,d_{k-1})$ and $g(d_0,d_1,\cdots,d_{k-1})$ are both linear expressions on d_0, d_1,..., d_{k-1}; thus EET and LET of the statement corresponding to this node can be represented as **max**$\{ g(d_0,d_1,\cdots,d_{k-1}) \}$ and **min**$\{ f(d_0,d_1,\cdots,d_{k-1}) \}$ respectively. Because of $d_i = T_{i+1}-T_i$ (i=0,1,2,…), so EET and LET also can be represented as **max**$\{ g'(T_0,T_1,\cdots,T_k) \}$ and **min**$\{ f'(T_0,T_1,\cdots,T_k) \}$, here, $g'(T_0,T_1,\cdots,T_k)$ and $f'(T_0,T_1,\cdots,T_k)$ are expressions by using $T_{i+1}-T_i$ in instead of d_i in $g(d_0,d_1,\cdots,d_{k-1})$ and $f(d_0,d_1,\cdots,d_{k-1})$ respectively, and they are all linear expressions on T_0, T_1,..., T_k. If a label is attached to the corresponding statement, T_i (i=0,1,2,…) can be get by function **T**( ). Thus EET and LET of the statement can be specified by using function **T**( ), **max**( ) and **min**( ). □

Table 4 the real-time TTCN test case for the test behavior tree in Fig. 5

Test Case Dynamic Behaviour							
Nr	**La**	**Time**	**TOpt**	**Behaviour Description**	**C**	**V**	**Comments**
1	L1	1		PCO1!A1			Node (a)
2	L2	0,2		PO?a			Node (b)
3		1, 2-T(L2)+T(L1)		PO?b			Node (1)

4		L1+2,L1+3	**M**	PCO1?B1		P	Node (7)
5				PCO1?otherwise		F	Node (9)
6		**max**(2-T(L2)+T(L1),1), 3		PO?b			Node (2)
7		0,1		PCO1?C1		P	Node (10)
8				PCO1?otherwise		F	Node (12)
9		0,3		PO?c			Node (3)
10		0,1		PCO1?C1		P	Node (13)
11				PCO1?otherwise		F	Node (15)
12		0,1		PO?b		F	Node (4)
13				PO?otherwise		F	Node (6)
14				PO?otherwise		F	

On the semantics level, we can also refine operational semantics for the syntactical extensions. Before evaluating a set of alternatives, the values of EET and LET for each alternative should be evaluated at first. If the value of EET for one statement is greater than LET, the corresponding statement should be ignored in the process of evaluation, and test execution should not be stopped.

By using these extensions of real-time TTCN, the test behavior tree shown in Fig. 5 can be converted to a real-time TTCN test case in Table 4.

5.4 Comparisons between *TIMED*TTCN-3 and real-time TTCN

Since *TIMED*TTCN-3 is a real-time extension of TTCN-3, it has also the characteristics of TTCN-3. In this section, we compare *TIMED*TTCN-3 with real-time TTCN only from the aspect of the capability of real-time testing. From the discussions in Section 4 and 5, we can see that

(1) *TIMED*TTCN-3 is powerful enough to specify time constraints in real-time interoperability testing; even more complicated time constraints can be evaluated easily by retrieving current absolute time, storing its value in a variable and passing it to **expression** statements. In real-time TTCN, no concept of absolute time is introduced; only Time and Time Options columns are added to Dynamic Behavior table to specify the earliest and latest execution times, which are constraints of time intervals relative to a fixed time point. So real-time TTCN cannot specify some complicated real-time requirements; one example of such situations has been analyzed in **Case 3** of Section 5.2. To remedy this gap, real-time TTCN should be extended both on the syntactical and semantic levels.

(2) *TIMED*TTCN-3 is more flexible than real-time TTCN for its style like common programming languages in specifying real-time requirements. However, the style of real-time TTCN is more compact and formal.

(3) The semantics of *TIMED*TTCN-3 is straightforward and simple, just like a common programming language. However, the semantics of real-time TTCN is a little more complicated, especially the two time options are fussy and impenetrable.

(4) *TIMED*TTCN-3 supports both online and offline evaluations, so it has the capabilities of evaluating both *hard* and *soft* real-time requirements and it can be used not only real-time testing but performance testing. However, real-time TTCN has only capabilities of evaluating *hard* real-time requirements.

6 Conclusion

*TIMED*TTCN-3 is a real-time extension of TTCN-3. In this paper, we use *TIMED*TTCN-3 in real-time interoperability testing. From system specifications, test behavior trees can be generated. Then transformation rules from such intermediate notations to *TIMED*TTCN-3 test cases are given. We also investigate a real-time extension of TTCN – real-time TTCN. Since this notation has not enough capabilities of specifying time constraints in real-time interoperability testing, we extend real-time TTCN to fill in such a gap and transform test behavior trees to extended real-time TTCN test cases. From the comparisons between the two real-time test notations, it can be concluded that *TIMED*TTCN-3 is more powerful and flexible than real-time TTCN and can be more suitable for real-time interoperability testing.

We have implemented initial prototypes of test execution for both real-time TTCN and *TIMED*TTCN-3. In our future work, we plan to study real-time interoperability testing under distributed test architecture and use the *TIMED*TTCN-3 based test system in real-life timed interoperability testing.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant No. 90104002 and No. 60572082/F010110, and 973 Program of China under Grant No. 2003CB314801.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183–235.

[BB04] L. B. Briones, E. Brinksma. A Test Generation Framework for quiescent Real-Time Systems. Workshop on Formal Approaches to Testing of Software (FATES) 2004: 64-78.

[BST98] S.Bornot, J.Sifakis, S. Tripakis. Modeling Urgency in Timed Systems. COMPOS'97, LNCS 1536, Springer Verlag, 1998.

[DGN02] Z. Dai, J. Grabowski, and H. Neukirchen. Timed TTCN-3 -- A Real-Time Extension for TTCN-3. Testcom2002: 407-424.

[DGN03] Z. Dai, J. Grabowski, H. Neukirchen. Timed TTCN-3 Based Graphical Real-Time Test Specification. TestCom 2003: 110-127.

[EDK02] A. En-Nouaary, R. Dssouli, F. Khendek. Timed Wp-method: testing real-time systems. IEEE Transactions on Software Engineering, 2002, 28(11): 1023 - 1038.

[ETSY04] K. El-Fakih, V. Trenkaev, N. Spitsyna and N. Yevtushenko. FSM Based Interoperability Testing Methods for Multi Stimuli Model. TestCom 2004: 60-75.

[GHRS03] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, et al. An introduction to the testing and test control notation (TTCN-3). Computer Networks, 2003, 42(3): 375-403.

[Hao97] R. Hao. Research on Protocol Conformance and Interoperability Testing based on Formal Methods (In Chinese). PhD thesis, Tsinghua University, P. R. China,

1997.

[HLSG04] R. Hao, D. Lee, R.K. Sinha and N. Griffeth. Integrated System Interoperability Testing With Applications to VoIP. IEEE/ACM Transactions on Networking, 2004, 12(5): 823-836.

[HNTC99] T. Higashino, A. Nakata, K. Taniguchi, and A. R. Cavalli. Generating test cases for a timed I/O automaton model. IFIP TC6 12th International Workshop on Testing Communicating Systems, 1999: 197-214.

[KD03] P. Krémer and S. Dibuz. Framework and Model for Automated Interoperability Test and Its Application to ROHC. Testcom2003: 243 - 257.

[KJM03] A. Khoumsi, T. Jéron and H. Marchand. Test cases generation for nondeterministic real-time systems. Workshop on Formal Approaches to Testing of Software (FATES) 2003, LNCS 2931: 131-146.

[KSK00] S. Kang, J. Shin, and M. Kim. Interoperability Test Suite Derivation for Communication Protocols. Computer Networks, 2000, 32(3): 347-364.

[KT04] M. Krichen and S. Tripakis. Black-Box Conformance Testing for Real-Time Systems. SPIN 2004: 109-126.

[LMN04] K. Larsen, M. Mikucionis, B. Nielsen. Online Testing of Real-time Systems Using Uppaal. Workshop on Formal Approaches to Testing of Software (FATES) 2004: 79-94.

[NDG04] H. Neukirchen, Z. Dai, J. Grabowski. Communication Patterns for Expressing Real-Time Requirements Using MSC and Their Application to Testing. TestCom 2004: 144-159.

[RC90] O. Rafiq and R. Castanet. From conformance testing to interoperability testing. The 3rd Int. Workshop on Protocol Test Systems, 1990.

[SKCK04] S. Seol, M. Kim, S. T. Chanson, and S. Kang. Interoperability Test Generation and Minimization for Communication Protocols Based on the Multiple Stimuli Principle. IEEE Journal on Selected Areas in Communications (JSAC), 2004, 22(10): 2062-2074.

[SKKJ03] S. Seol, M. Kim, S. Kang and J. Ryu. Fully Automated Interoperability Test Suite Derivation for Communication Protocols. Computer Networks, 2003, 43(6): 735-759.

[SVD01] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing Timed Automata. Theoretical Computer Science, 2001, 254(1-2): 225-257.

[TKS03] V. Trenkaev, M Kim, and S. Seol. Interoperability Testing Based on a Fault Model for a System of Communicating FSMs. TestCom 2003, LNCS 2644: 226–242.

[TTCN] ITU-T Recommendation X.292 (1998): OSI Conformance Testing Methodology and Framework for Protocol Recommendations for ITU-T Applications—The Tree and Tabular Combined Notation (TTCN). ITU-T, Geneva (Switzerland).

[TTCN3] ETSI European Standard (ES) 201 873-1 V2.2.1 (2002-08): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis (France), 2002.

[VBT01] C. Viho, S.Barbin and L. Tanguy. Towards a formal framework for interoperability testing. FORTE 2001: 51-68.

[WG99] T. Walter, J. Grabowski. A framework for the specification of test cases for real-time distributed systems. Information & Software Technology, 1999, 41(11-12): 781-798.

[WWY04] Zhiliang Wang, Jianping Wu, Xia Yin. Towards Interoperability Test Generation of Time Dependent Protocols: a Case Study. IEEE Globecom2004, Vol. 2: 589-594.

[WWY06] Zhiliang Wang, Jianping Wu and Xia Yin. A Formal Framework to Interoperability Testing for Real-time Systems. Submitted.