Protocol Conformance Test Case Verification Using Timed-Transitions

Kshirasagar Naik and Behcet Sarikaya

School of Computer Science and Engineering, University of Aizu, Aizu-Wakamatsu City, Fukushima, 965 JAPAN, (k-naik,sarikaya)@u-aizu.ac.jp

Keyword Codes: C.2.2; D.2.4; D.2.5
Keywords: Network Protocols; Program Verification; Testing

Abstract
We develop a methodology to verify the correctness of test cases designed to check timed behavior of protocol implementations. The verification process consists of four steps. First, we model a protocol specification, a test case, and an underlying service provider as Timed Extended Finite-State Machines (TEFSM) and define the resulting system as a Test Verification System (TVS). Next, we algorithmically obtain a *model, i.e.,* a predicated global state space, from the TVS. Test case properties are formulated in terms of safety and liveness using branching time temporal logic. Finally we verify the test case properties on the model of the TVS using a model checking algorithm. We apply the verification technique to a test case for the *Inres* protocol. A few errors are detected in the design of the test case. We observe that without the use of TEFSM model, it would not have been possible to detect any time related errors in the test case.

1 Introduction

Presently test suites for OSI protocols are manually developed and standardized. These test suites may, however, contain several errors [BS93]. Therefore interest in verification of these test suites using formal specification of the protocol is growing [DB90, NS93]. Standards organizations have defined various formal specification languages such as SDL [CCI92]. The Tree and Tabular Combined Notation (TTCN) [ISO91] is defined to specify abstract test suites.

The notion of *time* can be found in many communication systems. It is used in two ways: first, in the form of *timeouts* for the desired operations of the system and second, as a performance parameter of the system. In the present conformance testing framework, only the first use of time is taken into consideration. To control and observe timer activities, the test specification language TTCN defines the appropriate constructs. SDL uses the notion of time in the form of timeouts and delay statements.

Many test cases in various test suites heavily depend on timer actions to generate test events and to assign *Pass, Fail*, or *Inconclusive* test verdicts. Test cases in those test suites are designed with the purpose of testing various retransmission timers in protocol implementations. Thus, to verify the correctness of test case properties, the notion of time must be incorporated into the individual transitions of the component entities of a test system, into the global behavior of a test system, and into the test case properties.

An outline of the test verification methodology presented in this paper is as follows. Because protocols and test cases are generally specified using different languages (SDL and TTCN, respectively), it is essential to represent them in a common notation for the purpose of being able to obtain their global behavior. Thus, we define a kind of TEFSM. The notion of *time* in protocols is important to the proper functioning of communication systems. *Delay* and *timeouts* are two well known timed operations. Therefore, we use the notion of a *timed transition* [OST90] in our model of a TEFSM. We define the notions of safety and liveness properties of test cases. These properties are then verified, using a model checking approach, on the global behavior of the test verification system.

In Section 2, we define the protocol specification and test case models. A TEFSM model is developed in Section 3. The mapping from SDL and TTCN to TEFSM is shortly explained. In Section 4, we discuss the model generation and timed reachability analysis. In Section 5, test case safety and liveness properties are formulated in terms of constructs for specifying the ordering of timed events and specified as temporal formulas. Section 6 is on temporal formula verification using model checking.

2 Protocol and Test Case Specification

The protocol is assumed to be specified in SDL [CCI92]. In SDL it is possible to define timers that can be set/reset by a SDL process. If a timer times out, it is treated as an input to the process and thus transitions can be fired upon a time-out. Test cases can be specified in TTCN [ISO91].

2.1 Inres Protocol

Inres is a connection-oriented protocol that operates between two protocol entities *Initiator* and *Responder*. Inres is described in detail in [BHS91, SAR93]. We will shortly describe the Initiator entity and its SDL specification.

A connection establishment is initiated by the Initiator-user at the entity Initiator with an ICONreq. The Initiator then sends a CR to the entity Responder. After outputting the CR, a timer is started with a value of P (5) units. Responder answers with CC or DR. If Initiator receives a DR from Responder, the disconnection phase is entered. This behavior is specified in SDL as two transitions from the initial state "disconnected":

```
start;
          STATE disconnected;
             INPUT ICONreq;
                TASK counter := 1;
                OUTPUT CR;
                SET (NOW+P, T);
                   NEXTSTATE wait;
```

```
            INPUT DR;
               OUTPUT IDISind;
                  NEXTSTATE disconnected;
         ENDSTATE disconnected;
```

If Initiator receives nothing at all within 5 seconds, CR is transmitted again. If, after 4 attempts, still nothing is received by Initiator, it enters the disconnection phase. In SDL all this is specified as:

```
STATE wait;
 INPUT  T;
  DECISION  counter < 4;
   (true)  :  OUTPUT CR;
     TASK  counter := counter + 1;
     SET (NOW+P, T);
     NEXTSTATE wait;
    (false) :  OUTPUT  IDISind;
     NEXTSTATE disconnected;
   ENDDECISION;
```

In case the Initiator receives a CC in the state "wait" it issues an ICONconf to its user, and the data phase ("connected state") can be entered.

```
         STATE wait;
            INPUT CC;
                  RESET (T);
                  TASK  number := 1;
                  OUTPUT  ICONconf;
                     NEXTSTATE connected;
```

If the Initiator-user issues an IDATreq, the Initiator sends a DT to the Responder and is then ready to receive another IDATreq from the user. IDATreq has one parameter, a service data unit ISDU, which is used by the user to transmit information to the peer user. This user data is transmitted transparently by Initiator as a parameter of the protocol data unit DT. After having sent a DT to Responder, Initiator waits for 5 seconds for a respective acknowledgement AK. Then the DT is sent again. After 4 unsuccessful transmissions, Initiator enters the disconnection phase.

2.2 Test Case

A test case in TTCN for Inres protocol is shown in Fig. 1. The purpose of this test case is to check that the Implementation Under Test (IUT) retransmits a Connection Request (CR) PDU in case of timeout and sends a IDISind (Disconnect Indication) to its user after four unsuccessful attempts. The test case interacts with the IUT at two PCOs, one at the lower (L) and one at the upper (U) service boundary of the protocol entity. The test case sends an ICONreq to the IUT through PCO U. Then it starts a timer and waits for a CR PDU at PCO L. Upon receiving the first CR from the IUT,

Test Case Dynamic Behavior					
Test Case Name: T01					
Reference: INRES/Initiator/Valid Behavior/Connection Establishment					
Purpose: To check that the IUT retransmits CR PDU in case of timeout and releases the connection after four successful attempts					
Default:					
Comments:					
Nr	**L**	**Behavior Description**	**CRef**	**V**	**C**
1		U!ICONreq			User issues CONreq
2		L?CR START TM(5)(c:=1)			IUT transmits CR
3	LA	L?CR START TM(5) (c:=c+1)			CR retransmitted
4		[c<4] -->LA			
5		[c>=4] CANCEL TM			
6		U?IDISind		P	IUT releases conn.
7		U?OTHERWISE		F	
8		L?OTHERWISE		F	
9		?TIMEOUT TM		F	IUT not responding
10		L?OTHERWISE		F	
Detailed Comments:					

Figure 1: TTCN Description of the Test Case

the test case starts a timer of 5 seconds. Since the test case does not send a CC PDU to the IUT through PCO L at all, the IUT should retransmit the CR upon a timeout of 5 seconds. The test case is designed to check if the IUT makes *four* attempts to establish a connection by sending a CR PDU each time. However, on all the occasions, the test case does not respond by sending a CC PDU and the IUT should send a disconnect indication to the test case by sending a IDISind to PCO U. If the test case receives a IDISind at PCO U, the objective of the test case is fulfilled and a Pass test verdict is assigned to the IUT. If the test case receives any event except a CR at PCO L, it assigns a Fail test verdict to the IUT.

3 Timed Extended Finite-State Machine Model

We define a communicating TEFSM as $F = < S, S_t, V, R, s_{init}, Z, h_0, C_I, C_O >$, where S is a finite set of states, $S_t = \{(s,x) | s \in S$ and x is a tag value $\}$ is a tagged set of states, $V = \{vl, ..., vn\}$ is a finite set of data variables of types {t1, ..., tn}, respectively, R is a finite set defined below, $s_{init} \in S$ is the initial state, $Z \subseteq S$ is a set of final states, h_0 is a set of assignment functions initializing some variables in V, $C_I(C_O)$ is a set of input (output) FIFO channels.

R is a set of transitions of the form $r = < s, s', a, e, h, m, [l, u] >$, where s is the *from* state and s' is the *to* state of the transition, a is the *action* or *event* clause causing the transition to fire, e is the *enabling predicate* of the transition, h is a set of value assignments to a subset of V, m is a priority number of the transition in a set of alternative transitions with the same *from* state. The components l and u in $[l, u]$ represent the *lower* and *upper* time bounds, respectively, on r. The set R_F denotes the set R from TEFSM F and e_r denotes the enabling predicate e of transition r. For the tick transition, l is set to 0 and u is set to ∞; for a timeout transition, $l = u = T$ and for any other transition $l = u = 0$.

We assume that there is an external global clock which ticks infinitely often. The time clause $[l, u]$ of a transition contains lower and upper time bounds. The lower and upper time bounds are measured with respect to the ticks of the clock, and can thus be used in modeling timed properties including delays and timeouts. Once a transition's enabling predicate becomes true, the transition is fired within lower and upper time bounds from the moment the transition is enabled. A transition with upper time bound *infinity* is called a *spontaneous* or *nondeterministic* transition. In addition to the state and data variables, the variable set V always contains a clock variable t to hold the present clock tick number, which is a non-negative integer value. The global clock is represented by the following tick transition: $tick = < FROM, TO, NULL, TRUE, [t := \ t + 1], 1, [0, \infty] >$, where FROM and TO are states in the global state space of the test system, NULL represents a null event, the tick function $[t := \ t + 1]$ denotes the fact that the occurrence of *tick* results in t being incremented by 1 and all other state and data variables except timer variables remaining the same. The priority of *tick* is 1 because there is no executable transition alternative to *tick*. The idea behind associating a tuple $[0, \infty]$ with a tick is to be able to give a physical interpretation to *tick* in the sense that a *tick* represents a logical unit of time that can be implemented by a suitable real-time interval. Let TI_T and TI_S be the set of active timer identifiers in the test case and specification, respectively. When a tick occurs the timer identifiers are updated as: $[v_i := v_i - 1 | v_i \in TI_T \cup TI_S]$.

3.1 TEFSM Model of Specification

The first step in obtaining a TEFSM from an SDL specification is normalization, where one TEFSM is obtained from one SDL process. Syntactic transformations are applied to eliminate the decision clauses so that the normalized transitions contain single paths. Also channel names are assigned to input and output signals. We describe some details of normalization related to time constructs. For **SET(NOW+P,T)**, a transition of the form: $< From, To, null, true, h, 1, [0, 0] >$ is created, where h is the set of assignments containing $T := P$, and T is the timer identifier. Similarly for the **RESET** construct, $T := 0$ is generated in the h clause. For a timeout transition, INPUT T, we create a transition: $< From, To, null, true, \{\}, 1, [T, T] >$.

If a transition in SDL generates one or more outputs, a normalized transition is generated for each output statement. Normalization of SDL specifications described in [SAR93] allows several output statements in the normalized transitions while the TEFSM has transitions with only a single (input or output) event. For specifications containing several processes, several TEFSMs are obtained after normalization. A second step is needed to calculate the product of all the communicating TEFSMs using the traditional reachability algorithm. TEFSM model of the Inres Initiator can be generated from its SDL specification. For example the transition from "disconnected" with ICONreq as input is modelled as three transitions (numbered SPi) of the form:

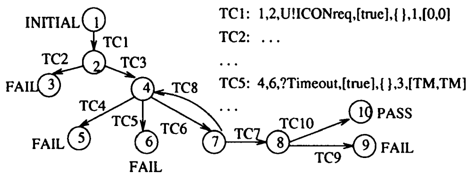


Figure 2: TEFSM model of the Test Case

$SP1 :< disconnected, templ, ISAP?ICONreq, true, \{counter := 1\}, 1, [0, 0] >$
$SP2 :< templ, temp2, MSAP!CR, true, \{\}, 1, [0, 0] >$
$SP3 :< temp2, wait, null, true, \{T := P\}, 1, [0, 0] >$
The other transitions are similarly generated. For example the true branch of the decision transition from "wait" state is modelled as:
$SP22 :< temp3, temp4, MSAP!CR, true, \{\}, 1, [0, 0] >$

3.2 TEFSM Model of Test Case

A TTCN test case is mapped to a TEFSM in three steps. In the first step, constraints are processed and default behaviors are expanded. A send constraint is translated to a set of assignments and a receive constraint is translated to a conjunction of predicates. In the second step, a TEFSM is derived from the main tree and for each of the subtrees in the dynamic behavior part of the test case. In the third step, subtree attachments are resolved by combining the corresponding TEFSMs. For example, TTCN event lines are processed as follows. An event line:
$L!P\_CONreq[x = 2](a := 1)P\_CON\_base,$
where P_CON_base is a constraint on the output event P_CONreq, is translated to a transition
$< s1, s2, L!P\_CONreq, [x = 2], (a := 1)Uf1, 1, [0, 0] >,$
where $f1$ is a set of assignments obtained by processing the constraint P_CON_base. TTCN timer events **START**, **CANCEL**, and **TIMEOUT(T)** are semantically similar to SDL **SET**, **RESET** and **INPUT T**, respectively and thus they are processed similarly as in SDL. TTCN alternatives are translated into transitions going out from the same state. For **OTHERWISE** a transition of the form:
$< sx, sy, OTHERWISE, [p], f, n, [0, 0] >$
where p is the conjunction of predicates associated with OTHERWISE, f is a set of assignments and n is the priority number.

Priority numbers are assigned in the top-down order of the syntactic appearance of event lines in a set of alternatives. The first event line in a set of alternatives is assigned a priority of 1, the second line a priority of 2, and so on. TEFSM model of the example test case in Fig. 1 is shown in Fig. 2.

4 Model Generation

Model generation consists of two steps: (i) generation of global state space from a TVS using a timed reachability analysis algorithm and (ii) generation of a model from the global state space by associating a set of predicates with each state.

4.1 Timed Reachability Analysis

A *Test Verification System* (TVS) is defined to be a 5-tuple, $TVS = <\Sigma, \Omega, P, \Psi, C>$, where Σ is a TEFSM corresponding to the Lower Tester, Ω is a TEFSM corresponding to the underlying service provider, P is a TEFSM corresponding to the protocol specification, Ψ is a TEFSM corresponding to the Upper Tester, and C is a set of *channel functions* defining the interconnection among Σ, Ω, P, and Ψ.

A channel function *channel(TEFSM1, TEFSM2)* denotes that $TEFSM1$ outputs messages to the *channel* which are received by $TEFSM2$.

The global state s of a test verification system $TVS = <\Sigma, \Omega, P, \Psi, C>$ is defined as a 6-tuple $<\Sigma_s, \Omega_s, P_s, \Psi_s, C_s, \Pi>$, where Σ_s, Ω_s, P_s, and Ψ_s represent the present states of Σ, Ω, P, and Ψ, respectively, and C_s is a set of states consisting of the present states of each channel in C; Π is a set containing values of all the variables in the TEFSMs in the TVS including v, a unique variable used to hold the test verdict, and t, another unique variable to hold the global time tick value.

The initial global state s_0 is defined as follows:
$<s_{init}(\Sigma), s_{init}(\Omega), s_{init}(P), s_{init}(\Psi), C_{empty}, init(\Pi)>$,
where $s_{init}(\Sigma)$ is the initial state of Σ, $s_{init}(\Omega)$ is the initial state of Ω, $s_{init}(P)$ is the initial state of the protocol specification entity P, $s_{init}(\Psi)$ is the initial state of Ψ, C_{empty} denotes all the channels in C to be empty, and $init(\Pi) = \{h_0(\Sigma) \cup h_0(\Omega) \cup h_0(P) \cup h_0(\Psi) \cup \{t := 0, v := null\}\}$, where the function h_0 denotes initial assignments to the variables in the corresponding TEFSM. Notationally, the present state of a TEFSM M is denoted by the function notation $ps(M)$.

The set of enabled transitions in a global state s consists of all the transitions, whose enabling conditions evaluate to true, in the present states, contained in s, of the component TEFSMs. Without any timing constraints, the set of enabled transitions can be used to perturb s to generate a set of successor states of s. However, in a timed transition system, the lower time bound l of an enabled transition r must elapse before r can be used to perturb s [OST90]. Therefore, we split the set of enabled transitions into two sets: a set of *executable* transitions and a set of *pending* transitions.

A transition remains pending from the instant it becomes enabled until the elapse of its lower time bound l at which point the transition becomes executable. Therefore, a *history* must be maintained of when each transition became enabled, so that it can be determined when the transition becomes eligible for execution. The history field is needed for construction of the reachability graph, but can be discarded once the complete graph has been obtained. A global state s can be extended to incorporate a history field to generate a *node* in the global state space. Formally, a node is denoted as $n = (s, ET(n))$, where the set of enabled transitions ET(n) is called the history field of n.

The set of *enabled* transitions $ET(n)$, occurring in the present global state $s = <\Sigma_s, \Omega_s, P_s, \Psi_s, C_s, \Pi>$ in node n in $TVS = <\Sigma, \Omega, P, \Psi, C>$ is defined as the set of all transitions, in the TEFSMs Σ, Ω, P, and Ψ, whose enabling conditions evaluate to true, that is,

$ET(n) = \{r | r \in \{R_\Sigma \cup R_\Omega \cup R_P \cup R_\Psi\} \wedge from(r) \in \{\Sigma_s, \Omega_s, P_s, \Psi_s\} \wedge e_r = true\}$.

Since the evaluation of e_r involves accessing the channel contents and taking the priority number of a transition in a set of alternative transitions in a TEFSM, $ET(n)$ is computed in the following manner:

$ET(n) = \{Exec(\Sigma, s) \cup Exec(\Psi, s) \cup Exec(P, s) \cup Exec(\Omega, s)\}$, where the procedure $Exec$ returns those transitions from s whose enabling conditions evaluate to true [NS93].

The set of *executable* transitions $XT(n)$ in a node $n = (s, ET(n))$ is defined as follows:

$XT(n) = \{r | r = < From, To, a, e, h, m, [l, u] > \in ET(n) \wedge l = 0\}$.

The set of *pending* transitions $PT(n)$ in a node $n = (s, ET(n))$ is defined as follows: $PT(n) = \{r | r = < From, To, a, e, h, m, [l, u] > \in ET(n) \wedge l > 0\}$.

Therefore, $ET(n) = XT(n) \cup PT(n)$. The firing of an executable transition can be delayed until the elapse of its upper tick bound. Thus, we define a set of *must* transitions, which are executed before the next clock tick.

The set of must transitions $MT(n)$ in node $n = (s, ET(n))$ is defined as follows:

$MT(n) = \{r | r = < From, To, a, e, h, m, [l, u] > \in ET(n) \wedge l = u = 0\}$.

When the clock ticks, the lower and upper time bounds of enabled transitions are decremented by one. Decrementing $l = 0$ leaves l at 0 and decrementing $u = \infty$ leaves u at ∞. By default, $ET(n)$ always contains the *tick* transition. In the state perturbation process, the transitions in $MT(n)$ must happen from node n prior to the next clock tick in order to meet the upper time bound requirements on those transitions.

In the perturbation process, we use two notations: $e_r(s)$, which is the enabling condition of a transition r in global state s and $h_r(s)$, which is a state obtained from s by applying the transformation function h in r to s.

Given any node $n = (s, ET(n))$ in the global state space, successor nodes are defined as $n' = (s', ET(n'))$ when a transition $r = < From, To, a, e, h, m, [l, u] > \in XT(n)$ is used to perturb n. There are two cases to be considered:

Case 1 : If the transition r is not a *tick* transition, then $s' = h_r(s)$ and $ET(n')$ is the newly computed set of enabled transitions in state s'. In this case, the value of the time variable t is unchanged.

Case 2 : If the transition r is a *tick* transition, s' is the same as s with the global time variable t updated as $t := (t + 1)$ and the bounds [l, u] in the transitions in $ET(n)$ are decremented by one, i.e.,

$ET(n') = \{< From, To, a, e, h, m, [l', u'] > | r = < From, To, a, e, h, m, [l, u] > \in ET(n)$ and $l' := (l - 1)$ if $l > 0$ otherwise 0, and $u' := (u - 1)\}$

If $l = 0$ then it stays at zero, and decrementing $u = \infty$ leaves it at infinity. If *tick* is taken for perturbing a state, then no $r \in ET(n)$ has $u = 0$. Thus, u can always be decremented without becoming negative.

ALGORITHM

Input: A test verification system and the capacities of the channels in it.

Output: Global state space S.

S1: Define a set of global nodes N and a set of global transitions R. Initially, N contains only the initial global node $n_1 = (s_1, ET(n_1))$ and $R = \phi$.

S2: Find a member $n = (s, ET(n)) \in N$ of the set of global nodes whose perturbations have not been determined. If no such member exists, then strip all the history fields from N to obtain S and stop.

```
S3: If $MT(n) \neq \phi$, then $XT(n) := XT(n) - tick$.
S4: Compute $N_p$, a set of global states by perturbing $n$. Initially $N_p = \phi$.
$\forall r = < From, To, a, e, h, m, [l, u] > \in XT(n)$, do {
compute $n' := (s', ET(n'))$, where $s' := h_r(s)$;
set $N_p := N_p \cup \{n'\}$;
set $R := R \cup \{< s, s', a, e, h, m, [l, u] >\}$}
S5: If $N_p$ is an empty set, report $n$ as a terminal node in the global node space.
S6: $\forall n = (s, ET(n)) \in N_p$ do {
if channeloverflow($s$) then mark $n$ "perturbed" and set $N := N \cup \{n\}$
else if $n \notin N$ then mark $n$ "unperturbed" and set $N := N \cup \{n\}$}
S7: Go to step S2.
```

Termination of Step 2 of the above algorithm is guaranteed if at least one TEFSM has a finite behavior. We will shortly explain an example execution of the above algorithm for constructing the global state space for the Inres example. The s component of the initial state $n1$ is:
(1,227,E,E,E,E,{counter:=1,c:=0,v:=null,TM:=0,t:=0})
and ET component is the union of XT which is:
$\{< 1, 2, U!ICONreq, [T], \{\}, 1, [0, 0] >\}$
and PT which is $\{\}$. Now, n_1 is perturbed using the transition in XT to reach n_2:
n2= ((2,227,ICONreq,E,E,E,{counter:=1,c:=0,v:=null,TM:=0,t:=0})
$\{< 227, 157, I?sp(1), [T], \{\}, 1, [0, 0] >\}\{\})$

After a few perturbation steps we reach node n_5:
n5= ((2,162,E,E,CR,E,{counter:=1,c:=0,v:=null,TM:=0,t:=0})
$\{< 162, 165, i, [true], \{counter := counter + 1\}, 1, [0, 0] >,$
$< 2, 4, L?CR, [true], \{TM := 5, c := 1\}, 1, [0, 0] >\}\{\})$

We perturb n_5 using the two transitions of its XT component to obtain the nodes n_6 and n_7 (see Fig. 3):
n6= ((4,162,E,E,E,E,{counter:=1,c:=1,v:=null,TM:=5,t:=0})
$\{< 162, 165, i, [T], \{counter := 1\}, 1, [0, 0] >\}\{\})$
n7= ((2,165,E,E,CR,E,{counter:=2,c:=0,v:=null,TM:=0,t:=0})
$\{< 2, 4, L?CR, [T], \{TM := 5, c := 1\}, 1, [0, 0] >\}$
$\{< 165, 169, i, [T], \{\}, 1, [5, 5] >\})$

4.2 Associating Predicates with Global States

There are five types of predicates that can be associated with the global states of a test verification system: *state predicate, variable predicates, event predicates, PCO predicates,* and *verdict predicates.* The state predicate INIT holds only in the initial state. The variable predicates are assertions about the values of the variables in the global state space. The event predicates characterize the possibility or the actual execution of specified events. The two event predicates are AT and AFTER. PCO predicates state assertions on the PCOs and the input/output directions of events occurring at the PCOs. The verdict predicates are assertions on the test verdict and is one of the following three: (v = Pass), (v = Inconclusive), and (v = Fail) where v is the unique variable.

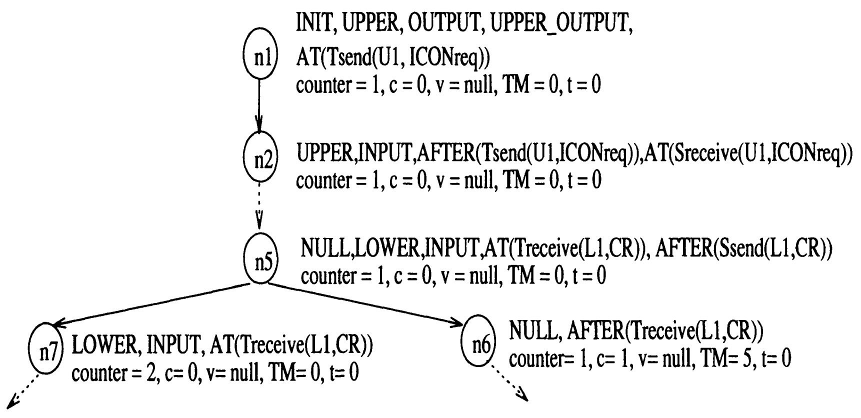

n1
INIT, UPPER, OUTPUT, UPPER_OUTPUT,
AT(Tsend(U1, ICONreq))
counter = 1, c = 0, v = null, TM = 0, t = 0
n2
UPPER,INPUT,AFTER(Tsend(U1,ICONreq)),AT(Sreceive(U1,ICONreq))
counter = 1, c = 0, v = null, TM = 0, t = 0
n5
NULL,LOWER,INPUT,AT(Treceive(L1,CR)), AFTER(Ssend(L1,CR))
counter = 1, c = 0, v = null, TM = 0, t = 0
n7
LOWER, INPUT, AT(Treceive(L1,CR))
counter = 2, c= 0, v= null, TM= 0, t= 0
n6
NULL, AFTER(Treceive(L1,CR))
counter= 1, c= 1, v= null, TM= 5, t= 0

Figure 3: Global State Space with Predicates

The association of predicates to the global states is straight forward. The variable predicate associated with a global state is simply the set of the enabling predicates of the outgoing transitions from the state. The verdict predicate for each state is obtained from the value of the *verdict* variable v associated with the state.

The event predicate is associated with a state when there is an external event in any transition leading **to** or outgoing **from** the state. If the event is a receive test case(specification) event then AT(T(S)receive(Channel, Event)) is associated with the **from** state and AFTER(T(S)receive(Channel, Event)) is associated with the **to** state.

The PCO predicate is associated with a state when there is an external event in any transition outgoing from the state. If the event is an input (output) event at the lower PCO then the predicate LOWER (LOWER, LOWER_OUTPUT) is true. Otherwise, if the event is an input (output) event at the upper PCO then the predicate UPPER (UPPER, UPPER_OUTPUT) is true. If the transition is an internal transition then the PCO predicate INTERNAL is associated. For all other transitions the NULL predicate is associated [NS93].

4.3 Example

The Inres Initiator Test Verification System global state space can be generated following the above two steps. A total of 126 states and 135 transitions results from this process. The complete global system is drawn in Appendix A. A part of the global system is shown in Fig. 3 with predicates assigned to each state.

5 Temporal Formula Generation

First the temporal logic is introduced. Next a set of constructs to characterize the ordering of events in timed CSP (Communicating Sequential Processes) is used to describe test case properties at a high level [KR93]. Finally these properties are formally defined in temporal logic [CES86]. Incorporating real-time requirements in temporal formulas can be done in three ways: bounded temporal operators, freeze quantification, and explicit clock variable [ALHE91]. We take the explicit clock variable approach because it facilitates direct use of the model checking algorithm. In addition, use of a clock variable to keep track of time makes the reachability analysis process easier.

5.1 Temporal Logic

Let AP be a set of atomic predicates. A Branching Time Logic (BTL) structure is defined as a 5-tuple: $M = < S, V, R, P_r, s_{init} >$, where S is a finite set of states, V is a finite set of variables, $R \subseteq (S \times S)$ is a set of transitions among the elements of S, $P_r : S \longrightarrow 2^{AP}$ assigns to each state the set of atomic predicates evaluating to true in that state, and $s_{init} \in S$ is the initial state [CES86].

Using the propositional logic operators $\neg, \wedge$, and $\vee$ and the $Until(U)$ operator, formulas of a BTL structure are defined as $f, \neg f, f \wedge g, f \vee g, A[fUg]$, and $E[fUg]$, where $f, g \in AP$ and AU and EU are referred to as the universal and existential *Until* operators, respectively [CES86]. We use a standard notation to express the truth value of a formula f in a BTL structure M : $(M, s_0 \models f)$ means that the temporal formula f holds at state s_0 in structure M. When the structure M is understood, we simply write $s_0 \models f$. The following abbreviations are also used in writing BTL formulas:
$AF(f) \equiv A[TrueUf]$ means that f holds in the future along every path from s_0. $EF(f) \equiv E[TrueUf]$ means that there is some path from s_0 that leads to a state at which f holds. $EG(f) \equiv \neg AF(\neg f)$ means that there is some path from s_0 on which f holds at every state. $AG(f) \equiv \neg EF(\neg f)$ means that f holds at every state on every path from s_0. $(f_1 \leadsto f_2) \equiv AG(f_1 \longrightarrow AF(f_2))$ (read "f_1 leads to f_2") means that for any time at which f_1 is true, f_2 must be true then or at some later time.

5.2 Temporal Formula for Safety

Based on the idea that nothing bad happens during a test case execution, the safety properties of a test case can be classified into three distinct categories [NS93]: transmission safety, reception safety, and verdict safety. All of the above properties can be expressed using the temporal formula corresponding to the following two constructs in timed-CSP:

CONS1: A **causes** $B \equiv$ After A there must be a B.

CONS2: A **causes** B **unless** $C \equiv$ After A there must be a B, unless a C occurs.

CONS1 is used to specify transmission safety properties such as:

After a send event in the test case (protocol specification) there must be a receive event in the protocol specification (test case). This property is stated in temporal logic as:

$s_{init} \models (AFTER(Tsend(Q, E)) \leadsto AFTER(Sreceive(Q, E)))$

$s_{init} \models (AFTER(Ssend(Q, E)) \leadsto AFTER(Treceive(Q, E)))$.

CONS2 is used to specify reception safety properties such as:

Arriving at a state causes a receive event in the test case (protocol specification) unless an internal event occurs. This property is stated in temporal logic as:

$s_{init} \models (AT(Treceive(Q_i, E_i)) \leadsto AFTER(Treceive(Q_i, E_i)) \vee AFTER(Tinternal))$

$s_{init} \models (AT(Sreceive(Q_i, E_i)) \leadsto AFTER(Sreceive(Q_i, E_i)) \vee AFTER(Sinternal))$.

CONS1 is used to specify the safety property about verdict assignment as follows:

After a receive (send) event in the test case there must be an assignment verdict $\neq$ *Fail.* This property is stated in temporal logic as:
$s_{init} \models true \rightsquigarrow AG(\neg(v = Fail))$.

5.3 Temporal Formula for Liveness

A test case that has the liveness property means that the test behavior satisfies the test purpose and eventually assigns a pass verdict. This can be expressed in temporal logic as: $s_{init} \models (f_1 \rightsquigarrow (v = Pass))$, where f_1 is a temporal formula representing the test purpose. Test purposes expressed in natural language can be converted into temporal formula in three steps: **(i)** rewrite the test purpose as a collection of primitive test purposes, **(ii)** express each primitive test purpose as a temporal formula, and **(iii)** combine the temporal formulas corresponding to the primitive test purposes into a single temporal formula. The following four constructs from timed-CSP are used in expressing primitive test purposes.

CONS3: **An Event Happens During Interval** T_0: The primitive test purpose for stating that the predicate p_s is true at time T and the event E_j is received through channel Q_j during an interval of length T_0 such that the predicate p_r is true is stated as:
$(p_s \wedge (t = T)) \rightsquigarrow (p_r \wedge AFTER(Treceive(Q_j, E_j)) \wedge (T \leq t \leq T + T_0))$

CONS4: **An Event does not Happen During** T_0: The primitive test purpose for stating that the predicate p_s is true at time T and the system waits for an interval T_0, then no event is received during this period is stated as:
$(p_s \wedge (t = T)) \rightsquigarrow (p_s \wedge \neg AFTER(Treceive(ANY, ANY)) \wedge (T \leq t \leq T + T_0))$

CONS5: **An Event A must Happen Only If a B Event Happens**: An event $E_i(B)$ is sent through channel Q_i with the predicate p_s is true, then an event $E_j(A)$ through channel Q_j is received and p_r holds is expressed as:
$(p_s \wedge AFTER(Tsend(Q_i, E_i)) \wedge (t = T)) \rightsquigarrow (p_r \wedge AFTER(Treceive(Q_j, E_j)) \wedge (T \leq t \leq T + T_0))$

CONS6: **After an A There Can Be no B During** T_0: An event E_i is sent through channel Q_i at time T with predicate p_s true and no event is received during T_0 can be expressed as:
$(p_s \wedge AFTER(Tsend(Q_i, E_i)) \wedge (t = T)) \rightsquigarrow (p_s \wedge \neg AFTER(Treceive(ANY, ANY)) \wedge (T \leq t \leq T + T_0))$

Primitive test purposes can be composed using the logic operator of "and" ($\wedge$). This way higher level, meaningful test purposes can be specified in temporal logic as a sequence of primitive test purposes.

6 Temporal Formula Verification

Temporal formula verification is done using a known model-checking algorithm [CES86], where a temporal formula is represented in a tree structure. Leaf nodes contain atomic predicates and all other nodes contain temporal operators. Showing that the formula holds is done by traversing the formula tree from leaf nodes to the root node and verifying each subformula. The entire formula is said to be verified if the root node subformula holds. We describe model checking in detail and give examples of safety and liveness property verification.

6.1 Model Checking Algorithm

For each formula, the model checker maintains two arrays *nf* and *sf*. The lengths of the arrays nf and sf are the length of the formula. nf[i] stores the ith subformula and sf[i] is the list of indices into the array nf to denote the position of successor subformulas of ith subformula. Essentially these two arrays maintain the formula in prefix notation.

As an example, the temporal formula :

$s_{init} \models (t = 0) \rightsquigarrow (AFTER(Treceive(L1, CR)) \wedge (0 \leq t \leq 0 + 5))$ is stored as:

nf[1]	$(\rightsquigarrow ((t = 0) \wedge ((AFTER(Treceive(L1, CR))(0 \leq t \leq 0 + 5)))))$	sf[1]	(2 3)
nf[2]	(t=0)	sf[2]	nil
nf[3]	$(\wedge((AFTER(Treceive(L1, CR))(0 \leq t \leq 0 + 5))))$	sf[3]	(4 5)
nf[4]	AFTER(Treceive(L1,CR))	sf[4]	nil
nf[5]	$0 \leq t \leq 0 + 5$	sf[5]	nil

A bit array L of the same length is defined for each state in the model. The verification starts with the formula nf[f_i] where f_i is the length of f. In the above example, first the formula $0 \leq t \leq 0 + 5$ is considered. All the states $s \in S$ of the model are labeled by setting L[s][5] to true for all s where nf[5] holds. This marks all states after INIT where the time is less than or equal 5 (units). Next the subformula AFTER(Treceive(L1,CR)) corresponding to $f_i = 4$ is processed similarly.

The subformula $(\wedge((AFTER(Treceive(L1, CR))(0 \leq t \leq 0 + 5))))$ corresponding to $f_i = 3$ is processed by setting L[s][3] to true for all $s \in S$ for which L[s][4] and L[s][5] are true. The subformula (t=0) corresponding to $f_i = 2$ is processed as the subformulas $f_i = 4$ or $f_i = 5$. Finally the root node subformula $f_i = 1$ is processed by checking on all the paths whether $s_j \in S$ is a successor of $s_i \in S$ with L[s_j][3] and L[s_i][2] set to true.

6.2 Verification of Test Case Safety and Liveness Properties

In Appendix A we show the structure of the global state space of the Inres Protocol TVS. The predicates are omitted to save space. Appendix A contains one initial state and four final states. Let us denote a sequence of states from the initial state to a final state as a path and represent the path by the function $path(n_i, n_j)$, where n_i and n_j are initial and final states, respectively. We will analyze the test case properties with respect to four paths, $path(n_1, n_{97})$, $path(n_1, n_{115})$, $path(n_1, n_{124})$, and $path(n_1, n_{70})$. Though there are many sequences of states leading from the initial state to a final state, the result of this analysis is the same for all such sequences.

6.2.1 Safety Properties

Following Section 5.2, we derive 3 transmission (one for the test case and two for the specification), 3 reception (for the test case), and one verdict safety properties. All 3 reception safety properties are proved to be true on the global state space. We will show that the transmission safety property formulated using CONS1 in Section 5.2 for the test case holds:

$s_{init} \models AFTER(Tsend(U1, ICONreq)) \rightsquigarrow AFTER(Sreceive(U1, ICONreq))$.

The predicate AFTER(Tsend(U1,ICONreq)) holds in the global state denoted by node n_2 and the predicate AFTER(Sreceive(U1,ICONreq)) holds at node n_3. Since node n_3 appears on all the paths from the initial state to the final states, the above property is satisfied by the model.

The safety property due to the transmission of the first CR
$s_{init} \models AFTER(Ssend(L1, CR_1)) \wedge (t = 0) \leadsto AFTER(Treceive(L1, CR_1)) \wedge (0 \leq t \leq \infty)$
can easily be shown to hold.

The safety property due to the transmission of the second CR from the specification given below:
$s_{init} \models AFTER(Ssend(L1, CR_2)) \wedge (t = 0) \leadsto AFTER(Treceive(L1, CR_2)) \wedge (0 \leq t \leq 5)$.
does not hold. The predicate AFTER(Ssend(L1,CR_2)) holds in the states corresponding to nodes n_{22}, n_{39}, and n_{56}, but the predicate AFTER(Treceive(L1,CR_2)) does not hold in any of the states on the paths from n_{22} to n_{97}, from n_{39} to n_{115}, and from n_{56} to n_{124}. This safety error arises because of a timeout in the test case as explained in the following.

From the node n_{15}, there are two possible transitions,
$< n_{15}, TC5, n_{21} >$ and $< n_{15}, SP22, n_{16} >$,
which are due to the TC5 and SP22 transitions in the test case TEFSM and the protocol specification TEFSM, respectively. Transition TC5 represents a timeout event in the test case and SP22 is a transition that outputs a CR PDU. That is, in this test case, the length of the timer is such that the timeout occurs in the test case before the specification can output the desired CR PDU. Hence, in order to eliminate this safety error, the duration of the timer in the test case must be suitably adjusted.

Now we consider the verdict safety property, which is given by
$s_{init} \models AG(\neg(v = Fail))$. This property does not hold because the predicate $(v = Fail)$ holds in many states such as n_{97}, n_{115} and n_{124}. The significance of a verdict safety error is that the test case is likely to assign a *Fail* verdict to a correct implementation of the protocol on some executions.

6.2.2 Liveness Property

In the following, we show that the test purpose is not properly implemented in the test case. The test purpose is specified as follows: *To check that IUT retransmits CR_PDU in case of timeout, and releases the connection after four unsuccessful attempts.*

Following Section 5.3 we first express the test purpose as a temporal formula. We rewrite the test purpose as a sequence of basic steps:

(i) When the TEFSM sends a ICONreq to the IUT at PCO U, the IUT sends a CR PDU to the TEFSM at PCO L.

(ii) If the TEFSM waits for *five* seconds, it receives a CR from the IUT at PCO L.

(iii) (Step (ii) repeats *four* times.)

(iv) The TEFSM receives a IDISind at PCO U.

The primitive test purposes in temporal logic corresponding to the above steps can be formulated using CONS3 in Section 5.3:

(i) $AFTER(Tsend(U1, ICONreq)) \leadsto AFTER(Ssend(L1, CR))$.

(ii) $(t = T) \leadsto (AFTER(Treceive(L1, CR)) \wedge (T \leq t \leq T + 5))$.

(iii) (Step (ii) repeats *four* times.)

(iv) $AFTER(Treceive(U2, IDISind)) \wedge (t \geq 15)$.

We compose these basic test purposes using the $\wedge$ operator to give rise to a formula for the entire test purpose as follows:

$$\begin{aligned} f_1 = \; & AFTER(Tsend(U1, ICONreq)) \leadsto AFTER(Ssend(L1, CR)) \wedge \\ & (t = 0) \leadsto (AFTER(Treceive(L1, CR)) \wedge (0 \leq t \leq 0 + 5)) \wedge \\ & (t = 5) \leadsto (AFTER(Treceive(L1, CR)) \wedge (5 \leq t \leq 5 + 5)) \wedge \\ & (t = 10) \leadsto (AFTER(Treceive(L1, CR)) \wedge (10 \leq t \leq 10 + 5)) \wedge \\ & (t = 15) \leadsto (AFTER(Treceive(L1, CR)) \wedge (15 \leq t \leq 15 + 5)) \wedge \\ & AFTER(Treceive(U2, IDISind)) \wedge (t \geq 15) \end{aligned}$$

Then, the liveness property of the test case is stated as $s_{init} \models (f_1 \leadsto (v = Pass))$. The predicate $(v = Pass)$ holds in node n_{70}. However, f_1 does not hold on $path(n_1, n_{70})$. The predicate AFTER(Tsend(U1, ICONreq)) holds in node n_2 and the predicate AFTER(Ssend(L1, CR)) holds in node n_5. The atomic predicate $(t = 0)$ is satisfied in nodes n_1 through n_8 and the predicate $(AFTER(Treceive(L1, CR)) \wedge (0 \leq t \leq 0 + 5))$ is satisfied in node n_{17}. The predicate $(t = 5)$ is satisfied in nodes n_{18} through n_{25} and the predicate $(AFTER(Treceive(L1, CR)) \wedge (5 \leq t \leq 5 + 5))$ is satisfied in node n_{34}. The predicate $(t = 10)$ is satisfied by the nodes n_{35} through n_{42} and the predicate $(AFTER(Treceive(L1, CR)) \wedge (10 \leq t \leq 10 + 5)$ is satisfied in node n_{51}. The predicate $(t = 15)$ is satisfied in nodes n_{52} through n_{59}, but no nodes following these nodes satisfy the predicate $(AFTER(Treceive(L1, CR)) \wedge (15 \leq t \leq 15 + 5))$.

The above analysis of the test purpose temporal formula suggests that the test case receives only *three* retransmissions of the CR PDU and not *four* as stated in the test purpose. Hence, an error exists in the dynamic behavior of the test case. We attribute the cause of the above error to a bad initialization of the counter variable c in the test case. Since the variable c has been initialized to 1 and the behavior of the test case loops back to receive a CR PDU with the condition $c < 4$, naturally the test case will receive only *three* retransmitted CR PDUs and not *four*.

7 Conclusions

We presented a methodology to verify timed properties of test cases. Test cases and protocol specifications are modeled as TEFSMs. Test case properties are formulated in terms of some general constructs to describe the occurrences of timer events similar to those in the timed CSP and are expressed as formulas in branching-time temporal logic. The traditional model checking algorithm is used to verify the temporal logic formulas of test cases on the model of a test verification system. We applied the verification methodology to a test case of the INRES protocol. A few safety errors were detected in the test case. The more important thing was the detection of the liveness error, that is the detection that the dynamic behavior of the test case did not satisfy the purpose of the test case for a Pass test verdict to be assigned.

References

[BS93] U. Bar and J.M. Schneider. Automated validation of TTCN test suites. In *IFIP PSTV XII*. North-Holland, 1993.

[CCI92] CCITT. *CCITT Specification and Description Language (SDL)*, pages 1-219. CCITT Recommendation Z.100, 1992.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8:244-263, April 1986.

[DB90] M. Dubuc and G. v. Bochmann. Translation from TTCN to LOTOS and the validation of test cases. In *FORTE-90*, pages 141-155. North-Holland, 1991.

[ISO91] ISO. *ISO/IEC 9646: Conformance Testing Methodology and Framework*, ISO/ IEC JTC1/SC21, 1991.

[ALHE91] R. Alur and T.A. Henzinger. Logics and Models of Real Time: A Survey. In *LNCS 600*, pp. 74-106, 1991.

[BHS91] F. Belina, D. Hogrefe, and A. Sarma. SDL with Applications from Protocol Specification. Prentice-Hall, 1991.

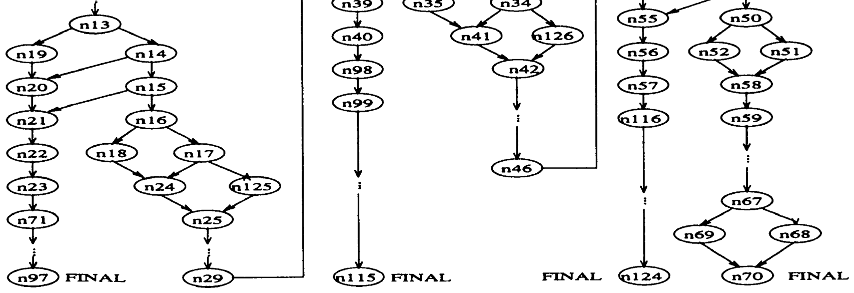
[KR93] A. Kay and J.N. Reed. A relay and guarantee method for timed CSP: A specification and design of a telephone exchange. *IEEE Trans. on Software Eng.*, 19(6):625-639, June 1993.

[NS93] K. Naik and B. Sarikaya. Test case verification by model checking. *Formal Methods in Systems Design*, 2(3):277-321, 1993.

[OST90] J.S. Ostroff. Deciding Properties of timed transition models. *IEEE Trans. on Parallel and Distributed Systems*, 1(2):170- 183, April 1990.

[SAR93] B. Sarikaya. *Principles of Protocol Engineering and Conformance Testing.* Simon and Schuster, September 1993.

APPENDIX A. Global State Space


n1
INITIAL
n2
n5
n7
n6
n8
n9
n13
n19
n14
n20
n15
n21
n16
n22
n18
n17
n23
n24
n125
n71
n25
n97
FINAL
n29
n30
n36
n31
n37
n32
n38
n33
n39
n35
n34
n40
n41
n126
n98
n42
n99
n46
n115
FINAL
n47
n53
n48
n54
n49
n55
n50
n56
n52
n51
n57
n58
n116
n59
n67
n69
n68
FINAL
n124
n70
FINAL