A Proposal for a Real-Time Extension of TTCN

Thomas Walter[1] and Jens Grabowski[2]

TIK-Report No. 20 (Version 2)
December 1996

[1]Eidgenössische Technische Hochschule Zürich, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich/Gebäude ETZ, 8092 Zürich, Schweiz, e-mail: walter@tik.ee.ethz.ch

[2]Medizinische Universität zu Lübeck, Institut für Telematik, Ratzeburger Allee 160, 23538 Lübeck, Deutschland, e-mail: jens@itm.mu-luebeck.de

Abstract

In this report we propose an extension of TTCN (Tree and Tabular Combined Notation) to *real-time TTCN*. The extension is defined on a syntactical and semantical level. Syntactically, we provide facilities to annotate TTCN statements with two time values, namely an earliest execution time (EET) and a latest execution time (LET). The informal interpretation of these time values is that a TTCN statement may be executed if it has been continuously enabled for at least EET units and it must be executed if it has been continuously enabled for LET units. The operational semantics of real-time TTCN is defined by means of timed transition systems. In timed transition systems an execution of a system is modelled by a timed state sequence which counts for time (progress of time) and state (execution of TTCN statements) activities. We define a mapping of real-time TTCN to timed transition systems and give examples in order to show the applicability of our approach.

Chapter 1

Introduction

TTCN (Tree and Tabular Combined Notation) [14] is a notation for the definition of conformance test suites for OSI (Open Systems Interconnection) protocol specifications. Test suites are used for ensuring that different implementations of the same protocol specification are checked for the same set of requirements [7, 12, 13].

Test suites are collections of test cases where each test case is defined with a specific test purpose in mind. Test cases are specified as sequences of test events. Essentially, test events are input and output events of abstract service primitives (ASP) or protocol data units (PDU). A test case describes how a tester should drive an implementation under test (IUT) through a sequence of test events in order to reach the test purpose. The relative ordering of test events is defined in a *behaviour description*. A behaviour description may also include tester specific events, e.g., initialization of variables or start of timers.

Although TTCN provides a timer mechanism which allows to set timers and to check their status, the absolute and relative timing of events cannot be specified. The TTCN timer mechanism might be sufficient for functional tests of traditional OSI protocols, but it is insufficient for testing *non-functional* requirements of multimedia and real-time communication protocols.

Due to the increasing dissemination of multimedia applications, testing of the corresponding protocol implementations will become an issue. We propose a real-time extension of TTCN so that TTCN can be used in testing multimedia and real-time communication protocols.

Our extension of TTCN to *real-time TTCN* is on a syntactical and a semantical level. In particular, the syntactical difference is that for real-time TTCN we allow an annotation of test events with an earliest execution time (EET) and a latest execution time (LET). Informally, a test event may be executed if it has been continuously enabled for at least EET time units and it must be executed if it has been continuously enabled for LET time units. Test events are executed instantaneously. For the definition of an operational semantics of real-time TTCN we adopted timed transition systems [11].

A number of formal description techniques for the specification of real-time constraints have been proposed: time Petri Nets [3, 20], LOTOS [2, 4, 10, 16, 17, 21, 24], SDL [4, 10, 18] and ESTELLE [4, 8]. From the cited literature it can be concluded that two directions of how to cope with real-time exist. Firstly, syntactical and semantical extensions of a specific technique are defined as, for instance, [16, 17, 21, 24] for LOTOS and [8] for ESTELLE. Secondly, some other proposals define an integration of a formal description technique and a real-time logic as, for instance, [18] for SDL and [2] for LOTOS. As in the cited literature, our approach allows the timing of actions relative to the occurrence of previous actions. The difference is that the former are used for the specification of functional and real-time requirements of systems whereas our emphasis is on testing real-time requirements. Real-time TTCN is used for the specification of properties of a test system and requirements on an IUT.

In this report we focus on mechanisms for the specification of real time requirements in test cases. Other important issues like test suite validation, test realization and tool support are for further study.

The report is structured as follows: Section 2 gives a brief introduction to TTCN. Section 3 introduces real-time TTCN. The feasibility of our approach is shown in Section 4. Section 5 concludes the report with an assessment of our approach and the identification of open issues.

Chapter 2

TTCN - Tree and Tabular Combined Notation

TTCN is a notation for the description of test cases to be used in conformance testing. TTCN provides two syntactical forms, TTCN/MP as a machine processable (i.e., pure textual) form, and TTCN/GR as a graphical representation. For the purpose of this report we mainly restrict our attention to TTCN/GR and TTCN concepts related to the description of the dynamic test case behaviour. Further details on TTCN can be found in [14, 19, 23, 25].

2.1 Abstract Testing Methods and TTCN

A test case specifies which outputs from an implementation under test (IUT) can be observed and which inputs to an IUT can be controlled. Inputs and outputs are either *abstract service primitives* (ASPs) or *protocol data units* (PDUs).

An example of an *abstract test method* [13], the *Multi-Party Testing Context*, is shown schematically in Fig. 2.1. Abstract testing functions, such as *lower tester* (LT), *upper tester* (UT) and *lower tester control function* (LTCF), are the active components in an abstract test method. Figure 2.1 includes three LTs, three UTs and one LTCF. LTs and UTs control and observe the IUT at *points of control and observation* (PCOs) which are interfaces above and below the IUT. The LTCF is responsible for the creation and coordination of LTs and UTs. LTs, UTs and LTCF are referred to as test components (TCs). They run in parallel. TCs are interconnected by *coordination points* (CPs) through which they exchange *coordination messages* (CMs). The rules that govern the exchange of CMs and the coordination between LTs and UTs are defined in *test coordination procedures* (TCPs). LT and IUT logically communicate by exchanging PDUs which are embedded in ASPs exchanged at PCOs. Since in most cases the lower boundary of an IUT does not provide adequate PCO interfaces, LTs and IUT communicate by using services of an underlying service provider.

PCOs and CPs are based on the same abstract model: a pair of unbounded FIFO queues (one for each direction of communication) which allow an asynchronous exchange of ASPs and CMs.

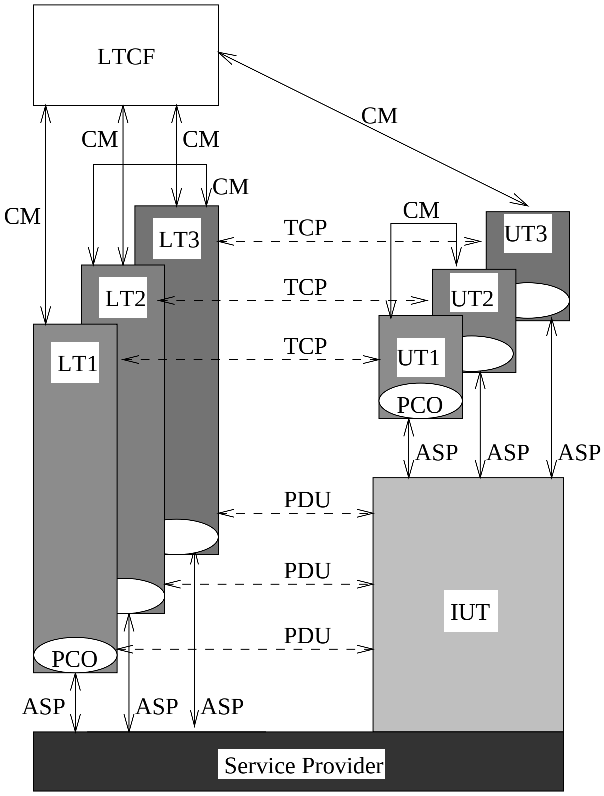

Figure 2.1: Multi-Party Testing Context [13]

2.2 Test Case Dynamic Behaviour Descriptions

A TTCN test case defines the behaviour of TCs during test execution. The description consists of *statements* and *verdict assignments*. A verdict assignment is a statement concerning the conformance of an IUT with respect to the sequence of events that have been performed. A PASS verdict is assigned if the IUT passes the test, FAIL is given if the IUT contradicts the specification, and INCONCLUSIVE is assigned if neither a PASS nor a FAIL verdict can be assigned. TTCN statements are *test events*, *constructs* and *pseudo events*.

- Test events are SEND, IMPLICIT SEND, RECEIVE, OTHERWISE, TIMEOUT and DONE. SEND and IMPLICIT SEND specify the sending of ASPs, PDUs and CMs. The event L ! N-DATA request on line 6 of Fig. 2.2 describes the sending of N-DATA request via PCO L. RECEIVE and OTHERWISE denote the processing of received ASPs, PDUs and CMs. The statement CP ? CM on line 1 of Fig. 2.2 specifies the reception of coordination message CM at coordination point CP. TIMEOUT events check for the expiration of a timer, e.g., line 5 in Fig. 2.2. DONE is used to check whether TCs have terminated. Test events may be qualified and/or followed by assignments and timer operations.

- Constructs are CREATE, ATTACH, ACTIVATE, RETURN, GOTO and REPEAT. CREATE specifies the creation of a TC. The created TC executes in parallel with all other running TCs. ATTACH is a construct which allows to transmit control to a sub-behaviour description, called *test step*. The mechanism is comparable to the procedure concept in programming languages.

Test Case Dynamic Behaviour					
Nr	**Label**	**Behaviour Description**	**CRef**	**V**	**Comments**
1		CP ? CM	connected		RECEIVE
2		(NumOfSends := 0)			Assignment
3		REPEAT SendData			Construct
		UNTIL [NumOfSends > MAX]			
4		START Timer			Timer Operation
5		?TIMEOUT timer			
6		L ! N-DATA request	data		SEND

Figure 2.2: TTCN Behaviour Description - Sequence of Statements

Test Case Dynamic Behaviour					
Nr	**Label**	**Behaviour Description**	**CRef**	**V**	**Comments**
1		[TRUE]			Qualifier
2	L1	(NumOfSends := NumOfSends + 1)			
3		+SendData			ATTACH
4		[NOT NumOfSends > MAX]			Alternative 1
5		-> L1			GOTO
6		[NumOfSends > MAX]			Alternative 2

Figure 2.3: TTCN Behaviour Description - Set of Alternatives

ACTIVATE and RETURN deal with *default behaviour descriptions*. Usually, a default behaviour description handles all incoming events which are not treated in the main behaviour description. ACTIVATE allows to change the default behaviour during the test run and RETURN allows to return from a default behaviour back to the main description. GOTO transfers control to a specified statement. REPEAT is used for the specification of loops.

- Pseudo-events are qualifiers (i.e. Boolean expressions), assignments and timer operations.

Statements can be grouped into *statement sequences* and *sets of alternatives*. In TTCN/GR, sequences of statements are represented one after the other on separate lines and being *indented* from left to right. In Fig. 2.2 the statements on lines 1 - 6 are a statement sequence. Statements on the same level of indentation and with the same predecessor are a set of alternatives. In Fig. 2.3 the statements on lines 4 and 6 form a set of alternatives. They are on the same level of indentation and have the statement on line 3 as their common predecessor.

2.3 Test Case Execution and Test Component Execution

Test case execution starts with only the main test component running. The main test component which fulfils the role of the LTCF, creates all other TCs. Immediately after creation the TC starts executing its initially assigned behaviour description.

The execution of a behaviour description starts with the first *level of indentation*, e.g., in Fig. 2.2 the test event on line 1, and proceeds towards the last level of indentation of the behaviour description, e.g., the statements on lines 2, 3, 4, 5, until finally the statement on line 6 is executed. If on a level of indentation a set of alternatives is found, only one alternative is executed and test case execution proceeds with the next level of indentation relative to the executed alternative. For example, in Fig. 2.3 the statements [NOT NumOfSends > MAX] (line 4) and [NumOfSends > MAX] (line 6) are alternatives. If the statement on line 4 is executed, processing continues with the statement on line 5. During execution each TC maintains its own set of local variables and may make use of a number of implicitly defined variables. Execution of a behaviour description stops if the last level of indentation is visited, a test verdict is assigned or a test case error occurs.

Whenever a new level of indentation is reached that level is *expanded.* This means that default behaviour descriptions, ATTACH statements and REPEAT loops are transformed in such a way that the set of alternatives only consists of test events, pseudo-events, GOTOs, CREATEs, ACTIVATEs and RETURNs.

Before a set of alternatives is evaluated, a *snapshot* is taken [14], i.e., the state of the TC and the state of all PCOs, CPs and timers related to the TC are updated and frozen for the time period the set of alternatives is evaluated. This guarantees that evaluation of a set of alternatives is an *atomic* and *deterministic action.*

Alternatives are evaluated in sequence and the first alternative which is *evaluated successfully* is executed. Then execution proceeds with the set of alternatives on the next level of indentation. If no alternative can be evaluated successfully, a new snapshot is taken and evaluation of the set of alternatives is started again. A statement is evaluated successfully under the following conditions:

- IMPLICIT SEND, CREATE, GOTO, ACTIVATE and RETURN are always evaluated successfully.
- SEND, an assignment and a timer operation are always evaluated successfully provided that the optional qualifier is *true.*
- RECEIVE is evaluated successfully if a matching ASP, PDU or CM is available at the specified PCO or CP and if the optional qualifier is *true.*
- OTHERWISE is evaluated successfully if an input is available at the specified PCO and if the optional qualifier is *true.*
- TIMEOUT is evaluated successfully if the specified timer has expired and if the optional qualifier is *true.*
- DONE is evaluated successfully if the specified test components have terminated and if the optional qualifier is *true.*
- Assignments and timer operations are always evaluated successfully provided that the optional qualifier is *true.*
- A qualifier (not in combination with a test event or an assignment or timer operation) is evaluated successfully if it is *true.*

2.4 TTCN and Real-Time Constraints

In TTCN no explicit time model is assumed in the sense that no predictions of the execution time of TTCN statements or the transmission times of ASPs, PDUs and CMs can be made. Only timers and the corresponding TIMEOUT event are a means for specifying real-time behaviour in TTCN. However, as stated in [14], a test case should be defined such that the relative speed of the systems executing the test case does not have an impact on the test result.

Timers can be started (with a timeout value from picoseconds to minutes), can be stopped and timer values can be read. The status of a timer can be checked in a set of alternatives using the TIMEOUT event. But, whenever a timer expires this has no immediate influence on the execution of a test component. If a timer expires while evaluation of a set of alternatives is in progress, expiration of that timer is not visible until the next snapshot is taken. An immediate reaction on the timeout event is not possible. Depending on the ordering of alternatives an expired timer may get undetected at all.

Chapter 3

Real-Time TTCN

This section discusses our proposal of an extension of TTCN to real-time TTCN. This includes syntactical changes and the definition of an operational semantics. For the latter we define a mapping of real-time TTCN to timed transition systems [11]. Our choice of timed transition systems has been inspired by our work on the definition of an operational semantics for TTCN in terms of (untimed) transition systems [6, 27, 28].

3.1 Timed Transition Systems

As stated in the literature [1, 3, 11, 20], real-time behaviour of systems can be expressed by assuming that execution of events is restricted by a finite interval of earliest and latest execution times and which assume that execution of events is instantaneous. In our approach we use timed transition systems for modelling real-time behaviour. In this section we quote the main definitions of [11].

A *transition system* [15, 22] consists of a set V of variables, a set Σ of states, a subset $\Theta \subseteq \Sigma$ of initial states and a finite set $\mathcal{T}$ of transitions which also includes the idle transition t_I. Every transition $t \in \mathcal{T}$ is binary relations over states; i.e., it defines for every state $s \in \Sigma$ a possibly empty set $t(s) \subseteq \Sigma$ of so-called t-successors. A transition t is said to be *enabled* on state s if and only if $t(s) \neq \emptyset$. For the idle transition t_I we have that $t_I = \{(s, s) \mid s \in \Sigma\}$.

An infinite sequence $\sigma = s_0 s_1 \ldots$ is a *computation* of the underlying transition system if $s_0 \in \Theta$ is an initial state, and for all $i \geq 0$ there exists a $t \in \mathcal{T}$ such that $s_{i+1} \in t(s_i)$, denoted $s_i \xrightarrow{t} s_{i+1}$, i.e., transition t is *taken* at position i of computation σ.

The extension of transition systems to timed transition systems is that we assume the existence of a real-valued global clock and that a system performs actions which either advance time or change a state [11]. Actions are executed instantaneously, i.e., they have no duration.

A *timed transition system* consists of an underlying transition system and, for each transition $t \in \mathcal{T}$, an earliest execution time $EET_t \in \mathbb{N}$ (with $\mathbb{N}$ the natural numbers including zero) and a latest execution time $LET_t \in \mathbb{N} \cup \{\infty\}$ is defined. We assume that $EET_t \leq LET_t$ and, by default, EET_t is zero and LET_t is ∞. For a transition t which is enabled on an initial state we have $LET_t = \infty$ and so for the idle transition t_I: $LET_{t_I} = \infty$. EET_t and LET_t define timing constraints which ensure that transitions cannot be performed neither to early (EET_t) nor too late (LET_t).

A *timed state sequence* $\rho = (\sigma, T)$ consists of an infinite sequence σ of states and an infinite sequence T of times $T_i \in \mathbb{R}$ (with $\mathbb{R}$ the real numbers) and T satisfies the following two conditions:

- *Monotonicity:* $\forall i \geq 0$ either $T_{i+1} = T_i$ or $T_{i+1} > T_i \wedge s_{i+1} = s_i$.
- *Progress:* $\forall t \in \mathbb{R}\, \exists\, i \geq 0$ such that $T_i \geq t$.

Monotonicity implies that time never decreases but possibly increases by any amount between two neighbouring states which are identical. If time increases this is called a *time step*.[1] The transition being performed in a time step is the idle transition which is always enabled (see above). The progress condition states that time never converges, i.e., since $\mathbb{R}$ has no maximal element every timed state sequence has infinitely many time steps. Summarizing, in timed state sequences state activities are interleaved with time activities. Throughout state activities time does not change, and throughout time steps the state does not change.

A timed state sequence $\rho = (\sigma, T)$ is a *computation* of a timed transition system if and only if state sequence σ is a computation of the underlying transition system and for every transition $t \in \mathcal{T}$ the following requirements are satisfied:

- for every transition $t \in \mathcal{T}$ and position $j \geq 0$ if t is taken at j then there exists a position i, $i \leq j$ such that $T_i + EET_t \leq T_j$ and t is enabled on $s_i, s_{i+1}, \ldots, s_{j-1}$ and is not taken at any of the positions $i, i+1, \ldots, j-1$, i.e., a transition must be continuously enabled for at least EET_t time units before the transition can be taken.
- for every transition $t \in \mathcal{T}$ and position $i \geq 0$, if t is enabled at position i, there exists a position j, $i \leq j$, such that $T_i + LET_t \geq T_j$ and either t is not enabled at j or t is taken at j, i.e., a transition must be taken if the transition has been continuously enabled for LET_t time units.

A finite timed state sequence is made infinite by adding idle transitions, so that we have an infinite sequence of time activities.

In order to ensure *operationality* of timed transition systems it is required that time can progress. This is achieved simply by putting a restriction on transitions with $LET = 0$. Let $\mathcal{T}_0 \subseteq \mathcal{T}$ be the subset of transitions with $LET = 0$. There should be no sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} s_n$ of states and transitions such that $n > \mid \mathcal{T}_0 \mid$ (where $\mid \mathcal{T}_0 \mid$ denotes the cardinality of set $\mathcal{T}_0$) and $t_i \in \mathcal{T}_0$ for all $0 \leq i < n$. In other words, computations are *fair* with respect to time activities.

3.2 Syntax of Real-Time TTCN

For our real time extension of TTCN we add time information in the declarations and the dynamic part of a TTCN test suite. In the declarations part we specify names for time values and units to be used in TTCN behaviour descriptions. In the dynamic part we add concrete EET and LET values to behaviour lines of test cases, test steps and default behaviour descriptions.

3.2.1 Extensions of the Declarations Part.

For the specification of EET and LET values and time units we introduce an Execution Time Declarations table (Fig. 3.1) and the keywords EET and LET.[2] The Time Name column can be used to declare names for EET and LET values.

[1] If time does not increase between two identical states, i.e., $(s_i, T_i) \longrightarrow (s_i, T_i)$, then this is called a ***stutter step***. For an interpretation of stutter steps see [11].

[2] We use different fonts for distinguishing between syntax, i.e., EET and LET, and semantics, i.e., EET and LET.

Execution Time Declarations			
Time Name	**Value**	**Unit**	**Comments**
EET	1	s	EET value
LET	1	min	LET value
WFN	5	ms	Wait For Nothing
NoDur		min	No specified value

Figure 3.1: Execution Time Declarations Table

The names can be used instead of concrete values within behaviour description tables. Value and Unit columns are used to associate a time value and a time unit to a time name. As time units we allow ps (picosecond), ns (nanosecond), us (microsecond), ms (millisecond), s (second) and min (minute), i.e., the time units already used in TTCN. The default time unit is ms (millisecond). Time values are converted to the default time unit whenever necessary, e.g., before time values are evaluated in a behaviour description.

EET and LET are predefined variables with 0 and ∞ as initial values. These initial values can be overwritten as shown in Fig. 3.1. Apart from column headings the table looks much like the TTCN Timer Declarations table.

Figure 3.1 includes a time name declaration NoDur for which only the time unit min but no value is given. In this case, a value has to be provided during test case execution by means of an assignment. If a name is evaluated and no value is assigned the test case will end with a dynamic test case error.

Due to practical reasons it is not appropriate to require that for each TTCN statement EET and LET values are specified. In this case the default values for EET $(= 0)$ and LET $(= \infty)$ are used. Additionally, we allow to overwrite these default values within an Execution Time Declarations table. For changing the default time values the keywords EET and LET are used. In Fig. 3.1, the default values for EET and LET are changed to 1 second and 1 minute, respectively.

Besides the static declarations of time values, a change of these values is also allowed within a behaviour description. During a test run time values can be changed by means of assignments. We only require that concrete EET and LET values can be determined when the corresponding TTCN statement is evaluated successfully and that the condition $0 \leq EET \leq LET$ holds. In all other cases the test case will end with a dynamic test case error.

Examples for the dynamic change of time values within a behaviour description can be found in Fig. 3.2. On line 2 the value 3 is assigned to the time name NoDur and on line 4 the default LET value is changed. This change becomes effective on the next level of indentation, i.e., for the TTCN statements on lines 5 and 6. As shown on line 6 it is also allowed to refer to default time values explicitly by using the keyword LET.

3.2.2 Extensions of the Dynamic Part.

The changes within the dynamic part of a TTCN test suite are related to behaviour lines in Test Case Dynamic Behaviour, Default Dynamic Behaviour and Test Step Dynamic Behaviour tables. The structure of the behaviour lines is the same in all these tables. Therefore, we discuss the syntactical changes by using a Test Case Dynamic Behaviour table only.

As indicated in Fig. 3.2 we add a Time column. An entry in the Time column specifies EET and LET values for the corresponding behaviour line. Entries may be variables or constants, e.g., the entry in Fig. 3.2 line 1 sets $EET = 2$ and $LET = 4$ with default time unit ms. Within the Time column EET and LET may also be specified by means of name references which have to be looked up within the declarations part of the test suite. For instance, on line 3 of Fig. 3.2 the time name NoDur is used. NoDur has been declared in table Fig. 3.1 and has been assigned a value on line 2 of Fig. 3.2.

In the TTCN BNF definition ([14] Annex A) the time annotation of a behaviour line has to be considered. We add the non-terminal Time to this rule and need three additional rules to define the syntax of time annotations (Fig. 3.3).

3.3 Operational Semantics of Real-Time TTCN

The operational semantics of real-time TTCN is defined in two steps: Firstly, we define the semantics of a TC in terms of a timed transition system which has the real numbers $\mathbb{R}$ as *abstract* time domain (in contrast to the *concrete* time domain in the syntactical extension of TTCN described in the previous section). Secondly, we extend this definition so that the behaviour of several concurrent TCs is modelled.

3.3.1 Operational Semantics of a Real-Time Test Component.

With a given definition of a TC we associate the following timed transition system: A state $s \in \Sigma$ of a TC is given by a mapping of *variables* to *values*. The set of variables V includes all variables defined for the TC in the test suite and, additionally, a variable for each timer. Furthermore, we introduce a *control variable* π which indicates the location of control in the behaviour description of the TC. π is updated when a new level of indentation is visited. We even let PCOs and CPs be pairs of variables so that each holds a queue of ASPs, PDUs or CMs sent and received, respectively. PCO and CP variables and timer variables are *shared* variables which can also be accessed from the environment of a TC. For instance, received ASPs are put on the corresponding PCO variable by the environment. Similarly, when a timer expires the value of the corresponding timer variable is updated by the environment. The environment performs its activities concurrently to the execution of the TC.

The initial state of a TC is the state where all variables having assigned their initial values (if specified) or being undefined. All PCO and CP variables have assigned an empty queue and all timer variables have assigned the value stop. The control variable π is initialized to the first level of indentation. If the TC is not running, i.e., the TC has not been created yet, then all variables are undefined.

The set $\mathcal{T}$ of transitions contains a transition for every TTCN statement in the TC behaviour description and the idle transition t_I. Furthermore, we have a transition t_E which models activities of the environment, i.e., reception of an ASP or expiration of a timer. t_E is not performed by the TC but may change the state

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1		2, 4	A ? DATA request			
2			(NoDur := 3)			Time assignment
3		2, NoDur	A ! DATA ack			
4			(LET := 50)			LET update (ms)
5			A ? Data request			
6		WFN, LET	B ? Alarm			

Figure 3.2: Adding EET and LET values to behaviour lines

BehaviourLine ::=	**$BehaviourLine** LabelId Line Time Cref VerdictId [Comment] **$End_BehaviourLine**
Time ::=	**$Time** [TimeAnnotation]
TimeAnnotation ::=	TimeId **,** TimeId
TimeId ::=	Value | Identifier | **EET** | **LET**

Figure 3.3: BNF rules for behaviour lines of real time TTCN

of the TC, because a shared PCO, CP or timer variable is updated.

In the following we assume that the currently visited level of indentation has been expanded as defined in Annex B of [14] and has the following general form: $A_1[EET_1, LET_1], \ldots, A_n[EET_n, LET_n]$, where A_i denotes an alternative and EET_i, LET_i denote the earliest and latest execution times of alternative A_i.

We say that $A_i[EET_i, LET_i]$ is *potentially enabled* if $A_i[EET_i, LET_i]$ is in the set of alternatives. $A_i[EET_i, LET_i]$ is *enabled* if $A_i[EET_i, LET_i]$ is evaluated successfully (Sect. 2.3), $A_i[EET_i, LET_i]$ is *executable* if $A_i[EET_i, LET_i]$ is enabled and $A_i[EET_i, LET_i]$ has been potentially enabled for at least EET_i.

The mapping described above defines the set of possible computations of a TC as a set of timed state sequences with *potentially enabled* substituted for *enabled* in the definitions of Sect. 3.1. If an alternative cannot be successfully evaluated within LET time units then test case execution stops with an error indication. To make the evaluation of a real-time TTCN behaviour description more explicit we introduce the following refined snapshot semantics (Sect. 2.3). For the rest of this section we let $T, T', T'' \in \mathbb{R}$ with $T \leq T' \leq T''$.

1. The TC is put into its initial state.

2. If the level of indentation is visited for the first time then all alternatives are marked *potentially enabled* and the global time T is saved. The state of PCO and CP variables and expired timer variables is locked, so that they cannot be updated by the environment.

   If for an $A_i[EET_i, LET_i]$ in $A_1[EET_i, LET_i], \ldots, A_n[EET_i, LET_i]$, $LET_i < T' - T$, where T' the current global time and T the time when the alternative has been marked potentially enabled, then test case execution stops.

3. All alternatives which can be evaluated successfully are marked *enabled*. If no alternative in the set of alternatives can be evaluated successfully then PCO, CP and timer variables are unlocked (and the environment may again update these variables). Processing continues with Step 2.

4. An enabled alternative $A_i[EET_i, LET_i]$ is marked *executable* provided that $EET_i \leq T' - T \leq LET_i$ and if there is another enabled alternative $A_j[EET_j, LET_j]$ with $EET_j \leq T' - T \leq LET_j$ then $i < j$, i.e., the i-th alternative comes before the j-th alternative in the set of alternatives.

   If no alternative can be marked executable then PCO, CP and timer variables are unlocked. Processing continues with Step 2.

5. The alternative $A_i[EET_i, LET_i]$ marked executable in Step 4 is executed and control variable π is updated to the next level of indentation. PCO, CP and timer variables are unlocked. A test case terminates if the last level of indentation is reached, otherwise evaluation continues with Step 2.

Remarks:

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1		2, 4	PCO1 ? N-DATA indication	info		
2			...			next level
3		2, 4	PCO2 ? N-ABORT indication	abort		
4			...			next level

Figure 3.4: Partial Real-Time TTCN Behaviour Description

- If a new level of indentation is visited for the first time (Step 2) then all alternatives become *potentially enabled* and time starts running although some alternatives may wait for some further conditions to become fulfilled.

  If a potentially enabled alternative cannot be evaluated successfully within the specified earliest and latest execution times then this is an indication that a specified real-time constraint has not been met and that test case execution should stop.

- If no alternative can be evaluated successfully (Step 3) then a next iteration of Steps 2 - 5 must be performed. But before, PCO, CP and timer variables are unlocked.

- In Step 4, the selection of alternatives for execution from the set of executable alternatives follows the same rules as in TTCN [14].

- If a TC stops (Step 5) then the finite timed state sequence is extended to an infinite sequence by adding an infinite sequence of idle transitions.

- Every iteration of Steps 2 - 5 is *atomic*. This complies with the snapshot semantics of TTCN [14].

Example 1 We consider the partial behaviour description in real-time TTCN given in Fig. 3.4. Assuming that the level of indentation with the two alternatives on lines 1 and 3 has been visited for the first time at T. The first alternative may be executed in the interval $EET_1 = 2$ and $LET_1 = 4$ provided that a N-DATA indication with data info has been received at PCO PCO1. Furthermore, let us assume that at T' an N-DATA indication is received. Then, the first alternative may be executed at T'' with $EET_1 \leq (T'' - T) \leq LET_1$, because this alternative is enabled (Step 3) and is executable (Step 4) and no other alternative is executable (no N-ABORT indication has been received yet). A corresponding computation might be:

$$\ldots \longrightarrow (s, T) \xrightarrow{t_I} (s, T') \xrightarrow{t_E} (s', T') \xrightarrow{t_I} (s', T'') \xrightarrow{t_1} (s'', T'') \longrightarrow \ldots$$

The reception of an N-DATA indication at time T' is a state activity, $(s, T') \xrightarrow{t_E} (s', T')$, because a PCO variable is updated by the environment performing transition t_E.

Supposing that an N-DATA indication and an N-ABORT indication have been received from the environment at some $T''' \leq T''$. Then, although both alternatives are executable, the first alternative is executed according to Step 4 because of the ordering of alternatives in the set of alternatives.

If no N-DATA indication and no N-ABORT indication have been received before LET_1 or LET_2 time units after the alternatives have been potentially enabled then test case execution stops (Step 2).

□

3.3.2 Operational Semantics of a Real-Time Test System.

In general, more than one TC participates in the execution of a test case and, because of the multiplicity of executable alternatives, several TTCN statements may be executed in parallel. In timed transition systems the parallel behaviour of TCs is modelled as finite sequences of state activities which are not interleaved with time activities. Each state activity is performed by another test component.

Example 2 We assume a test system of n active or running TCs $TC_1, \ldots, TC_n$. Each TC has its own processor. Given these assumptions we associate the following timed transition system with the test system:

$V = V_1 \cup \ldots \cup V_n$ where $V_i \cap V_j = \emptyset$ for $0 \leq i, j \leq n$ and $i \neq j$, i.e., the set of variables is the union of the set of variables of all TCs. Σ contains all interpretations of V, i.e., the mapping from variables to values. The initial state ($\in \Theta$) of the test system is the one where only the MTC is initialized. The set $\mathcal{T}$ of transitions consists of the idle transition t_I and the environment transition T_E plus the sets of transitions of all TCs labelled with the corresponding time values EET and LET of TTCN statements.

Assuming that during a test run every TCs has an executable alternative ready at time T then execution of all executable alternatives may yield the following computation:

$$\begin{array}{l} \ldots (s_0, T) \xrightarrow{t_1} (s_1, T) \xrightarrow{t_2} \ldots \\ \xrightarrow{t_{n-1}} (s_{n-1}, T) \xrightarrow{t_n} (s_n, T) \ldots \end{array}$$

i.e., a sequence of n state activities. Note that, scheduling the executable alternatives in a different order would have yielded another computation.

$\square$

The model described above which assumes that a processor is available for every TC, is termed *multiprocessing model* in [11]. In a *multiprogramming model*, a single processor is shared among a number of TCs. All TC which are running on the same processor and which have a transition executable have to be scheduled for execution. Fortunately, as shown in [11], the semantics of the multiprogramming model can also be defined in terms of timed transition systems. The only additional constructs necessary are a special processor control variable μ which holds the identifier of the currently executing TC, and a scheduling transition t_S that changes the status of TCs by resuming a temporarily suspended TC. Executing the scheduling transition is a state activity that changes the processor control variable μ. In the initial state of a test system, the variable μ is assigned with the identifier of the MTC. In a computation of a test system scheduling transitions are interleaved with state activities and time steps. Unlike as in [11], scheduling another TC does not preempt any potentially enabled, enabled or executable transition of any other TC. For a the test system satisfies the defined real-time constraints, the system must be sufficiently fast and scheduling of TC must be done properly.

3.4 Discussion of the Proposal

If we assume that no time values are defined (in this case EET and LET are set to zero and ∞, respectively) then execution of a test case results in the same sequence of state-transitions as in TTCN. In this sense our definition of real-time TTCN is downwards compatible.

Execution of a statement is modelled as an instantaneous state change. However, it is common knowledge that execution of a statement has a finite duration. The point we would like to emphasize is that during execution of a statement a state becomes “transient” in the sense that the result of executing a statement is not available (or observable) immediately. If execution of a statement has come to an end then the result becomes permanent (or observable). In the approach employed, time instance when processing of a statement has terminated is recorded.

Real-time TTCN combines property and requirement oriented specification styles. Time labels for TTCN statements, in general, define real-time constraints for the test system. The test system is assumed to be sufficiently fast so that a correctly behaving test system complies with the properties defined in the real-time TTCN behaviour description. Time labels for RECEIVE and OTHERWISE events, which imply a communication with the IUT, define requirements on the IUT and the underlying service provider. As well as the test system, the underlying service provider is assumed to be sufficiently reliable particularly with respect to the timing of activities. Therefore, if a timing constraint of a RECEIVE or OTHERWISE event is violated, this clearly is an indication that the IUT is faulty and the test run should end with a FAIL verdict assignment.

Chapter 4

An Application of Real-Time TTCN

Before discussing a concrete example we give an idea how real-time TTCN can be used in the description of real-time constraints.

4.1 Specifying Real-Time Constraints with Real-Time TTCN

Figure 4.1 is an example of (partial) behaviour description for defining two real-time constraints. Firstly, every two time units the test system should generated an N-DATA request (lines 1 and 2). Secondly, the test system should receive every two to three time units a T-DATA indication (lines 3 and 4). Assuming a constant delay for the transmission of PDUs from an LT to the IUT the second constraint implies the requirement on the IUT that the IUT is capable of generating every two to three time units an T-DATA indication.

A TTCN behaviour description almost equivalent to the real-time TTCN behaviour description for the second constraint is shown in Fig. 4.2. The first timer is used for the lower execution time and the second timer is used to control the latest execution time. If the second TIMEOUT event can be observed this is an indication for a erroneous behaviour of the IUT. This TTCN behaviour description, however, only is correct under the assumption that the test system is *infinitely* fast so that no extra delay is introduced due to the execution of TTCN statements. In real-time TTCN all assumptions which are to be fulfilled by a test system are made explicit. Besides its conciseness (compare Figs. 4.1 and 4.2) this is a further advantage.

Although the behaviour description shown in Fig. 4.3 is legal (syntactically correct), the assignment of time labels to TTCN statements is not appropriate. If the

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1	L1	2, 2	PCO ! N-DATA request	req		
2			-> L1			GOTO
3	L2	2, 3	PCO ? T-DATA indication	ind		
4			-> L2			GOTO

Figure 4.1: Continuous Sending and Receiving in Real-Time TTCN

Test Case Dynamic Behaviour					
Nr	**Label**	**Behaviour Description**	**CRef**	**V**	**Comments**
1	L2	START timer(2)			
2		?TIMEOUT timer			
3		START timer(1)			
4		PCO ? T-DATA indication	ind		
5		STOP timer			
6		-> L2			
7		?TIMEOUT timer		FAIL	

Figure 4.2: Time constraints in TTCN for Continuous Receiving

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1		6, 8	PCO1 ? N-DATA indication	data		
2			...			next level
3		2, 4	PCO2 ? N-ABORT indication	abort		
4			...			next level

Figure 4.3: Non-usage of Real-Time TTCN

first RECEIVE event becomes enabled but is not executable, and subsequently the second RECEIVE becomes enabled and executable then the second alternative is executed. In TTCN the first alternative would have been executed immediately after it has been enabled. Thus, the assignment of time labels to TTCN statements has to be done carefully because it may change the execution order of alternatives.

The behaviour description shown in Fig. 4.4 is not legal because the *operationality* requirement is not fulfilled. As soon as the SEND event on line 2 becomes *potentially enabled* it must be immediately taken because it is executable and has been enabled for *LET* time units. The GOTO on the next level of indentation returns location of control to the SEND event which is executed again and again ... Thus, we get an infinite sequence of state activities without any progress of time, i.e., no time steps. Consequently, the sequence of TTCN statements which forms a loop must contain at least one TTCN statement with a non-zero *LET* value.

4.2 Application of Real-Time TTCN

We discuss an application of real-time TTCN to quality-of-service (QoS) testing [9, 26, 29]. Suppose that for a transport connection a specific *throughput*[1] QoS parameter value has been negotiated. A possible test purpose is that the IUT

[1]Throughput is the ratio of the size of the last received transport service data unit to the time elapsed between the corresponding last and next T-DATA indications (and similar for the sending site) [5].

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1			...			
2	L1	0, 0	PCO ! N-DATA request	data		
3			-> L1			GOTO

Figure 4.4: Illegal Real-Time TTCN Behaviour Description

Test Case Dynamic Behaviour						
Nr	**Label**	**Time**	**Behaviour Description**	**CRef**	**V**	**Comments**
1			(NumOfSends := 0)			Assignment
2			REPEAT SendData (L) UNTIL [NumOfSends > MAX]			
3		10, 12	L ! N-DATA request	req		
4			SendData (PCO)			
5		2, 2	PCO ! N-DATA request	req		

Figure 4.5: Real-Time TTCN Test Case for QoS Testing

should abort the connection if the actual monitored throughput is less than the negotiated throughput.

Let the IUT be the receiving transport protocol implementation. The corresponding real-time TTCN test case is shown in Fig. 4.5. The behaviour description for the UT receiving transport data is similar to the one shown in Fig. 4.1 lines 3 and 4. From the negotiated throughput QoS parameter value we can compute the time interval between successive T-DATA indication ASPs that satisfies the negotiated throughput. The LT transmits transport data with N-DATA request at the computed rate. Delaying an N-DATA request (line 3 in Fig. 4.5) should cause the connection being aborted (not shown in Fig. 4.5).

Chapter 5

Conclusions and Outlook

In this report we have discussed a proposal for a real-time extension of TTCN. The motivation for our work has been given by the demand for a test language that can express real-time constraints. This demand mainly comes from the use of multimedia applications which are quite restrictive with respect to the fulfilment of real-time requirements. Since TTCN cannot express real-time constraints, we have made a proposal for a syntactical and semantical extension of TTCN. On a syntactical level TTCN statements can be annotated by time labels which specify earliest and latest execution times. The operational semantics of our TTCN extension is based on timed transition systems [11]. In the report we have described how real-time TTCN test cases are interpreted in timed transition systems.

In our approach a TTCN statement is annotated by time labels. The advantages of this approach are twofold: Firstly, only a few syntactical changes are necessary. Secondly, the extension of TTCN to real-time TTCN is downwards compatible: If we assume that zero and ∞ are earliest and latest execution times then a computation of a real-time TTCN test case is the same as in standard TTCN. A possible extension of our approach is to allow the annotation of test events, assignments and timer operations that are combined on a single statement line with time labels. A mapping of TTCN to transition systems at that level of detail has been investigated in [27, 28]. This mapping may be further extended and evaluated.

Based on the real-time extension of TTCN as proposed in this report techniques for the analysis of the real-time behaviour of testers against specified test cases are to be defined. For this it seems necessary that the discussion of an operational semantics of real-time TTCN as discussed in Sect. 3.3.2 is being extended. Particularly, the different processing models (multiprocessing and multiprogramming models) have to be refined and, in a second step, the modelling of communication channels (PCOs, CPs and service provider) have to be integrated. Our future work will focus on these aspects.

Acknowledgments: The authors are indebted to Stefan Heymer for proofreading and for his detailed comments on earlier drafts of the report. We are also grateful to the anonymous reviewers providing detailed comments and valuable suggestions which have improved contents and presentation of the paper.

Bibliography

[1] R. Alur, T. Henzinger. *Logics and Models of Real Time: A Survey.* In *Real-Time: Theory in Practice*, Lecture Notes in Computer Science 600, 1991.

[2] H. Bowman, L. Blair, G. Blair, A. Chetwynd. *A Formal Description Technique Supporting Expression of Quality of Service and Media Synchronization.* In *Multimedia Transport and Teleservices*, Lecture Notes in Computer Science 882, 1994.

[3] B. Berthomieu, M. Diaz. *Modeling and Verification of Time Dependent Systems Using Time Petri Nets.* In IEEE Transactions on Software Engineering, Vol. 17, No. 3, March 1991.

[4] G. v. Bochmann, J. Vaucher. *Adding Performance Aspects to Specification Languages.* In Protocol Specification, Testing and Verification VIII, North-Holland, 1988.

[5] A. Danthine, Y. Baguette, G. Leduc, Léonard. *The OSI 95 Connection-Mode Transport Service - The Enhanced QoS.* In *High Performance Networking*, IFIP, 1992.

[6] T. Walter, J. Ellsberger, F. Kristoffersen, P.v.d. Merkhof. *Methods for Testing and Specification (MTS) Semantical relationship between SDL and TTCN A Common Semantics Representation.* European Telecommunications Standards Institute, ETR 071, 1993.

[7] ISO/ITU-T. *Formal Methods in Conformance Testing.* ITU-T TS SG10 Q8 and ISO SC21 WG1 P54.1, March 1995.

[8] S. Fischer. *Spezifikation von Multimediasystemen mit Real-Time Estelle.* In 6. GI/ITG Fachgespräch Formale Beschreibungstechniken für verteilte Systeme, Erlangen, June, 1996.

[9] J. Grabowski, T. Walter. *Testing Quality-of-Service Aspects in Multimedia Applications.* In Proceedings of the Second Workshop on Protocols for Multimedia Systems (PROMS), Salzburg, Austria, October 1995.

[10] D. Hogrefe, S. Leue. *Specifying Real-Time Requirements for Communication Protocols.* Technical Report IAM 92-015, University of Berne, 1992.

[11] T. Henzinger, Z. Manna, A. Pnueli. *Timed Transition Systems.* In *Real-Time: Theory in Practice*, Lecture Notes in Computer Science 600, 1991.

[12] ISO/IEC. *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts.* ISO/IEC 9646-1, 1994.

[13] ISO/IEC. *Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 2: Abstract Test Suite Specification.* ISO/IEC 9646-2, 1994.

[14] ISO/IEC. *Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN).* ISO/IEC 9646-3, June 1996.

[15] R. Keller. *Formal verification of parallel programs.* In Communications of the ACM, Vol. 19, No. 7, 1976.

[16] G. Leduc, L. Léonard. *A timed LOTOS supporting a dense time domain and including new timed operators.* In Formal Description Techniques V, North-Holland, 1993.

[17] L. Léonard, G. Leduc. *An Enhanced Version of Timed LOTOS and its Application to a Case Study.* In Formal Description Techniques VI, North-Holland, 1994.

[18] S. Leue. *Specifying Real-Time Requirements for SDL Specifications - A Temporal Logic Based Approach.* In Protocol Specification, Testing and Verification XV, 1995.

[19] R. Linn. *Conformance Evaluation Methodology and Protocol Testing.* In IEEE Journal on Selected Areas in Communications, Vol. 7, No. 7, 1989.

[20] P. Merlin, D. Faber. *Recoverability of communication protocols.* In IEEE Transactions on Communication, Vol. 24, No. 9, September 1976.

[21] C. Miguel, A. Ferández, L. Vidaller. *Extending LOTOS Towards Performance Evaluation.* In Formal Description Techniques V, North-Holland, 1993.

[22] G. Plotkin. *A structural approach to operational semantics.* Aarhus University, Computer Science Department, 1981.

[23] R. Probert, O. Monkewich. *TTCN: the international notation for specifying tests of communications systems.* In Computer Networks and ISDN Systems, Vol. 23, 1992.

[24] J. Quemada, A. Fernandez. *Introduction of Quantitative Relative Time into LOTOS.* In Protocol Specification, Testing and Verification VII, North-Holland, 1987.

[25] B. Sarikaya. *Conformance Testing: Architectures and Test Sequences.* In Computer Networks and ISDN Systems, Vol. 17, 1989.

[26] J. Montiel, E. Rudolph, J. Burmeister (Editors). *Methods for QoS Verification and Protocol Conformance Testing in IBC - Application Guidelines -.* RACE Ref: 2088, 1993.

[27] T. Walter, J. Ellsberger, F. Kristoffersen, P.v.d. Merkhof. *A Common Semantics Representation for SDL and TTCN.* In Protocol Specification, Testing and Verification XII, North-Holland, 1992.

[28] T. Walter, B. Plattner. *An Operational Semantics for Concurrent TTCN.* In Proceedings Protocol Test Systems V, North-Holland, 1992.

[29] T. Walter, J. Grabowski. *Towards the new Test Specification and Implementation Language 'TelCom TSL'.* In 5. GI/ITG Fachgespräch Formale Beschreibungstechniken für verteilte Systeme, Kaiserslautern, June, 1995.