Quality-of-Service Testing
Specifying Functional QoS Testing Requirements by using Message Sequence Charts and TTCN

Jens Grabowski[a] and Thomas Walter[b]

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

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

Abstract

With the upcoming use of multimedia systems, quality-of-service (QoS) architectures and QoS testing have become a research topic. QoS testing refers to assessing the behavior of a system performing monitoring of negotiated QoS parameter values. QoS testing extends OSI conformance testing since non-functional properties have to be tested, too. We identify requirements for a QoS test specification language and evaluate existing languages against these requirements. We propose the combined use of TTCN (Tree and Tabular Combined Notation) and MSC (Message Sequence Chart) for specifying functional QoS testing requirements. In our approach we use MSC for a high-level overview of the test system components and their respective interactions that are defined in a TTCN test suite. For specifying non-functional requirements, e.g., time constraints, we elaborate on a real-time version of TTCN.

1 Introduction

Quality of Service (QoS) defines a contract between applications and a service provider or communication system to provide a communication service that meets certain quality requirements which respect to throughput, delay, delay jitter etc. Negotiation of QoS parameter values is done between applications and service provider. A QoS semantics defines the procedures how QoS parameter values are negotiated and how the actual QoS parameter values are handled or are enforced during the lifetime of a connection.[1] For performing the latter, QoS monitoring functions have to be implemented in the service provider.

QoS testing refers to assessing the behavior of a service provider performing QoS monitoring functions. It is not necessary to control and observe the QoS monitoring component directly. It suffices doing this implicitly. Assuming that applications have agreed on a set of QoS parameter values and that the service provider supports a specific QoS semantics, i.e., that a connection is aborted if the negotiated QoS cannot be guaranteed further. A lowering of a negotiated QoS parameter, thus, should be detected by the service provider and, subsequently, the connection should be aborted. Therefore, a test case specification for QoS testing has to cover *functional* requirements, i.e., negotiation of QoS parameter values and data exchange over an established connection, and *non-functional* requirements,i.e., the amount of QoS value variation necessary to observe the defined behavior of the service provider.

Functional requirements define the interactions between a system under test (SUT) and a test system. An SUT is meant to be an implementation under test (IUT), e.g., an implementation of a service provider, together with an underlying communication facility, e.g., an ISDN or ATM network. A test system comprises all soft- and hardware components necessary to run the tests on the SUT. Interactions are

[1]A brief summary on these QoS issues can be found in [16].

defined as sequences of stimuli to be sent to an SUT and the foreseen responses of an SUT. Within the ISO/IEC standard 9646 *Conformance Testing Methodology and Framework* [7] stimuli to and responses from the SUT are either abstract service primitives (ASP) or protocol data units (PDU).

Non-functional requirements are often related to time constraints.[2] In the case of QoS testing time constraints are related to the behavior of an SUT and a test system. These time constraints have to be guaranteed during a test run in order to reach a specific test purpose. Time constraints for the test system are needed for driving the SUT into states where the foreseen behavior according to a specific QoS semantics can be checked. For instance, a test system may show a behavior which violates some of the actual QoS parameter values so that re-negotiation of these QoS parameter values has to be performed. Time constraints may be related to a single PDU or ASP, e.g., an SUT has to reply to a connection request within some time limit, or to a computation phase which may include the exchange of several PDUs or ASPs, e.g., if throughput is a negotiated QoS parameter then within some time limits an SUT must be able to receive and to process a number of PDUs or ASPs.

The specification of a test case for QoS testing starts with definition of the functional requirements. The functional test specification will be enhanced by time constraints imposed by non-functional requirements. The language which is used for the functional QoS test specification should also facilitate expressing non-functional requirements. This means we need a language which

- allows the description of static test information, i.e., the declaration of types, constants, variables, PDU and ASP types, and constraints on PDU and ASP parameter values,
- allows the definition of dynamic test information, i.e., the exchange of PDUs and ASPs between SUT and test system in a structured, intuitive and unambiguous manner,
- supports the treatment of test specific information, for instance, test verdicts which indicate whether an SUT conforms to the relevant protocol requirements, and
- includes facilities to express non-functional requirements, like time constraints.

The test specification language recommended by ISO and ITU-T for the description of conformance test suites for OSI protocols is the *Tree and Tabular Combined Notation* (TTCN) [8]. TTCN has two syntax forms: a graphical and textual machine processable form. In this paper we concentrate on the graphical form.

The graphical form of TTCN gives static, dynamic and specific test information in form of tables containing textual information. Practice has shown that real test cases may consist of several hundreds of TTCN tables. Practice has also shown that it is possible to express tests for functional requirements by means of TTCN, but that the notation is not very intuitive, even if tools are used. Furthermore, TTCN is meant to be a notation and not a language. This means that TTCN has an informal semantics definition only which includes severals flaws and ambiguities [1]. Particularly, the description of time constraints is a problem in TTCN. For QoS testing TTCN is, therefore, not suitable.

A second language candidate for covering the functional requirements of QoS testing is *Message Sequence Chart* (MSC). MSC is a formal language recommended by the ITU-T in Z.120 [18]. MSC also distinguishes a graphical and a textual form. In contrast to TTCN, MSC has a formal semantics definition based on basic process algebra [12, 19]. MSC can be characterized as a trace language which concentrates on message interchange of communicating entities (such as services, processes, or protocol entities) and their environment. An advantage of an MSC is its clear graphical layout which immediately gives an intuitive understanding of the described system behavior. MSC has already been used successfully in conformance testing for test purpose definition [2], but also for the definition of the PDU and ASP exchange between SUT and test system [4].

MSC is not meant to be a test specification language. Therefore, static and specific test information cannot be expressed adequately. Furthermore, time is not covered in the formal MSC semantics. We

[2]Besides time constraints non-functional requirements may also be related to error rates, connection establishment failure probability, or security requirements.

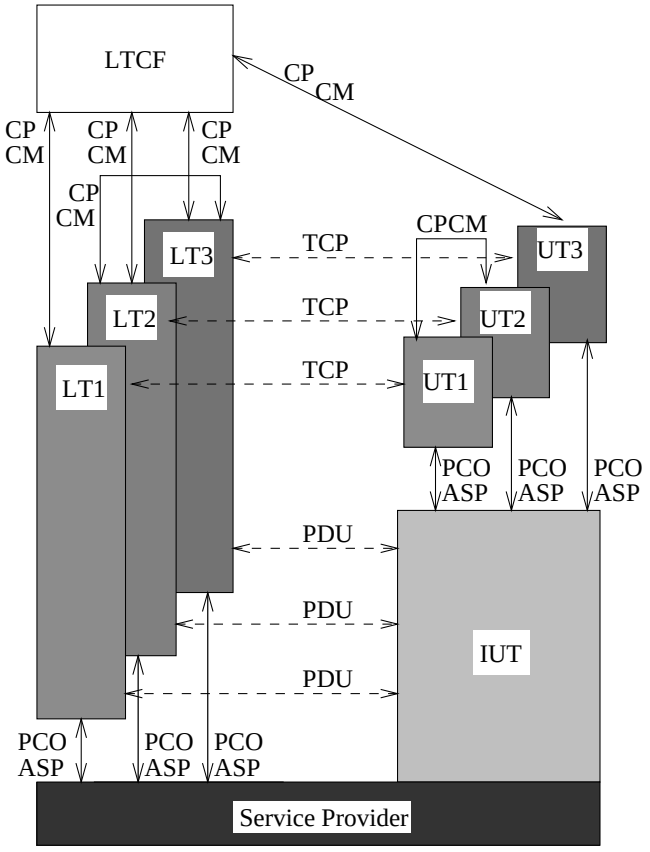


Figure 1: Multi party Testing Context [9].

conclude that MSCs are useful for an abstract test specification in an intuitive and, due to its semantics, unambiguous manner, but MSCs cannot be used for the specification of non-functional requirements as found (and discussed above) in QoS testing.

In this paper we propose a combination of TTCN and MSC for specifying the functional aspects of QoS testing. Sections 2 and 3 give a brief introduction to concurrent TTCN and MSC, respectively. Their combined usage for test specification is discussed in Section 4. QoS testing issues and real-time requirements are covered in Section 5. We elaborate on a real-time extension of concurrent TTCN and, finally, conclude with a summary.

2 Concurrent TTCN

For the purpose of this paper we restrict our attention to *concurrent TTCN* (for short TTCN) [10] concepts related to the description of the dynamic test case behavior. Further details on TTCN can be found in [8, 10, 11, 13, 14].

2.1 Concurrent TTCN Testing Architecture

TTCN supports the definition of complex test configurations, called *multi party testing contexts*, like the one shown in Figure 1. A multi party testing context consists of several concurrently running *test components* (TC). We distinguish between TCs providing the functionality of a *lower tester* (LT), an *upper tester* (UT), and the *lower tester control function* (LTCF). The LTCF is responsible for the creation of all lower and upper tester components. The SUT which is driven by the TCs consists of the IUT and an underlying service provider.

TCs are connected by *coordination points* (CP) through which they exchange *coordination messages* (CM). The rules that govern the exchange of CMs and, furthermore, the coordination between LTs and

Test Case Dynamic Behavior					
Nr	**Label**	**Behavior Description**	**Constraints Ref**	**Verdict**	**Comments**
1		CP ? CM	connected		RECEIVE
2		Start timer			Timer operation
3	L1	? TIMEOUT			TIMEOUT
4		+QoSViolation			ATTACH
5		L ! Datareq	data		SEND
6		-> L1			GOTO

Figure 2: TTCN test case description for Lower Tester B Channel

Test Case Dynamic Behavior					
Nr	**Label**	**Behavior Description**	**Constraints Ref**	**Verdict**	**Comments**
1		[disconnected]			Boolean
2		+ISDN_ConnectionSetUp			ATTACH
3		[connected]			
4		CP ! CM	connected		SEND
5		+ISDN_Disconnection			

Figure 3: TTCN test case description for Lower Tester D Channel

UTs are defined in *test coordination procedures* (TCP).

Communication between TCs and SUT is performed via *points of control and observation* (PCO) through which they exchange ASPs.[3] PCOs and CPs are meant to be unbounded and bi-directional FIFO channels which allow an asynchronous message exchange in both directions.

2.2 Test Case Dynamic Behavior Descriptions

A TTCN test case describes the dynamic behavior of TCs during test execution. A dynamic behavior description consists of *statements* and *verdicts*. A verdict is a statement concerning the conformance of an IUT with respect to the sequence of test case events that was performed. Every execution of a test case results in a verdict assignment.

Statements can be grouped into *statement sequences* and *sets of alternatives*. In the graphical form of TTCN, sequences of statements are identified by different levels of indentation. Statements on the same level of indentation are alternatives. Examples for TTCN behavior descriptions can be found in Figure 2 and 3. The figures will be referred to by the test case example provided in Section 4. TTCN distinguishes between *test events*, *constructs* and *pseudo events*.

Test events are RECEIVE and SEND events. RECEIVE events denote the reception of ASPs, PDUs, and CMs from PCOs and CPs. SEND events describe the sending of ASPs, PDUs and CMs to PCOs or CPs. Test events may be qualified by Boolean expressions and may be followed by a combination of assignments and timer operations.

Constructs are used to create TCs, to guide the flow of control in test cases and to give test cases a modular structure. For the latter TTCN provides the ATTACH mechanism that allows to attach (sub)behavior descriptions, called *test steps*, to a test case. The flow of control in a test case can be modified by using either the GOTO or REPEAT construct. CREATE is a construct used for the creation of TCs. The CREATE operation associates a TC with a behavior tree. If the behavior tree is parameterized the CREATE operation passes also actual parameters for formal parameters. The created TC runs in parallel with all other TCs.

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

[3]TTCN also allows to specify test cases on a more abstract level by specifying the exchange of PDUs. For implementing such a test specification all PDUs have to be encoded in ASPs.

2.3 Test Case Execution

A *behavior tree* [8, 10] defines the relative ordering of statements that a TC executes. A statement is executed if a statement is successful. A statement is successful under the following conditions: A REPEAT and a GOTO is always successful. A SEND event, an assignment and a timer operation are always successful provided that the optional Boolean qualifier holds. A RECEIVE event is successful if an ASP, a PDU or a CM can be received from the specified PCO or CP and the ASP, PDU or CM received matches the constraint and provided that the optional qualifier holds. A CREATE operation is successful if the TC created is not executing already. A qualifier is successful if it evaluates to true. A TIMEOUT event is successful if the specified timer has expired.

Alternatives of a set of alternatives are evaluated one after another starting with the first alternative until one alternative is successful. If an alternative is successful it is executed and test execution proceeds to the next set of alternatives. If no alternative is successful, the evaluation is repeated starting again with the first alternative. Before a set of alternatives is evaluated, the current state of a test system is frozen or, as stated in [8], a *snapshot* is taken: all PCO and CP queues and expired timer lists are updated and during evaluation of a set of alternatives no updates are performed. In a specific sense evaluation of a set of alternatives is an *atomic action*. Test case execution terminates when a final verdict is assigned or a leaf in the behavior tree is reached.

2.4 Concurrent TTCN and Real-Time Constraints

In concurrent TTCN a *discrete* time model is assumed where time is counted in time units of arbitrary granularity (from picoseconds to minutes). Whenever a timer is defined or is started a timeout value is associated with that timer. The status of a timer can be checked in a set of alternatives using the TIMEOUT event. However the time model is not assumed to be used for defining real-time requirements in test cases. To give an example we consider the semantics of a TIMEOUT.

Whenever a timer expires this has no immediate influence on the execution of a test component. Even if the timer expires while evaluation of a set of alternatives is in progress, expiration of the timer is not visible as the expired timers list is only updated when the next snapshot is taken.

Furthermore, the time model of concurrent TTCN cannot be used for the definition of real-time requirements of the test system itself since we cannot predict the execution of TTCN statements, latency of communication between test components, the transmission time of coordination messages etc. This is mainly because we cannot establish a relation between the underlying time model and concurrent TTCN on a syntactical and semantical level.

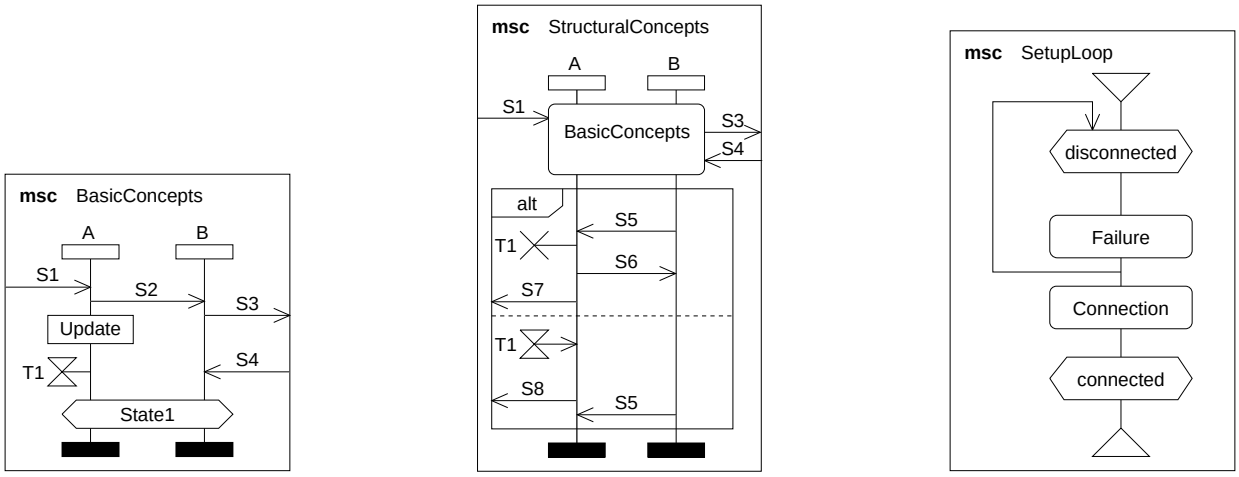
3 Message Sequence Charts

Message Sequence Chart (MSC[4]) is a means for the graphical visualization of selected system runs of communication systems. Since 1990 MSC evolved from an informal notation to a formal description language with complete syntax and semantics definitions [18, 19]. The language definition includes two syntactical forms: a pure textual and a graphical representation. In this paper we focus on the graphical form. The new 1996 version of the MSC language supports the combination of MSCs by means of operators. This allows to describe more complex system behavior and makes MSC usable for test case specification. This section includes a brief overall view of the MSC language and describes strengths and problems of specifying test cases by means of MSC.

3.1 Basic MSC

Figure 4 (a) presents an example of a basic MSC. It describes the message flow between the system environment and the instances A and B. The diagram frame denotes the system environment. Instances are represented by vertical axes and messages are described by horizontal arrows. An arrow origin and

[4]The term *MSC* is used for a diagram written in the MSC language and the language itself. Where necessary, we distinguish between both by using the terms *MSC language* and *MSC diagram*.



(a) (b) (c)

Figure 4: MSC examples

the corresponding arrow head denote sending and consumption of a message. In addition to the message name, parameters may be assigned to a message. All events along an instance axis are totally ordered. The order of events on different instance axes is mediated by the messages, i.e., a message must be sent before it can be received. The inscribed rectangle in Figure 4 (a) describes an *instance action*, i.e., a local activity, of instance A. The inscribed hexagon which covers the instances A and B is a *condition*. It denotes the state *State1* which the covered instances have in common. The hour glass denotes the *setting of a timer* with the name *T1*. Examples for the corresponding *reset* and *timeout* constructs can be found in Figure 4 (b). A cross denotes the reset of a timer and a hour glass with corresponding arrow describes a timeout. Further basic constructs of the MSC language are *instance creation*, *instance termination*, and the order of events along an instance axis (*coregion*).

3.2 Structural Concepts and HMSC

In this section we describe MSC language elements for structuring and combining MSC descriptions. These are *MSC references*, *inline expressions*, and *High-level MSC* (HMSC).

An *MSC reference* refers to another MSC which defines the meaning of the reference, i.e., the reference construct can be seen as a placeholder for an MSC diagram. For example, the MSC in Figure 4 (b) includes a reference to the MSC *BasicConcepts* in Figure 4 (a). In our example the connection between the messages of MSC *BasicConcepts* and MSC *StructuralConcepts* is done implicitly, i.e., via message names. In case of ambiguities the connection can be defined explicitly via *gates*.

MSC *inline expressions* can be used to relate MSC sections by means of operators in order to specify more complex system behavior. Figure 4 (b) includes an example. The inline expression is described by means of a rectangle. The operator name, in our example *alt*, can be found in the upper left corner. Different MSC sections are separated by dashed lines. In the example there are two MSC sections which are related by means of the *alternative* operator. This means they describe alternative system runs, i.e., when the MSC is executed only one of the alternative sections will be executed. In addition to the alternative operator the MSC language definition supports the operators *loop*, *option*, *exception*, and *parallel*.

High-level MSC (HMSC) denotes a special class of MSC diagrams. An HMSC defines graphically how a set of MSC is combined. An HMSC is a graph where each node is either a *start symbol* ($\bigtriangledown$, there is only one start symbol in each HMSC), an *end symbol* ($\bigtriangleup$), an *MSC reference* (Section 3.1), a *condition* (Section 3.1), or a *parallel frame* (i.e., a frame including MSCs which are performed in parallel, e.g., Figure 6 (a)). HMSC symbols are connected by means of flow lines. They indicate the possible sequencing among the nodes in the HMSC. An HMSC example can be found in Figure 4 (c). The first symbol reachable from the start node is the condition *disconnected*. This implies that if the following

MSC, i.e, denoted by the reference to *Failure*, starts with initial condition that it has to be a *disconnected*. After the execution of *Failure* this MSC may be executed again (indicated by the flow line back to the *disconnected* condition), or the MSC *Connection* will be performed. If *Connection* is executed the HMSC will end up in a *connected* state. For consistency, it is required that if *Connection* runs into a final condition, it has to be a condition *connected*.

3.3 MSC and test case specification

There exist several attempts to use MSC for test case specification [2] and test case implementation [4]. These attempts make use of the clear and intuitive understanding of MSC diagrams, but, have to deal with the problem that MSC is not meant to be a test specification and implementation language, i.e., concepts necessary for testing purposes are not implemented in MSC. For example, there exists (a) no notion of test verdicts, (b) no possibility to specify types and constraints, i.e., instances of PDUs and ASPs, (c) no possibility to describe variables and constants for test cases or test suites, and, (d) no notion of test architectures, PICS, and PIXT.

In [2] and [4] additional assumptions and language extensions have been made in order to adapt MSC to the needs of testing. These approaches were successful, but, they are not general, i.e., they support the needs of special test specification and test implementation procedures.

On the one hand a main strength of an MSC is to provide a graphical representation of the overall configuration of a distributed system together with an example for the message exchange between the different system components. On the other hand MSC often allow no complete description of the system behavior. Due to the numerous system runs possible, in most cases it is impossible to describe a distributed system by means of all possible system runs. Instead, it is more suitable to specify the behavior of each component separately.

In the case of QoS testing, we propose to use MSC for providing an abstract and high level view on test configuration and the required message exchange. The behavior of individual test components should be specified by means of concurrent TTCN.

4 Combing concurrent TTCN and MSC

The dynamic behavior of a test case in concurrent TTCN is given in a textual form distributed over several tables where every table contains the dynamic behavior description of a single test component. The combined behavior of test components is not described explicitly since concurrent TTCN does not provide a means for representing the overall configuration of test components and the temporal ordering of test events graphically. As described in the previous section MSCs are quite useful for giving an abstract, high-level view of the system structure and the exchange of messages between instances. Combining concurrent TTCN with MSCs seems to be a feasible approach to enhance the readability of test case specifications. In order to substantiate this claim we discuss an example.

In ISDN (Integrated Digital Services Network) [5, 15] the basic subscriber interface provides two 64 kbps (kilo bits per second) channels called B channels and one 16 kbps signaling channel called D channel. The B channels are used by applications for data exchange whereas the D channel is used for the management of connections between users or application processes. Assuming that communication of application processes is bound by a maximal end-to-end delay, this QoS requirement has to be enforced within the communication system (Figure 5). Whenever the end-to-end delay becomes greater than the negotiated end-to-end delay, the communication system should indicate the violation of the QoS requirement to the applications and abort the connection (using the signaling protocol on the D channel).

A test case for testing compliance of the communication system on the B-subscriber[5] site with the above stated QoS requirement would involve three lower testers which control B and D channels on the A-subscriber site and one upper tester which replaces the application process on the B-subscriber site. Lower and upper tester are running in parallel, i.e., in concurrent TTCN they are described as test components. The test components have to perform the following functions: connection establishment on

[5]In ISDN the A-subscriber is meant to be the calling party. Thus, the B-subscriber is the called party.

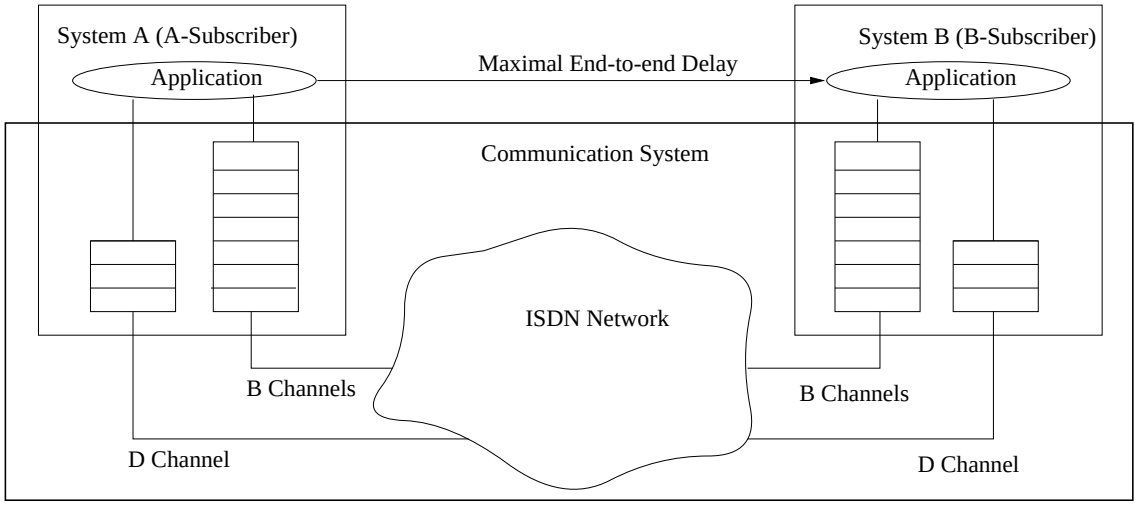
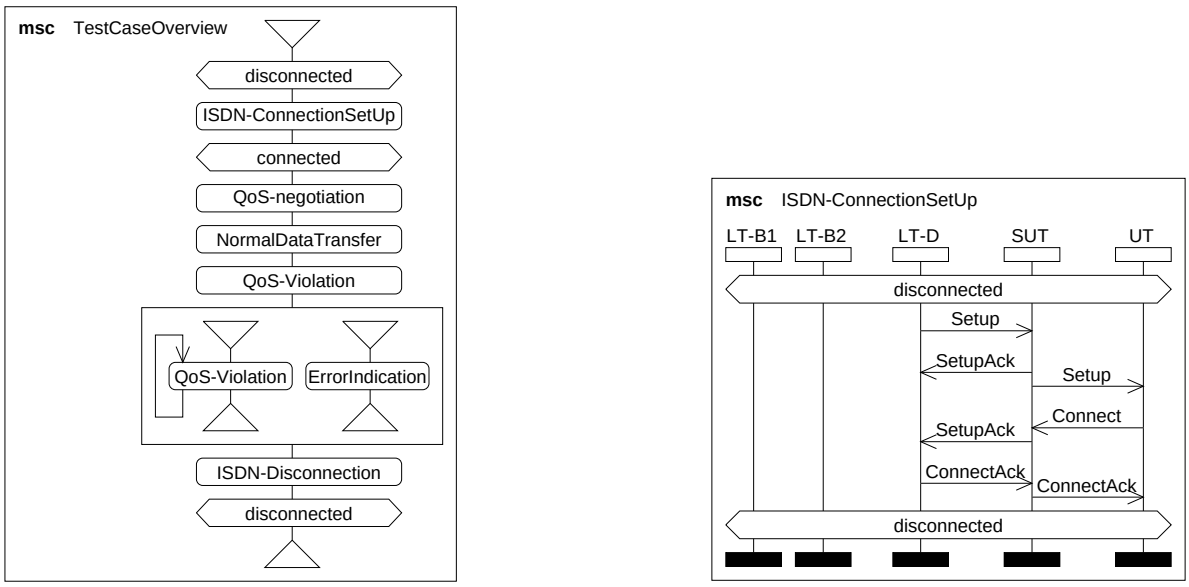


Figure 5: ISDN user interface



(a) (b)

Figure 6: MSC description of the test case example

the D channel, QoS negotiation on the D channel, data transmission on the B channels, and connection abort on the D channel.

The HMSC in Figure 6 (a) provides an overall view of the test case behavior. The functionality is indicated by means of MSC references. Stable testing states[6] are described by means of condition symbols.

The test case starts in the stable testing state *disconnected*. An ISDN connection set up is performed. This is done by using the ISDN signaling protocol on the D channel. The MSC in Figure 6 (b) shows a refinement of this procedure. After connection establishment, i.e., the system is in a *connected* state, the QoS parameters have to be negotiated.

In order to put the SUT into normal operation during a certain period of time, a normal data transfer has to be performed, i.e., all QoS parameters have to be kept. The behavior of lower tester controlling and observing the B channels comprises the sending and reception of data packets where sending a data

[6]A stable testing state denotes a global system states where the system is stable until a tester process provides the next input.

packet means that a time stamp is generated and transmitted with the data packet to the SUT so that the SUT is able to calculate the end-to-end delay from the time stamp received and the local time of the receiver's clock[7].

In the course of exchange of data packets the test components controlling the B channels on the A-subscriber site attempts to increase the end-to-end delay. This can be done by using an additional buffer for all messages to be sent or by producing 'incorrect' time stamps.

Eventually, the SUT determines that the maximal end-to-end delay is not further guaranteed so that the connection have to be aborted. Thus, the SUT indicates the disconnection to the upper tester and sends an abort indication message to the lower tester which controls the D channel. The test case ends in a *disconnected* state.

This example shows that the MSC language is suitable to increase the readability of test case descriptions. HMSCs provide a graphical means for an abstract overall view of the test case behavior and MSCs can be used to describe more detailed communication aspects, i.e., the concrete message exchange between SUT and TCs.

The dynamic behavior of two test components for the provided test case example is partly provided by the Figures 2 and 3. The overall view and the communication between all system components is hidden in several tables. Each table describes just part of the behavior of one test component. Furthermore, besides lower and upper tester a fourth test component, the LTCF, has to be defined. It performs the tasks of the lower tester control function.

5 QoS Testing and Real-Time

The description of the test case example in the previous section is not complete. In order to test the behavior of a communication system supporting maintenance of performance related QoS parameters, a test case description must also give information on the real-time ordering of test events. As we have already argued in Section 2.4, concurrent TTCN cannot express real-time requirements. We propose an extension to concurrent TTCN so that real-time is incorporated in concurrent TTCN.

On a syntactical basis we extend concurrent TTCN so that with every TTCN statement we associate a start time and an end time. Start time and end time are relative to the enabling of a TTCN statement. A TTCN statement becomes enabled whenever execution of a test case comes to the set of alternatives which includes that TTCN statement. Intuitively, execution of a TTCN statement should not start before start time and and should not end after end time. Evaluation of a set of alternatives is done as follows: Firstly, a snapshot is taken which also includes that the current (global) time is taken. Secondly, the alternatives are evaluated in sequential order. An alternative is successful under the conditions described in Section 2.3 and if the current (global) time is greater than or equal the start time of the alternative. The alternative is executed and test case execution proceeds to the next set of alternatives on the next level of indentation. If no alternative is successful a new snapshot is taken and evaluation of alternatives starts again. A test case error occurs if no alternative is successful until the current (global) time has passed the least end time of all alternatives.

We assume that taking a snapshot, evaluation of alternatives and execution of a TTCN statement coincide and are instantaneous, i.e., they do not consume time. With these assumptions the semantics of real-time concurrent TTCN can be defined in terms of *timed transition systems* (TTS) [6].

Applying real-time concurrent TTCN to the previously introduced example allows us to add the timing information with each TTCN statement that is part of the description of lower tester of the B channel in Figure 3. Execution of the test case would result in a timed execution sequence where the time between consecutive send test events is less than the negotiated maximal end-to-end delay except for the last (couple of) test events which are delayed. Eventually, the connection between system A and B is aborted by system B which detects the violation of the negotiated maximal delay.

[7]For simplicity we assume that all clocks are synchronized.

6 Conclusion

In the paper we discussed requirements for a QoS test specification language. In turned out that none of the currently defined language comply with all requirements identified. Nonetheless, we have found a combination of concurrent TTCN and MSCs useful for test specification. Furthermore, we have outlined an extension of concurrent TTCN for specifying real time requirements. The extensions are defined on a syntactical and semantical basis. We will further elaborate on these extensions.

We have started elaborating on multimedia conformance testing in 1995 in a project which we presented in [16]. This paper puts further our initial ideas on a test specification language for multimedia conformance testing.

References

[1] B. Baumgarten, *Open Issues in Conformance Testing Test Specification.* In T. Mizuno, T. Higashino, and N. Shiratori, editors, Proceedings of the 7th IFIP WG6.1 International Workshop on Protocol Test Systems (IWPTS VII) in Tokyo (Japan), November 1994.

[2] J. Grabowski, D. Hogrefe, R. Nahm, *Test Case Generation with Test Purpose Specification by MSCs.* In: SDL'93 - Using Objects. North-Holland, October 1993.

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

[4] J. Grabowski, D. Hogrefe, I. Nussbaumer, A. Spichiger. *Test Case Specification Based on MSCs and ASN.1.* In: SDL'95 - Proceedings of the 7th SDL Forum, Sept. 25 - 29, 1995, Oslo, Norway, North-Holland, Sept. 1995.

[5] F. Halsall, *Data Communications, Computer Networks and Open Systems*, Addison-Wesley, 1994.

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

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

[8] 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, 1992.

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

[10] ISO/IEC, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN): Amendment 1: TTCN Extensions*, ISO/IEC 9646-3 DAM 1, 1993.

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

[12] S. Mauw, M. A. Reniers, *An Algebraic Semantics of Basic Message Sequence Charts.* In: The Computer Journal (Special Issue on Process Algebra), 36(5), 1993.

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

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

[15] A. Tanenbaum, *Computer Networks*, Prentice-Hall, 1989.

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

[17] IUT-T, *Specification and Description Language – SDL*, ITU-T Recommendation Z.100, 1992.

[18] ITU-T, *Message Sequence Chart (MSC)*, ITU-T Draft Recommendation Z.120, 1996.

[19] ITU-T, *Message Sequence Chart - Algebraic Semantics*, ITU-T Rec. Z.120 Annex B, Publ. sched.: May 1995.