Towards an Integrated Test Methodology for Advanced Distributed Systems

Jens Grabowski[a] and Thomas Walter[b]

[a]Institute for Telematics (ITM), Medical University of Lübeck, Ratzeburger Allee 160, D-23538 Lübeck, Germany, phone (+49 451) 500 3723, fax (+49 451) 500 3722, http://www.itm.mu-luebeck.de

[b]Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology Zürich, Gloriastrasse 35, CH-8092 Zürich, Switzerland, phone (+41 1) 632 7007, fax (+41 1) 632 1035, http://www.tik.ee.ethz.ch

Advanced distributed systems like real-time and multimedia systems, behave correctly if functional and non-functional requirements are implemented in accordance with the respective specifications. Functional requirements describe the correct input/output behaviour of a system only, whereas non-functional requirements cover aspects like real-time, performance or robustness. Unfortunately, there exist no testing methodology applicable to advanced distributed systems to assess the compliance with functional and non-functional requirements. In this paper a classification of types of testing is presented. the state of the art of testing distributed systems is summarized and the steps towards an integrated testing systems methodology are described.

Keywords: distributed systems, interoperability, testing, formal description techniques, real-time distributed systems.

1. Introduction

"Product testing is still seen as the only reliable way to assure that outsourced products meet the required specification and are suitable for inclusion in the live network". This statement, that can be found in a publication of the National Physical Laboratory (NPL) [14], is specifically true for communication and distributed systems where components from different vendors have to interwork in order to carry out successfully a common task. Testing is a feasible approach to increase the probability that different communication products are compliant to their respective specifications and, thus, are able to interwork.

Over the past decade, testing of OSI compliant distributed systems has been defined and standardized by ISO (International Organization for Standardization) and ITU-T (International Telecommunication Union) in a document known as *conformance testing methodology and framework* (CTMF) [9]. CTMF defines a black-box testing approach of functional properties. An implementation under test (IUT) is checked for compliance with communication functions defined in the relevant protocol specification. Test cases are derived from the specification and are executed against the IUT. Robustness, failure handling, performance and other non-functional requirements are tested only to a limited extent; if at all. Although CTMF has been defined in the OSI context, it has also been applied outside OSI [2,19].

The progress made in computer technology and communication systems has stimulated a number of new applications of distributed systems that pose new requirements on a testing framework. For the correct operation of, for instance, real-time distributed systems (e.g., process control systems or computer aided manufacturing systems) or multimedia systems (e.g., video-conferencing systems, teleteaching or Internet telephony), compliance of the systems to both functional and non-functional properties is equally important.

In this paper we are looking into the requirements of new advanced distributed systems with respect to testing. We start out by introducing a classification of types of testing (Section 2). Subsequently, we take a look into the state of the art in testing and identify the shortcomings of CTMF with respect to

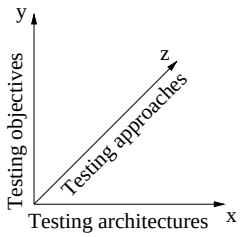

Figure 1. Classification of different types of testing

the requirements of the systems mentioned above (Section 3). In Section 4, we propose extensions to CTMF which will lift CTMF to a testing methodology for advanced distributed systems. We conclude by summarizing our findings and discussing further research work.

2. Types of Testing

Testing is understood as the process of evaluating a system or system components by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results [7]. Essentially, testing is understood as the task of executing a programme in order to find errors. A test is successful if test execution discovers an error [13,20]. Testing as understood in CTMF, has a different focus: "Conformance testing is testing the extent to which an implementation under test satisfies the observable behaviour as permitted by the relevant specifications" [9]. Test execution is successful if observed behaviour complies with foreseen or expected behaviour. Thus, tests are designed for checking specified behaviour to validate that specified capabilities are supported by an implementation, and not to find errors. The following classifiaction of types of testing (Figure 1)[1] is applicable to software as well as conformance testing.

2.1. Testing objectives

A distributed system maintains an ongoing interaction with its environment while performing communication tasks. It receives inputs from and sends outputs to the environment. We refer to this input/output behaviour as *functional behaviour*. In contrast, *non-functional behaviour* covers all aspects beyond pure input/output behaviour, e.g., real-time requirements, performance properties and robustness of an implementation with respect to invalid behaviour of the environment. In [3], a more detailed classification of functional and non-functional properties is introduced which covers aspects like different response time properties (i.e., non-functional properties), functional testing of features and capabilities of an implementation, as well as reliability testing (i.e., a non-functional property), regression testing (i.e., testing functional and non-functional properties of a new software or hardware release) and configuration and capacity planning. The latter aspects are beyond our classification.

Conformance testing is testing the functional behaviour of distributed systems [9]. Conformance testing establishes a link between specified and implemented properties. This is done by running test cases which have been derived from the system specification, against the IUT. If observed behaviour of the IUT and foreseen behaviour as defined by a test case comply then the IUT is a conforming implementation.

With applications of distributed systems in process-control, business and administration (e.g., electronic commerce, workflow management and video-conferencing), home (home banking and entertainment) and teaching (teleteaching and teletutoring), testing of non-functional properties becomes as important as testing of functional properties. For example, in a process control system, computations must be done in strict time frames; otherwise, the controlled process may behave incorrectly which may have catastrophic impacts (e.g., in an air flight control systems), or for a videoconferencing application, a guaranteed quality-of-service of the communication network is essential for the good perception of video and audio.

Unfortunately, testing of non-functional properties of distributed systems is less developed than testing of functional properties. Only very recently, researchers have developed approaches for performance and

[1] Organisational aspects of the testing process [3] are not addressed in this paper. An extended classification scheme may cover these issues by a fourth parameter.

real-time testing [21,23,24].

2.2. Testing approaches

The second parameter of our classification concerns the testing approach, i.e., the necessary assumptions on the control and observability of an IUT. In principle, we can distinguish approaches that only require a minimal control over the implementation at identified interfaces, and approaches that require a maximum of control, e.g., that state variables can be accessed (grey-box testing) or communication between components of a distributed system can be monitored.

In conformance testing, the IUT is accessible only indirectly, having a communications network between test system and system under test. The internals of the implementation are hidden and only a limited access to the IUT is required (see Section 3 for more details).

2.3. Testing architectures

Traditionally, conformance testing is understood as black-box testing of functional properties of a protocol implementation which might be embedded in a system. The focus is on a centralized system where the IUT can be controlled and observed through precisely identified interfaces below and above the IUT. If the IUT is embedded in a larger system, then control and observation of the IUT is only possible indirectly through several protocol layers.

As already pointed out, advanced distributed systems impose new requirements on testing approaches which manifests itself in the types of systems that are to be tested. A multi-point videoconference system, for example, involves several active sites that have to be tested simultaneously. Thus, the IUT as well as the test system must be distributed over real systems. Similarly, it should be possible to assess that a distributed maintains certain guaranteed quality-of-service contracts for the lifetime of a connection. This involves that several test components are distributed over real systems which control and observe several distributed systems which are in charge of the provision of service guarantees.

3. State of the art and open issues

In this section we summarize the existing methodologies and identify open issues.

3.1. OSI Conformance Testing Methodology and Framework

The most complete testing methodology for OSI protocol implementations is the ISO standard 9646 *OSI Conformance Testing Methodology and Framework* (CTMF) [9]. CTMF consists of seven different parts which standardize the entire test procedure of OSI protocol entities to assess their conformance with the corresponding protocol standards. CTMF regulates the specification, execution and evaluation of conformance tests. For making conformance tests comparable and test results reproducible, CTMF also standardizes the tools for test specification.

The tools are abstract test architectures and the *Tree and Tabular Combined Notation* (TTCN) [10] which is a notation for the description of abstract test cases. *Abstract* means that both test architectures and test cases are described independently of concrete implementations.[2]

3.1.1. CTMF test architectures

The principles of CTMF test architectures are based on the OSI Basic Reference Model [8]. The IUT is meant to be a protocol entity which may be embedded in between adjacent layers. In the following, we often use the term *system under test* (SUT) to denote the IUT together with all components necessary to interface the IUT by a test component (TC).

The IUT is controlled and observed from a higher layer TC, called *upper tester* (UT), and from a lower layer TC, called *lower tester* (LT). Control and observation should take place at *points of control and observation* (PCOs). PCOs are infinite FIFO queues at which an asynchronous message exchange can take place. PCOs should be standardized interfaces and therefore in most cases refer to OSI service access points (SAPs).

The existence of accessible PCOs cannot always be guaranteed. Therefore, control and observation at the lower layer interface has to be performed via the service of the lower layer from a remote side, i.e., the

[2]For the usage in a real test environment, tools exist, e.g., TTCN compilers, which bridge the gap between abstract descriptions and concrete implementations.

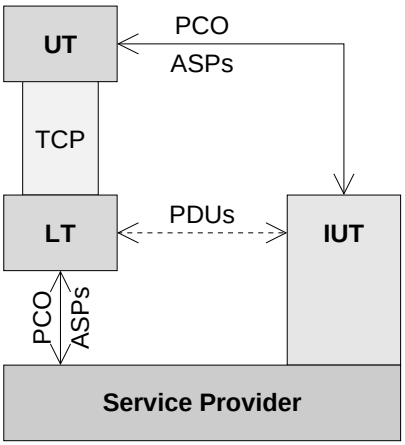

Figure 2. CTMF local test method

LT plays the role of a peer entity of the IUT. If an adequate PCO for the UT cannot be guaranteed, CTMF allows to describe the function of the UT indirectly, i.e., to abstract from the concrete UT behaviour.

CTMF standardizes four basic test architectures called *local*, *distributed*, *coordinated* and *remote test method*. They differ in the possibilities to control and observe the IUT and the distribution of TCs and SUT. As an example, the local test method is sketched in Figure 2.

In the local test method, UT and LT reside on the same hardware and synchronize by using a test coordination procedure (TCP). UT, LT and IUT exchange *abstract service primitives* (ASPs) at the two PCOs. Logically, LT and IUT exchange *protocol data units* (PDUs) which are packed in the ASPs of the underlying service provider.

CTMF generalizes the test methods by introducing the *multi-party testing context*. In the multi-party context several TCs (UTs and LTs) are allowed, which may synchronize by exchanging coordination messages (CMs) at coordination points (CPs). A CP has the same semantics as a PCO, i.e., its model is an infinite FIFO queue used for the asynchronous exchange of messages. In the multi-party context, a *main test component* (MTS) exists, which creates all LTs and which is responsible for determining the test verdict when test execution terminates.

3.1.2. Tree and Tabular Combined Notation

The tool for specifying the behaviour of test cases for CTMF test architectures is the *Tree and Tabular Combined Notation* (TTCN) [10]. In TTCN, a test case specifies which outputs from an IUT can be observed and which inputs can be sent to an IUT. Inputs and outputs are either ASPs or PDUs.[3] Depending on the chosen test architecture, several concurrently running distributed TCs (representing LTs or UTs) may participate in the execution of a test case.

In TTCN, the behaviour description of a test component is given in form of a table (Figure 3 and 4) describing the behaviour of a TC in form of a behaviour tree. The behaviour description consists of *statements* and *verdict assignments*. A verdict assignment is a statement of either PASS, FAIL or INCONCLUSIVE, which describes the conformance of an IUT with respect to the sequence of events which has been performed. TTCN statements are *test events* (SEND, IMPLICIT SEND, RECEIVE, OTHERWISE, TIMEOUT and DONE), *constructs* (CREATE, ATTACH, ACTIVATE, RETURN, GOTO and REPEAT) and *pseudo events* (qualifiers, timer operations and assignments).

The tree structure is given by grouping the statements into *statement sequences* and *sets of alternatives*. A sequence of statements is represented by putting the statements on separate lines with increasing indentation. Lines 1 - 6 in Figure 3 is a statement sequence. Statements on the same level of indentation and with the same predecessor are alternatives. In Figure 4, the statements on lines 4 and 6 are a set of alternatives: they are on the same level of indentation and have the statement on line 3 as their common

[3] In a concrete test case implementation only ASPs are exchanged, i.e., all PDUs are encoded in ASPs. However, TTCN also allows to specify the PDU exchange directly and, therefore, to abstract from a concrete test case implementation.

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			TIMEOUT
6		L ! N-DATA request	data		SEND

Figure 3. 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 4. TTCN Behaviour Description - Set of Alternatives.

predecessor.

3.2. Interoperability testing

Closely related to conformance testing is interoperability testing [1,5]. The goal of interoperability testing is to assess that an implementation is able to interwork with other implementations in order to provide a specific service to all service users. Interoperability testing is often mixed up with CTMF. This is due to the fact that interoperability testing refers to layered architectures and denotes the same type of testing as CTMF, i.e., functional black-box testing. Furthermore, TTCN and the CTMF test architectures or variation thereof are often used for interoperability testing, too. Nevertheless there are some differences. Interoperability testing extends CTMF in two directions: (1) There are PCOs above the IUT only.[4] (2) Some of the proposed interoperability testing approaches require additional monitor points for monitoring communication [4]. Currently, there exists no notion of such monitor points in CTMF, i.e., in these cases CTMF can not be used. Summaries of some interoperability testing approaches and projects can be found in [17] and [24].

3.3. Functional testing beyond OSI

In the last couple of years, several new software architectures for distributed systems have been developed. Examples are the *Common Object Request Broker Architecture* (CORBA) [15], the *OSF Distributed Computing Environment* (DCE) [18], the ISO/ITU-T *Reference Model for Open Distributed Processing* (RM-ODP) [11], or the *Telecommunication Information Networking Architecture* (TINA) [22].

Testing is also used to assess the compliance of applications based on these new architectures with the relevant CORBA, DCE, RM-ODP or TINA specifications. Complete test methodologies comparable to CTMF do not exist for the new architectures. For some approaches, at least the reference points for controlling, observing and monitoring an IUT have been defined, e.g., see [16] for CORBA.

CTMF itself is only partly suitable to cope with the new types of applications. A new integrated test methodology has to extend CTMF in several directions:[5]

[4] For OSI conformance testing, an LT on the remote site of the IUT has always to be present, whereas it is possible to abstract from the UT. In interoperability testing, we have only UTs.

[5] Due to space limitations we cannot discuss all details and therefore just summarize the new requirements for an integrated

- Distribution of the IUT should be allowed. In CTMF, an IUT is one black-box running on a centralized system.
- Dynamic test configurations have to be supported. During test execution the creation and stopping of test components and communication links should be allowed.
- The scope of the new test methodology should be beyond black-box testing towards grey-box testing. For some new applications, monitor points and tester processes which monitor the communication between IUT components are needed. CTMF only supports active communication between TCs and SUT.
- New communication mechanisms have to be introduced. CTMF uses asynchronous point-to-point communication via FIFO buffers. The new applications require further communication mechanisms like broadcast, multicast or synchronous communication.
- New test components have to be introduced. For some applications, components are required which set-up the test environment but do not directly contribute to the test run. For functional tests, examples of such components are emulators or simulators. But the introduction of such new test components is also needed for testing non-functional requirements, e.g., load generators are needed to put the SUT into specific load situations.

3.4. Beyond functional testing

Recently, standardization bodies have started to define the parameters used in performance evaluation. For ATM networks, the parameters to be tested have been defined in [12] and include such as throughput, frame latency, throughput fairness, frame loss ratio, maximum frame burst size and call establishment latency. Some of these parameters are tested with and without background load and, due to statistical variations, the test of some parameters require several test runs.

In addition to the extensions of CTMF mentioned in the previous section, an integrated test methodology which includes mechanism for testing of non-functional properties, has to cope with test components for producing background load and evaluation of several test runs. An immediate consequence is that issues like reproducibility and comparability of test runs and results have to be investigated.

4. Integrated test methodology and framework

For testing advanced communication systems, a new integrated test methodology has to be developed. Following the CTMF approach, such a methodology consists of a generic test architecture and a test language. The test architecture describes static aspects of a test case in terms of a test system configuration. The test language adds dynamic aspects in terms of behaviour descriptions which are assigned to TCs. In the proposed methodology, behaviour descriptions cover the functional and non-functional behaviour of a system. Some initial steps defining a generic test architecture [24] and studying real-time and performance extensions of TTCN [21,23] have already been performed and will be discussed in this section.

4.1. Generic test architecture model

In this section, we present a generic test architecture [24] that extends CTMF along the testing architectures dimension and, along the testing approaches dimension to a limited extent. For the latter, we add some elements for grey-box testing. The basic idea of our approach is to provide a tool-box of elements, which can be combined generically into a test architecture which is suitable for a specific application or system. A test architecture comprises possibly several instances of different types of components. These types are:

- An *Implementation under test* (IUT) represents the implementation or parts of the distributed system to be tested. In principle, an IUT may be distributed over physically separated real systems.
- An *Interface Component* (IC) is a component which is needed for interfacing IUTs, e.g., an underlying service or a system in which an IUT is embedded.

test methodology.

- A *Test Component* (TC) describes a component which contributes to the test verdict by coordinating other TCs or controlling and observing IUTs. A test configuration identifies all TCs necessary for the execution of a specific test case. A TC exists from the start of a test case or is created dynamically by other TCs. In each test architecture, there should be one special *Main Test Component* (MTC) which starts, ends and coordinates the test run.
- A *Controlled Component* (CC) is a component which does not contribute to the test verdict but provides SUT specific data to TCs or the SUT, e.g., a load generator, an emulator or a simulator.
- A *Communication Point* (CoP) represents a point at which communication takes place and at which communication can be observed, controlled or monitored. CoPs denote communication points between all types of components, including communication between IUT components. For the latter case CoPs may be placed somewhere in the IUT, thus CoPs may be used for controlling and observing state information internal to the IUT or to monitor communication between IUTs.
- A *Communication Link* (CL) is a means for describing possible communication flows between TCs, IUTs, ICs and CCs and the kind of communication which may take place. For CLs we distinguish between active and passive CLs. An active CL can be characterized by its kind (synchronous or asynchronous) and its direction (unidirectional or bidirectional). A passive CL allows to monitor communication, i.e., to listen at a CoP. For the dynamic creation of TCs it is assumed that CLs are also created dynamically and that the CoPs linked to the CLs are all known before test execution, i.e., CoPs cannot to be created dynamically.
- The term *System Under Test* (SUT) denotes a combination of ICs and IUTs.

An example for the use of the proposed model is shown in Figure 5. The different hard- and software components of the architecture are shown as boxes and ellipses. The communication flow, the kind of communication and the creation of TCs is indicated by different types of arrows.

Figure 5 describes an architecture proposed for interoperability testing in [24]. There are two IUTs to be tested. One is called *toBeTested1* and the other is embedded in the SUT *toBeTested2*. The IUTs communicate by using an underlying network which in our case is emulated by the CC *Network Emulator*. The IUT *toBeTested1* needs the IC *Lower Layers* for having the interface *CoP2* with CC *Network Emulator*. The communication at *CoP2* is monitored by TC *Monitor*. This is described by the passive CL between *CoP2* and *Monitor*. If necessary the CC *Network Emulator* can be controlled by the TC *Control*.

The MTC is called *UpperTesterFunction*. It communicates asynchronously via *CoP5* with the peer TC *UpperTester*. As indicated by the dotted arrow the TCs *Monitor* and *Control* are created by the MTC. It is assumed that they are running on the same computer and perform a synchronous communication with the MTC.

As can be seen from the example (Figure 5) and the previous discussion, our generic test provides a means to test *really* distributed systems. *toBeTested1* and *toBeTested2* are running on separate systems. Even the test system is physically distributed. Some elements of the specific test architecture in Figure 5 have direct access to parts of the IUT through the interface component *LowerLayers*. In summary, the proposed generic test architecture complements CTMF along the testing architectures dimension and adds elements to enhance CTMF partly with regard to grey-box testing.

4.2. Behaviour descriptions for testing non-functional properties

An essential part of the proposed integrated test methodology is concerned with extending TTCN into a real-time test specification language.

4.2.1. Real-time dynamic test case behaviour

Real-time testing checks that the functional behaviour of an SUT and its real-time behaviour complies with the system specification. In contrast to performance testing (see Section 4.2.2 below), which measures *soft* real-time requirements to be fulfilled statistically, real-time testing deals with *hard* real-time requirements.

In a proposal for an extension of TTCN [23], language constructs are added that support the annotation of TTCN statements with time labels. The language extension is called *real-time TTCN* (RT-TTCN).

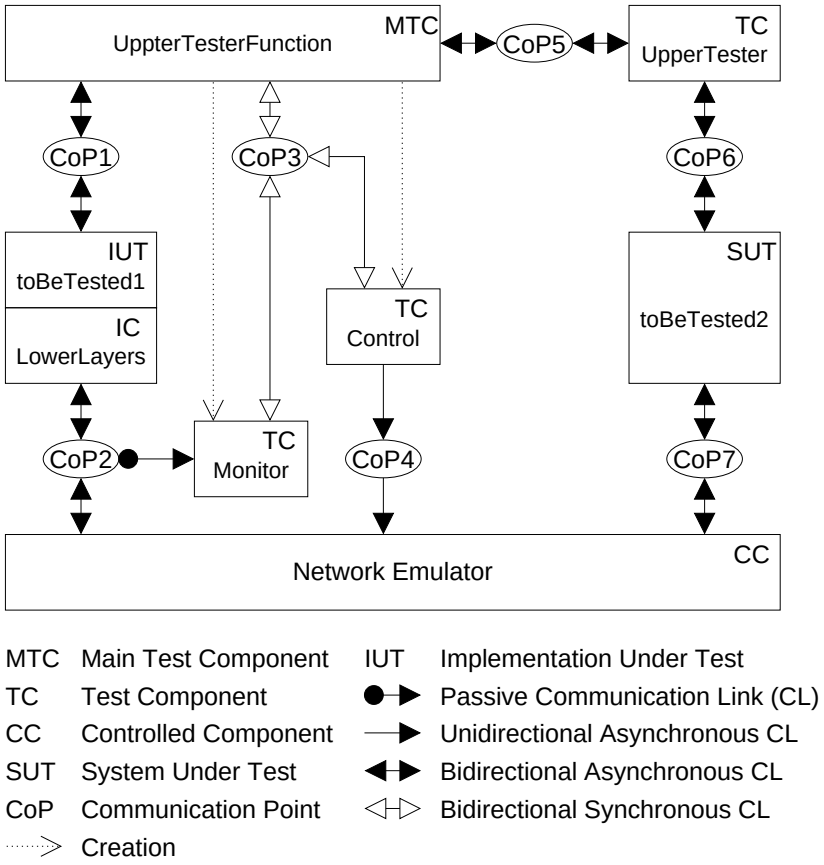

Figure 5. Test architecture for interoperability testing

Test Case Dynamic Behaviour							
Nr	**Label**	**Time**	**Time Options**	**Behaviour Description**	**C**	**V**	**Comments**
1	L1	2, 4	M	A ? DATA und			Time label Mandatory *EET*
2				(NoDur := 3)			Time assignment
3		2, NoDur		A ! DATA ack			
4				(LET := 50)			LET update (ms)
5				A ? Data ind			
6		L1 + WFN, L1 + LET	M, N	B ? Alarm			Mandatory *EET* not pre-emptive

Figure 6. Annotation of TTCN behaviour lines with time labels

An example of an RT-TTCN behaviour description is shown in Figure 6. The annotations of time labels is done in the *Time* column. Relative to the execution of the previous statement, the time labels define a time interval within which the TTCN statements must be executed. A formal semantics of RT-TTCN has been defined using *timed transition systems* [6]. In [23] it has been shown that real-time TTCN can be applied to multimedia systems for testing quality-of-service guarantees.

4.2.2. Performance testing

Performance testing aims at measuring the level of (performance) quality of an implementation under well-known conditions. The level of quality can be expressed in form of a metric using latency, throughput or end-to-end delay as parameters.

Well-known conditions are necessary because performance testing normally has to be carried out in normal and overload situations of the implementation. Test runs should be reproducible and therefore the TCs have to control and monitor all relevant components of the SUT including background load generators.

For the specification of background load, traffic models are used. They describe traffic patterns for continuous streams and data packets with varying inter-arrival times and varying packet length. Markov modulated Poisson processes are an often used model for the description of traffic patterns.

A performance test configuration consists of distributed foreground TCs, background TCs and an MTC for the coordination of test execution. Foreground TCs implement communication with the IUT. Background TCs generate and send continuous data streams to the IUT in order to force the IUT in overload situations. Background and foreground TCs map to CCs and TCs of the proposed generic test architecture.

While executing a test case, a measurement is started which defines the time period during which the performance of the system is being observed. A measurement is performed by monitoring the TCs that are sensitive to the test events to be analyzed. Constraints describe the format of the test events belonging to a measurement, so that a monitor can collect time stamps whenever an event matches the defined constraint.

Real-time and performance testing clearly extend CTMF along the tested properties dimension indicated in Figure 1.

5. Conclusions and Outlook

The discussion presented in this paper is based on our experience of using CTMF and TTCN in several projects organized by ETSI (European Telecommunications Standards Institute) and former EWOS (European Workshop for Open Systems) and from discussions with testing experts. It turned out that, except for the basic principles defined in CTMF, the use of CTMF concepts and TTCN is rather demand-driven. Therefore, our classification of testing and the developed testing methodology are of main importance.

The proposed classification of testing is defined over three parameters. CTMF and TTCN cover only a small fraction of the possible combinations of parameter values. Due to the demands of the upcoming new applications and architectures of distributed systems, a new integrated test methodology is needed.

The generic test architectures extends the CTMF approach by (1) allowing IUT and TCs to be distributed over several real systems; (2) supporting dynamic test configurations where test components and communication links can be created dynamically during a test run; (3) adding controllable test components used as passive components, e.g., for monitoring traffic or as load generators; and (4) allowing for a flexible communication infrastructure which is adaptable to the needs of a specific test case, e.g., establishing a multicast communication using several communication links which are connected to a single communication point.

In order to support all facilities of the generic test architecture, a suitable test specification language has to be defined. Our proposal is to base it on TTCN. We identified the additional concepts needed to test the functional requirements of new advanced distributed systems and discussed the existing proposals for a real-time and performance extension of TTCN.

Parts of the discussed extensions to CTMF and TTCN are considered by ETSI for integration in a future version of TTCN. This effort is done by an expert team which has been established within ETSI and one author is member of this expert team. Furthermore, the test architecture will be implemented in the context of a research project funded by the Swiss National Science Foundation.

Acknowledgements

We thank Stefan Heymer, Beat Koch and Michael Schmitt who read earlier drafts of this paper and provided valuable feedback.

REFERENCES

1. H. Bertine, W. Elsner, P. Verma, K. Tewami. Overview of Protocol Testing Programs, Methodologies, and Standards. AT&T Technical Journal, January 1990.
2. J. Bi, J. Wu. Application of a TTCN based conformance test environment on the Internet email protocol. In M. Kim, S. Kang, K. Hong, editors, *Testing of Communicating Systems*, volume 10, Chapman & Hall, September 1997.
3. R. W. Buchanan. The Art of Testing Network Systems. Wiley Computer Publishing, 1996.
4. W. Buehler (ed.). Introduction to ATM Forum Test Specifications, Version 1.0. ATM Forum Technical Committee, Testing Subworking Group, af-test-0022.000, 1994.
5. J. Gadre, C. Rohrer, C. Summers, S. Symington. A COS Study of OSI Interoperability. Computer Standards & Interfaces, volume 9, 1989.
6. T. Henzinger, Z. Manna, A. Pnueli. Timed Transition Systems. Real-Time: Theory in Practice. Lecture Notes in Computer Science 600, 1991.
7. ANSI/IEEE. Glossary of Software Engineering Terminology. ANSI/IEEE Std 729-1983, ANSI/IEEE Std 729-1983, 1983.
8. ISO. Information Processing Systems - Open Systems - Basic Reference Model. ISO IS 7498, 1984.
9. ISO. Information Technology - OSI - Conformance Testing Methodology and Framework - Part 1: General Concepts. ISO IS 9646-1, 1994.
10. ISO. Information Technology - OSI - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN). ISO IS 9646-3, 1997.
11. ISO. Infomation Technology - OSI - Reference Model for Open Distributed Processing, Part 2: Descriptive Model. ISO IS 10746-2, 1991.
12. R. Jain, G. Babic, A. Durresi. ATM Forum Performance Testing Specification - Baseline Text. ATM Forum Document Number: BTD-TEST-TM-PERF.00.05 (96-0810R8), February 1998.
13. G. J. Myers. The Art of Software Testing. John Wiley, 1979.
14. National Physical Laboratory. Counting on IT. Issue 7, Summer 1998.
15. A. Pope. The CORBA Reference Guide - Understanding the Common Object Request Broker Architecture. Addison Wesley, 1998.
16. S. Rao, C. BeHanna, M. Sun, F. Forys. CORBA Service Test Environment. NEC Systems Laboratory Inc., 1997.
17. D. Rayner. Future directions for protocol testing, learning the lessons from the past. In M. Kim, S. Kang, K. Hong, editors, *Testing of Communicating Systems*, volume 10, Chapman & Hall, September 1997.
18. A. Schill. DCE – The OSF Distributed Computing Environment (in German). Springer Verlag, 1997.
19. I. Schieferdecker, A. Rennoch. Formal Based Testing of ATM Signalling. In U. Herzog, H. Hermanns, editors, *Formal Beschreibungstechniken für Verteilte Systeme*, Band 29, Nummer 9, Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung, Erlangen, Mai 1996.
20. I. Sommerville. Software engineering. Addison Wesley, 1989.
21. I. Schieferdecker, S. Stepien, A. Rennoch. PerfTTCN, a TTCN Language Extension for Performance Testing. In M. Kim, S. Kang, K. Hong, editors, *Testing of Communicating Systems*, volume 10, Chapman & Hall, September 1997.
22. TINA Consortium. http://www.tinac.com
23. T. Walter, J. Grabowski. Real-time TTCN for Testing Real-time and Multimedia Systems. In M. Kim, S. Kang, K. Hong, editors, *Testing of Communicating Systems*, volume 10, Chapman & Hall, September 1997.
24. T. Walter, I. Schieferdecker, J. Grabowski. Test Architectures for Distributed Systems - State of the Art and Beyond (Invited Paper). In A. Petrenko, N. Yevtushenko, editor, *Testing of Communicating Systems*, volume 11, Chapman & Hall, September 1998.