Test Architectures for Distributed Systems - State of the Art and Beyond

T. Walter
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology Zurich
Gloriastrasse 35, CH-8092 Zürich, Switzerland
Phone: (+41 1) 632 7007, Fax: (+41 1) 632 1035
http://www.tik.ee.ethz.ch

I. Schieferdecker
GMD FOKUS
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
Phone: (+49 30) 3463 7241, Fax: (+49 30) 3463 8241
http://www.fokus.gmd.de/tip

J. Grabowski
Institute for Telematics (ITM)
Medical University of Lübeck
Ratzeburger Allee, D-23538 Lübeck, Germany
Phone: (+49 451) 500 3723, Fax: (+49 451) 500 3722
http://www.itm.mu-luebeck.de

Abstract

A generic test architecture for conformance, interoperability and performance testing of distributed systems is presented. The generic test architecture extends current test architectures with respect to the types of systems that can be tested. Whereas in the conformance testing methodology and framework the focus is on testing protocol implementations, the generic architecture focuses on testing real distributed systems whose communication functions are implemented on different real systems and whose correctness can only be assessed when tested as a whole. In support of the latter requirement, a test system itself is regarded as being a distributed system whose behaviour is determined by the behaviour of components and their interaction using a flexible and dynamic communication structure.

Keywords

CTMF, TTCN, test architecture, types of testing, advanced distributed systems

1 INTRODUCTION

A distributed processing system is a system which can exploit a physical architecture consisting of multiple, autonomous processing elements that do not share memory but cooperate by sending messages over a communications network (Blair et al, 1998). Distributed processing is information processing in which discrete components may be located in different places, or where communication between components may suffer delay or may fail.

Distributed systems offer several advantages in comparison to centralized systems such as the ability to share resources, to be dynamically extended with new ones and to potentially increase availability and performance. Typically, distributed systems are heterogeneous in terms of interconnected networks, operating systems and middleware platforms they are based on, as well as in terms of programming languages used to develop individual components.

The goal of open distributed systems is to enable access to components of a distributed system from anywhere in the distributed environment without concern of its heterogeneity. Openness includes openness to various levels of the distributed system architecture: communication network, middleware platform and application level. Openness requires the definition of interfaces to the components on the different levels of distributed systems and a notion of compliance to these interfaces in order to ensure compatibility, interoperability and portability.

The rapid growth of distributed processing systems has led to a need for coordination and for standardized frameworks:

- The ISO/ITU-T Reference Model for Open Distributed Systems (RM-ODP) (ISO International Standard 10746, 1991), (Blair et al, 1998) provides a framework by describing a generic object-oriented architecture that is based on five different viewpoints. The viewpoints enable the description of distributed systems from various perspectives: enterprise, information, computation, engineering, and technology viewpoint. The framework also defines functions required to support ODP systems and transparency prescriptions showing how to use the ODP functions to achieve distribution transparency. The ODP framework is provided in terms of architectural concepts and terminology and is a rather general and generic framework to enable concrete standards for open distributed systems to emerge.

- The OMG Common Object Request Broker Architecture (CORBA) Object Management Group, 1995) provides an object-oriented framework for distributed computing and mechanisms that support the transparent interaction of objects in a distributed environment. The basis of the architecture is an object request broker (ORB) which provides mechanisms by which objects transparently invoke operations and receive results.
- The Telecommunication Information Networking Architecture (TINA) (TINA-C, 1998) aims at providing a framework for future telecommunication networks. The kernel of the architecture is a distributed processing environment (DPE) that adopts the concepts of the RM-ODP computational and engineering model. The basis for the TINA DPE is a CORBA ORB.
- The Open Group's Distributed Computing Environment (DCE) (Digital Equipment Corporation, 1992), (Open Software Foundation, 1992), (Lockhart, 1994), (Schill, 1996) provides a communication network and middleware platform independent platform for distributed processing systems. It is based on a client-server architecture and does not support object-oriented technologies.

The main concept which should ensure compatibility, interoperability and portability of the various components in a distributed processing system is conformance. Conformance testing is the main tool to check if components of a distributed processing system are able to interwork. ODP, CORBA, TINA and DCE have different definitions of conformance:

In ODP conformance is a relationship between specifications and implementations. The relationship holds when specific requirements in the specification are met by the implementation. ODP does not define conformance for specific requirements but rather identifies conformance points at which an implementation should be tested. Conformance testing in ODP means to test all requirements that are given by the viewpoint models of a distributed system.

Conformance in CORBA is defined in terms of compliance points, which refer to the interfaces of the CORBA architecture. Emphasis is given to compliance points from the ORB Interoperability Architecture (with GIOP, IIOP and ESIOP) in order to improve the interoperability between ORBs.

Currently, in the TINA framework there are no explicit definitions for conformance. However, work is in progress to use reference points, e.g., the retailer reference point, as the basis for defining conformance.

The DCE approach towards conformance is based on system specifications and associated conformance test suites. An implementation will only be claimed DCE conforming if it passes the conformance tests.

As a summary it can be said that the problem of conformance testing for ODP, CORBA, TINA and DCE based applications is not resolved. Research, industry and standardization bodies are working on solutions for providing unique and comparable test specifications.

However, the most successful conformance testing standard is the Conformance Testing Methodology and Framework (CTMF) (ISO International Standard 9646-1, 1995), (Linn, 1989), (Sarikaya, 1989), (Baumgarten et al, 1994) defined by ISO/IEC for the test of communication protocols and services that are in accordance with the OSI (Open Systems Interconnection) basic reference model (ISO International Standard 7498, 1984). Beside others, CTMF defines several test architectures, called abstract test methods, and the Tree and Tabular Combined Notation (TTCN) (ISO International Standard 9646, 1996), (Probert et al, 1992) for the specification of test cases. In this paper, the terms test architectures and abstract test methods are used as synonyms.

The CTMF test architectures and TTCN have been used successfully also outside the OSI conformance testing area, e.g., for testing ATM protocols. But, it has been recognized that the CTMF definitions, in general, cannot cope with other types of testing, e.g., real-time and performance testing, and with applications based on new architectures and frameworks (ODP, CORBA, TINA and DCE). For opening the scope of TTCN for real-time and performance testing several language extensions have been proposed (Schieferdecker et al, 1997), (Walter et al, 1997).

This paper intends to open the discussion on test architectures. A definition of a generic test architecture is proposed which can be adapted to different types of testing and to testing of applications based on new architectures and frameworks.

The paper is organized as follows: Section 2 presents the state of the art in test architectures. In Section 3 the requirements on test architectures from advanced distributed systems are identified. A new generic test architecture for distributed systems is proposed in Section 4. The application of the new architecture model is shown in Section 5. The paper concludes with a summary.

2 STATE OF THE ART IN TEST ARCHITECTURES

For a couple of years, the international standard 9646 (Conformance Testing Methodology and Framework, CTMF) (ISO International Standard 9646-1, 1995), (ISO International Standard 9646-2, 1995) has been the reference for test architectures. In its first and second part, it defines a number of abstract test methods. Abstract test methods describe how an implementation under test (IUT) is to be tested, i.e., what outputs from the IUT are observed and what inputs to the IUT can be controlled.

2.1 Abstract test methods

Taking the possible variety of real open systems configuration into account, a number of abstract test methods have been defined. Abstract test methods resemble a number of concepts that have been introduced in the Open Systems Interconnection Basic Reference Model (OSI BRM) (ISO International Standard 7498, 1984). In particular, a system under test (SUT) is generally a system that uses standardized OSI protocols in all layers, from the physical to the application layer. However, CTMF is also applicable to partial open systems, i.e., systems that use OSI protocols only in some layers to provide an OSI service, and CTMF is applicable to relay systems at network and application level. Within the SUT the IUT is subject to conformance testing and may comprise several adjacent protocol layers. The IUT is assumed to rest on an underlying service provider connecting SUT and test system.

Given the situation of an IUT which is a single layer protocol implementation. The interfaces between IUT and its adjacent layers within the SUT are referred to as points of control and observation (PCO). As the name suggests, at these points the behaviour of the IUT in performing communication tasks can be controlled and observed. As stated above control and observation are in terms of ASPs (which map to OSI service primitives) and PDUs embedded in ASPs. Similar to the PCOs located in the SUT, corresponding PCOs can be identified in the test system and the underlying service provider, i.e., PCOs may be allocated somewhere below and above the IUT. Even if the PCO through which the IUT is accessed is on the remote test system, it is said that control and observation of the IUT takes place from below.

Although PCOs can be located somewhere in the SUT below and above the IUT, not all conceivable arrangement of PCOs must be supported by the SUT. A few specific abstract test methods have been defined which clearly identify the requirements on the SUT and IUT concerning access to interfaces and location.

These abstract test methods use one or two PCOs. In case of two PCOs, one is below and one is above the IUT, otherwise only the PCO below the IUT is used for controlling and observing the IUT. Even if the PCO above the IUT is available, this does not imply the control and observation of this interface to the IUT is done from within the SUT. In the local test method (Figure 1) the upper PCO is managed from within the test system. In the distributed test method (Figure 1) this is done from within the SUT.

The active components in a test system which perform control and observation of the IUT are named lower and upper tester (LT & UT). Coordination of these components is defined in a test coordination procedure (TCP), which similar to a protocol specification, defines the rules for the cooperation of LT and UT. In addition, the underlying service provider is regarded as an active component which is assumed to be sufficiently reliable so that control and observation of an IUT can be done remotely.

The coordinated test method (Figure 2) explicitly uses a test management protocol (TMP) as TCP. Although an UT is sitting on top of the IUT, no PCO is used in this test methods. The UT is assumed to implement parts of the functionality of the test management protocol.

The remote test methods puts the least requirements on IUT, SUT and availability of PCOs (Figure 2). Only the PCO below the LT is available. No UT is used in the remote test method. However, for the purpose of test case specification, some UT functions which may be present, can be used if necessary. No assumptions are made

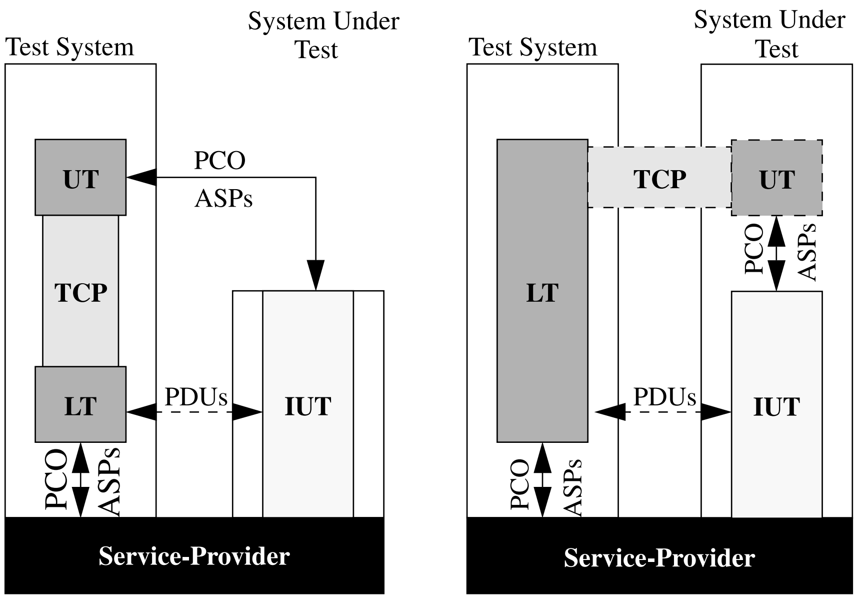


Figure 1 Local and distributed abstract test methods.

on the feasibility or implementability of these function in a real system.

In (Zeng et al, 1989) the Ferry Clip approach is presented. It provides implementation support for abstract test methods. Its purpose is to provide access to the lower and upper boundaries of an IUT in the SUT. The test system is remote from the SUT, but a specific component, called passive ferry clip, is installed on the SUT. It controls and observes the respective IUT interfaces. The passive ferry clip communication with its counterpart in the test system, called active ferry clip, uses a ferry control protocol and ferry transfer service. UT and LT run on the test system and communicate with the active ferry clip. The test case logic is still in the test components, but the ferry clips perform as mediators between test system and SUT and IUT, respectively. This approach makes the implementation of abstract test methods rather simple because the complexity of implementing a passive ferry clip is lower than implementing a complete UT.

2.2 Multi-party testing context

A generalization of the above discussed abstract test methods, also known as single-party testing context, is given by the multi-party testing context (Figure 3). In this setting, an IUT is supposed to communicate simultaneously with several real open systems; a networt of application relays, for instance, maintains communication links with several peers at the same time. In the multi-party testing context more than

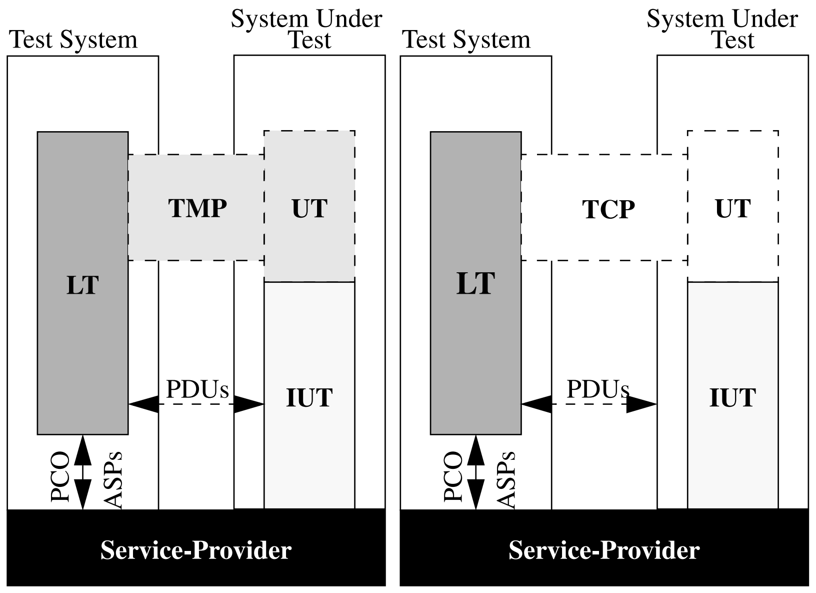


Figure 2 Coordinated and remote test methods.

one LT and zero, one or more UTs are active and control and observe the IUT. The coordination of LTs is performed by an entity referred to as lower tester control function (LTCF). In particular, the LTCF is the entity that determines the final test verdict after test execution. All LTs are required to return a preliminary test result to the LTCF after they have stopped test case execution. The LTCF is also responsible for starting all LTs. Coordination of LTCF, LTs and UTs is defined in a TCP. Communication between LTCF, LTs and UTs is supported by coordination points (CP) which, like PCOs, are modelled as two FIFO queues for inputs and outputs.

LTs communicate with the IUT and possibly with an UT as in the single-party context, the same rules for identifying points of control and observation apply.

2.3 Test Architecture beyond Conformance

In (ISO International Standard 9646-1, 1995), the following statement can be found: "The primary purpose of conformance testing is to increase the probability that different implementations are able to interwork.... Even if two implementations conform to the same protocol specification, they may fail to interwork fully. Trial interworking is therefore recommended." According to (ISO International Standard 9646-1, 1995) and (Gadre et al, 1990), conforming implementations may fail to interwork for the following reasons:

- protocol specifications contain options and implementations may differ in the options being supported

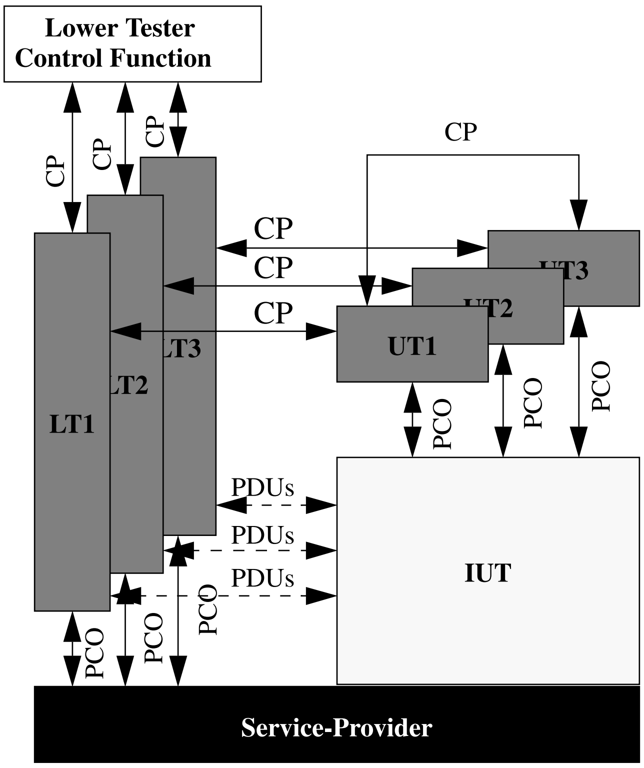


Figure 3 Multi-party testing contex.

- factors outside the scope of conformance testing and OSI, e.g., performance requirements, may impact the behaviour of real systems which is not foreseen by the specification.

In the following subsections, approaches to testing interoperability, performance and quality-of-service are discussed. Only since the evolution of multimedia applications has started some years ago, the latter type of testing has become an issue.

2.3.1 Interoperability Testing

Interoperability testing evaluates the degree of interoperability of implementations with one another. It involves testing both, the capabilities and the behaviour of an implementation in an interconnected environment and checking whether an implementation can communicate with another implementation of the same or of a different type. Interoperability testing is not standardized. However, a couple of interoperability testing proposals and guidelines exist (Buehler, 1994), (Myungchul et al, 1996), (EWOS ETG 028, 1993), (Hopkinson, 1996), (Gadre et al, 1990) (Hogrefe, 1990).

The two main approaches to interoperability testing in the context of OSI are passive and active testing (Gadre et al, 1990) (Hogrefe, 1990). The difference is that active testing allows controlled error generation and a more detailed observation of the communication, whereas passive testing on the other side involves testing valid behaviour only. OSI interoperability testing should be done by using a reference implementation (RI) of the protocol entity to be tested. Within the SUT RI and IUT are combined and the tester functions play the role of service users. The behaviour of the RI is correct by definition. In case that IUT and RI do not interwork this is due to an error in the IUT.

Figure 4 shows a generic passive interoperability test system architecture where two implementations are interconnected and control and observation are performed by two UTs.

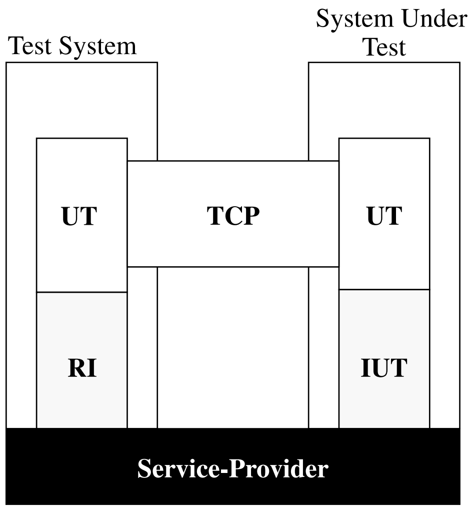


Figure 4 Passive Interoperability Testing Architecture.

Practice has shown that in most cases no RI is available. As a consequence, two IUTs are used and the communication between the IUTs is monitored or emulated. A test configuration following this approach and proposed by the ATM Forum in (Buehler, 1994) is shown in Figure 5.

However, the test configuration most often used for interoperability testing is simply the interconnection of the network components which have to interoperate. The message exchange between the network components is stimulated at the edges by a test components or by any suitable application. This complies with Figure 5 without monitor point C. The observed reactions on the edges were used to assign a verdict about the interaction between the SUTs. This form of testing has been termed pure interoperability testing (Myungchul et al, 1996). Set-up of such an pure interoperability test session is possible with limited hardware and software requirements, and thus it is very popular within the test labs. The main disadvantage of this test configuration is the missing monitor point C between the SUTs. It does not allow a statement about the protocol between the two systems and makes locating errors very difficult in the case of failure.

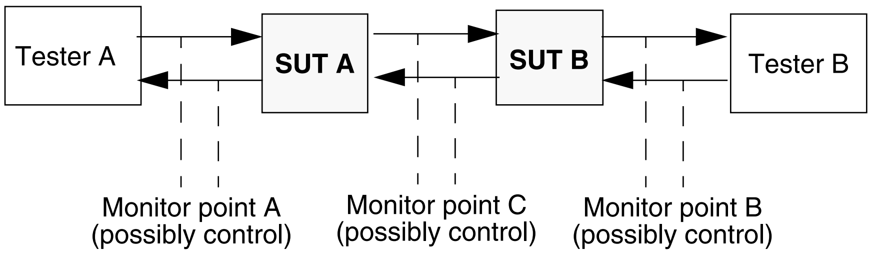

Tester A
SUT A
SUT B
Tester B
Monitor point A (possibly control)
Monitor point C (possibly control)
Monitor point B (possibly control)

Figure 5 ATM Forum interoperability test architecture

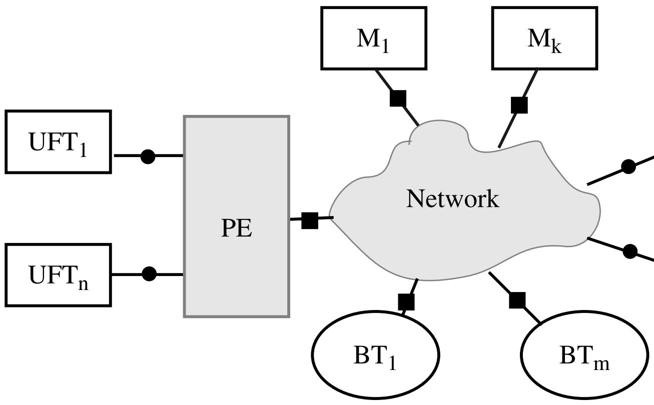
Other approaches to interoperability testing use the test configuration shown in Figure 5, but differ in the definition of conformance requirements. Interoperability testing with strong conformance requirements on all monitor points realizes conformance testing according to CTMF with the ancillary condition of two IUTs. This includes static and dynamic conformance testing. It provides a high level of confidence that the implementations are able to interoperate, but causes problems if implementations to not meet the standard specifications but still interoperate.

In order to have a direct control on the tested systems, a specific test component is used for emulating the transmission service in between. The emulator component makes the monitor point C an active testing point, where possibly impairments are generated in accordance with the properties of the transmission service, i.e., errors such as loss, corruption, disordering, or insertion are generated. Please note, that in the case of a reliable transmission service without errors the emulator test component is not needed. An extended interoperability architecture is shown in Figure 11.

2.3.2 Performance Testing

The main objective of performance testing (Schieferdecker et al, 1997) is to test the performance of a network component under normal and overload situations. Performance testing identifies performance levels of the network component for different ranges of parameter settings and assesses the measured performance of the component. A performance test suite describes precisely the performance characteristics that have to be measured and procedures how to execute the measurements. In addition, the performance test configuration including the configuration of the network component, the configuration of the underlying networt, and the network load characteristics are described.

Depending on the characteristics of the network component under test, different types of performance test configurations are defined: end-user telecommunication application, end-to-end telecommunication service and communication protocol (Figure 6). Foreground test components (FT) implement control and observation of the network under test. Background test components (BT) generate continuous streams of data to load the network component under test. Monitor components are used to monitor the real network load during the performance test. BTs do not control or observe directly the network under test but implicitly influences the network under test by putting the network into normal and overload situations.



☐	**Performance Test Components**	●	Points of Control and Observation
▨	**SUT Components**	■	Measurement Points

UFT: Foreground Tester for Emulated Protocol User
LFT: Foreground Tester for Emulated Peer-to-Peer Protocol Entity
BT: Background Tester
M: Monitors of Real Network Load
PE: Tested Protocol Entity

Figure 6 Performance test configuration for a communication protocol.

2.3.3 Quality-of-Service Testing

Quality-of-service (QoS) (ITU-T Recommendation X.200, 1989), (Danthine et al, 1992), (Danthine et al, 1993) refers to a set of parameters that characterize a connection between communication entities across a network. QoS parameters are performance and reliability oriented such as throughput, delay, jitter or error rates and failure probabilities, respectively. QoS parameters are negotiated by service users and service provider at connection set-up and should be maintained during the life-time of the connection. A QoS semantics defines the procedures how QoS parameters are negotiated and how negotiated QoS parameters are to be handled. In particular, a QoS semantics defines how QoS parameter violations are to be processed. It is mainly the service provider who is in charge of maintaining negotiated QoS parameters.

QoS testing refers to assessing the behaviour of protocol implementations performing QoS maintenance. However, it is not necessary to control and to observe the behaviour of the implementation directly. It suffices if the tester can eventually observe the specified behaviour according to the agreed QoS semantics.

Figure 7 presents a testing architecture for QoS testing (Grabowski et al, 1995). As can be seen, the testing architecture is very similar to the passive interoperability testing architecture (Figure 4). The obvious difference is that in QoS testing the IUT is distributed. Furthermore, some QoS parameter tests, e.g., error rate tests and delay tests, require the active involvement of the network. For this, the network has to be configurable. The testing architecture, therefore, provides a communication link between testers and network for the exchange of configuration information.

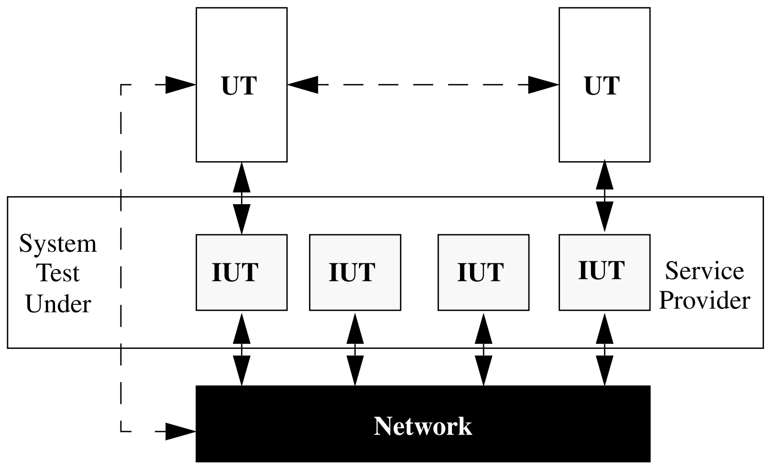


Figure 7 QoS testing architecture.

2.3.4 An Evaluation of Test Architectures

As the previous discussion of the different test architecture has shown, interoperability, performance and QoS test architecture are straightforward extensions of the CTMF abstract test methods. This is particularly true for the interoperability test architecture where the lower tester functions are provided by a reference implementation, which in turn is driven by a test component on top. Similarly, the QoS test architecture transforms the interoperability test architecture into a distributed test architecture, where the functions of the IUT are distributed over several real systems. Control and observation of the distributed IUT are done by UTs. Additionally, UTs can control the behaviour of the underlying network. This, however, makes the QoS test architecture different. Whereas in the CTMF test methods and interoperability test architectures the underlying network is regarded as a black-box, the QoS test architecture weakens this assumption because certain behaviour of the SUT is observable only if network behaviour changes in a controlled manner. As such, the performance test architecture does something similar. In order to force the IUT into specific states, additional network load is generated by specifically designed test components. Thus, the network is being controlled by the test system as well.

.

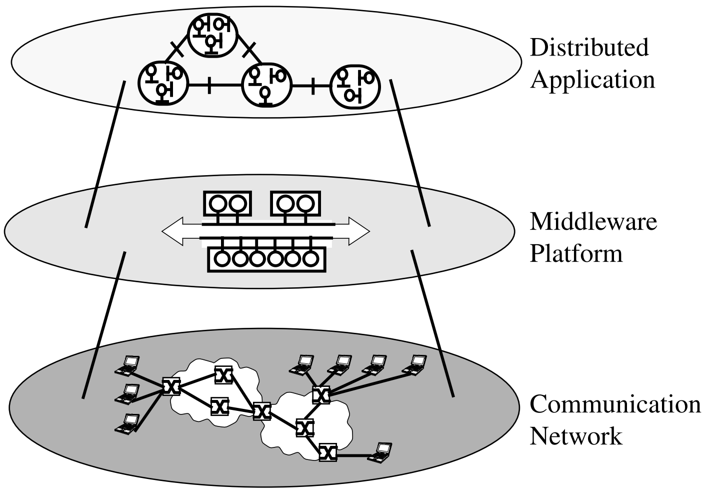


Figure 8 Distributed Systems Architecture.

3 REQUIREMENTS FROM ADVANCED DISTRIBUTED SYSTEMS

A simplified view of distributed system architectures is given in Figure 8. A distributed system consists of application level objects that interact through well-defined interfaces. The middleware platform is a distributed computing environment that supports the implementation of the distributed application by offering various distribution transparencies such as access and location transparency. Example middleware platforms are OMG CORBA or OSF DCE. The communication network with various network nodes and end-systems offers transmission services to the middleware platform that are used to support the communication between the components of the distributed system.

Testing one of the three layers imposes different requirements on testing. Subsequently, testing the communication network, the distributed processing environment, and the distributed applications level is discussed.

3.1 Testing of Advanced Communication Networks

The IETF activities on defining communication protocols, services and mechanisms in internetworks are the driving forces in defining advanced network technologies. The following aspects of the current Internet (in particular of the Integrated and Differentiated Service Architecture (IETF Differentiated Service Working Group, 1998), (IETF Integrated Services Working Group, 1998)) impose new requirements on testing network nodes as well as end-to-end services:

- a variety of group communication scenarios such as multicast with dynamic join and leave in multicast groups,
- stream support with different levels of QoS guarantees and soft-state resource reservations, and
- variety of routing protocols including multilayer routing approaches.

Another direction in advanced communication technology are active networks. Active networks use programmable network nodes or capsules, which combine communication data with code fragments on how to handle the data in the network nodes (Tennenhouse, 1997). Testing of active networks primarily requires efficient means to test the interoperability and compatibility of network nodes in highly dynamic environments.

3.2 Testing of Advanced Middleware Platforms

Due to the need for interoperable implementations of various vendors and due to the complex nature of middleware platforms such as CORBA, there is a need for a methodology that middleware platforms can be analysed with respect to their compliance to the respective specifications. This is done in order to increase the likelihood that they can interoperate.

In essence, a middleware platform is used by a distributed application like a black box with various service access points. However, in order to evaluate for example a CORBA ORB as such (what includes testing the ORB Core, the Interoperability Reference Points, CORBA Services, and CORBA Facilities), a grey-box testing approach has to be taken, which supports the test access to ORB internal interfaces (Rao et al, 1997).

3.3 Testing of Advanced Distributed Applications

With the development of ODP and TINA -- being an instantiation of ODP for telecommunication systems -- and the provision of new and complex services such as telecommunication, management and information services which may be deployed in the context of various distributed object computing environments, the need to validate and test large and heterogeneous object systems is becoming increasingly important. Testing may be used to check

- components of the applications individually,
- conformance to reference points, and
- to check individual service components working together in a multi-service environment.

3.4 Requirements of Testing Advanced Distributed Systems

As a result of the above analysis, the following requirements of testing advanced distributed systems can be identified:

- development of distributed testing architectures with means for synchronizing distributed test components;
- support of dynamically configurable and scalable test architectures;
- ability to express test configurations for different communication scenarios;
- possibility to use grey-box testing with access to internal components and interfaces;
- support of real-time, performance and QoS testing for distributed systems to test time-related aspects of distributed systems;
- support of interoperability testing to focus on essential interoperability aspects;
- development of a test methodology coherent with object-oriented technologies which are used for the development of distributed systems;
- test architectures that make use of management, monitoring and measurement systems;
- methods that support testing in the pre-deployment, deployment and usage phase of distributed systems;
- efficient testing methods in order to deal with the complexity of distributed systems.

4 GENERIC TEST ARCHITECTURE FOR DISTRIBUTED SYSTEMS

The requirements of advanced distributed systems with respect to testing architectures are not met by CTMF. There exists today no unified approach for a flexible and adaptable testing architecture. Since the grade of distribution defines the complexity of testing and determines applicable test architectures, a taxonomy of testing is given in Table 1. In this section a definition of a generic test system architecture is presented which equally well fits the different testing types in Table 1.

Table 1: Testing taxonomy (with respect to the grade of distribution)

System under Test	Test System	Selected Approaches
centralized	centralized	CTMF peer-to-peer
centralized	distributed	CTMF multi-party testing context

Table 1: Testing taxonomy (with respect to the grade of distribution)

System under Test	Test System	Selected Approaches
distributed	centralized	Test execution of telecommunications services (Lima et al, 1997)
distributed	distributed	General design rules for distributed tester

4.1 Generic Test Architecture Model

The idea for a generic test architecture is a tool box of elements, which can be combined generically to a test architecture suitable for a specific application or system to be tested. A test architecture comprises several instances of different types of components. The types are:

- implementation under test (IUT), i.e., the implementation or a part of the distributed application to be tested;
- interface component (IC), i.e., a component which is needed for interfacing IUTs, e.g., an underlying service or an application in which an IUT is embedded;
- test component (TC), i.e., a component which contributes to the test verdict by coordinating other TCs or controlling and observing IUTs.
- controlled component (CC), i.e., 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;
- communication point (CoP), i.e., a point at which communication takes place and at which communication can be observed, controlled or monitored;
- communication link (CL), i.e., a means for describing possible communication flows between TCs, IUTs, NCCs and CCs;
- system under test (SUT), i.e., a combination of ICs and IUTs.

For describing a test architecture in an intuitive and understandable manner a graphical representation might be preferable. The different types of components are explained by using the test architecture shown in Figure 9.

4.1.1 Implementation Under Test (IUT)

An IUT is meant to be a piece of software or hardware to be tested. The entire application to be tested may comprise several IUTs. The IUTs may have different functionality or be different instances of the same type, i.e., symmetrical peer entities of a protocol. An IUT is a black box which can be observed and controlled either di-

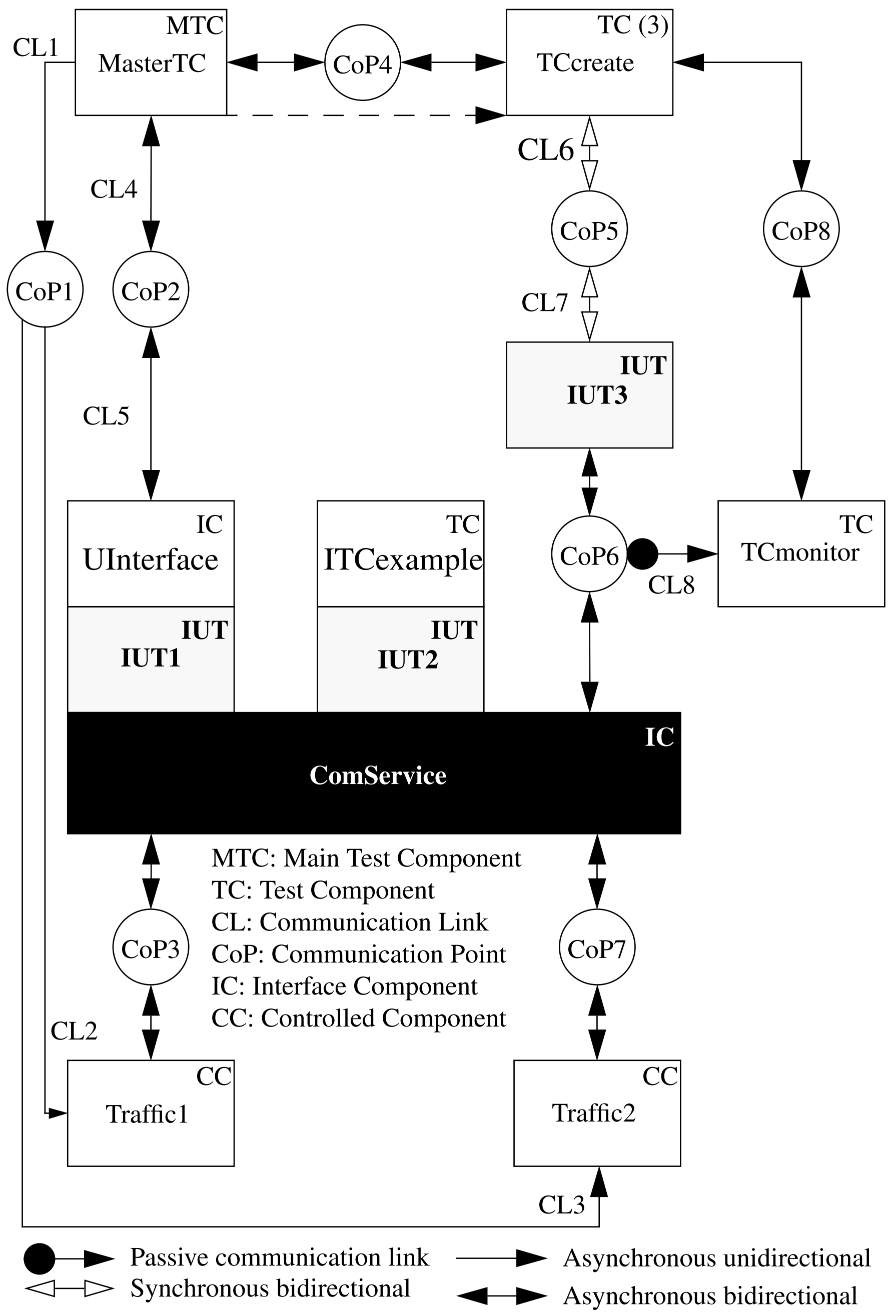


Figure 9 Generic test architecture.

rectly via CLs, or indirectly via ICs.

The test architecture shown in Figure 9 includes three IUTs. They are represented by means of boxes which are inscribed with the keyword IUT in the upper right corner and an IUT name, i.e., in our example IUT1, IUT2 and IUT3.

4.1.2 Interface Component (IC)

An IUT may be embedded in other applications or may be only interfaced via underlying services. The components are termed Interface Components (IC). An IC is not controlled by the test equipment, it is only used to interface the IUT. For testing, it is assumed that an IC is working correct. Within a test architecture there may exist several ICs.

Figure 9 includes two ICs. IC ComService provides a communication service between the three IUTs and IC UInterface describes an interface to the IUT IUT1.

4.1.3 Test Component (TC)

A TC is a component which drives the test. This can be done by creation of further TCs, by controlling other TCs, by contributing to evaluate a test run, and by controlling and observing IUTs.

There should be one *Main Test Component* (MTC) which starts and ends a test run. A start is done by creation and instantiation of further TCs or by initiating the first stimuli to the IUTs. Ending a test run does not necessarily mean to stop all communication and to stop all TCs. It means that the MTC should be able to indicate when the run is finished. When a test run is finished a final verdict is assigned or, in case that for statistical evaluation several runs are required, it should be decided whether the run contributes to the statistics or not.

In addition to TCs and MTC, implicit TC (ITC) is a third type of TC. An ITC corresponds to the UT function in the remote test method of CTMF (Section 2.1). In CTMF the remote test method is used if no standardized interface above the IUT can be used. This means, an ITC indicates the existence of a TC, but how an ITC communicates with other components of the test architecture is not specified.

Figure 9 presents four TCs, one MTC called MasterTC, one ITC (ITCexample) used to interface IUT2 and two normal TCs called TCcreate und TCmonitor. As indicated by the dashed arrow, TCcreate is created by the MTC. During the test run the MTC may create further TCs of the same type. This is indicated by the number in parentheses following the keyword TC. This number describes the maximal number of instances of the same type to be created during a test run. In the example, MasterTC is able to create three TCcreate instances. Omitting the parentheses means by default that there is only one instance and empty parentheses describes an undefined number of instances.

Use and specification of TCs needs special support in the used test specification language. In TTCN, for example, the handling of the UT function within the remote test method is supported by means of implicit send signals. Other points to be handled in the used test specification language are the creation of initially existing TCs when a test run starts, the stopping of TCs when a test run finishes, the evaluation of test runs by TCs, and the communication among TCs, e.g., addressing in case of dynamic TC creation.

4.1.4 Controlled Component (CC)

A CC is a component which is used to set-up the test case specific environment or is used by TCs to control test execution. Examples of CC types are load generators for providing background load, emulators which may be used instead of ICs, or simulators which can be used for comparing the reactions of an IUT with the output of a simulator.

From an abstract point of view a CC may be seen as a TC. Our intention for introducing the CC type is to distinguish explicitly between control and environment of a test. In performance testing (ATM Forum, 1994) a similar distinction is made by using the terms foreground and background load.

Figure 9 includes the CCs Traffic1 and Traffic2. Both generate and manage background load for the IC ComService.

4.1.5 Communication Point (CoP)

A CoP in the generic test architecture corresponds to the PCOs in CTMF. It denotes a point in the test architecture where communication can be observed, controlled, and, in addition to CTMF, be monitored. It is allowed to access several communication flows at the same CoP. From this point of view, a CoP bundles communication flows. For simplicity, no special semantics are assigned to CoPs; note that in CTMF PCOs have a FIFO queue semantics. The semantics of the communication is given to CLs.

In Figure 9 eight CoPs, named CoP1, CoP2, ..., CoP8, are used. They are used to describe the CoPs relevant for testing between TCs, IUTs, ICs, and CCs. Points of communication which cannot be accessed, e.g., between IUT1 and UInterface, or from which the test specification may have to abstract, e.g., between IUT2 and ITCexample, are not described.

4.1.6 Communication Link (CL)

IUTs, ICs, TCs and CCs are connected with CoPs by using CLs. A CL describes a possible communication and the kind of communication which may take place. Active communication can be classified by the kind, i.e., either synchronous or asynchronous, and the direction, i.e., either unidirectional or bidirectional. In addition, a passive CL allows to monitor communication, i.e., to listen at a CoP.

The test architecture shown in Figure 9 includes 18 CLs (only the CLs used in the discussion below are annotated with labels) describing several types of communication. For example, asynchronous unidirectional communication from MasterTC to

Traffic1 and Traffic2 is realized by using CL1, CL2 and CL3 via CoP1. This communication will only be used to start and stop the CCs (for instance, in form of broadcast messages). Bidirectional asynchronous communication between MasterTC and UInterface is described by using CL4 and CL5. Bidirectional synchronous communication between IUT3 and TCcreate is defined by CL6 and CL7 (via CoP5). CL8 describes a passive CL. TCmonitor monitors the communication between IUT3 and ComService.

For the dynamic creation of TCs it is assumed that CLs are also created dynamically and that the CoPs used for communication are all known before test execution, i.e., CoPs cannot to be created dynamically. For the example in Figure 9 this means that during a test run there may exist up to three instances of CL6.

For the different testing requirements, this basic components are combined so that a testing architecture results which fits the specific needs of an application. It has to be noted that for some specific cases, for instance, real-time testing, additional test specific information has to be coded in the dynamic behaviour of a test case. Not all information relevant for testing can be mapped to testing architecture only.

4.1.7 System Under Test (SUT)

In CTMF an IUT together with ICs is called system under test (SUT). The term SUT is used, but in contrast to CTMF, an SUT includes several IUTs. An example of the use of SUTs within a test architecture for interoperability testing is shown in Figure 10.

4.2 Evaluation of Requirements from Advanced Distributed Systems

In the following an assessment is given whether and how the defined requirements of testing advanced distributed system are met by the generic test architecture:

- Development of distributed testing architectures with means for synchronizing distributed test components: With the generic test architecture a support for distributed test architectures is given. Test system and SUT may be arbitrarily distributed over real systems. Support for synchronizing distributed test components is given. Communication of synchronization information along communication links is supported.
- Support of dynamically configurable and scalable test architectures: Firstly, test components can be created on demand by the main test component. Secondly, more than one instance of a specific test component may exist in the system. Together, this gives a test case specifier some initial degree for a dynamic configuration of test architectures.
- Ability to express test configurations for different communication scenarios (unicast, multicast, or broadcast): Communication links support unicast communication, either one-way or two-way. By adding the concept of communication points, multicast communication scenarios can be defined. Attaching a test component to a communication point gives this component access to all data going

through the communication point.

- Possibility to use grey-box testing with access to internal components and interfaces: By introducing passive communication links a means for monitoring even inside the SUT is given. However, an CoP has to be defined within the SUT.
- Support of real-time, performance and QoS testing for distributed systems to test time-related aspects of distributed systems: The generic test architecture is an essential component which is required in the mentioned types of testing. In performance testing, for instance, background test components are used for overloading the SUT. These test components map to controlled components. However, the challenges in real-time, performance and QoS testing are more on how to define test cases; which notation to be used and how test cases are to be implemented.
- Support of interoperability testing to focus on essential interoperability aspects: See Section 5.
- Development of a test methodology that is coherent with object-oriented technologies which are used for the development of distributed systems: In this paper the focus is on test architectures, and not on a test methodology which is more than just a test architecture, e.g., does also consider test case specification and generation, test implementation, test execution, test result analysis etc.
- Test architectures that make use of management and/or monitoring and measurement systems: These specific components can be integrated in a specific instantiation of the generic test architecture. The basic support is given, for instance in terms of passive communication links to monitor and to measure systems.
- Methods that support testing in the pre-deployment, deployment and usage phase of distributed systems: This again is a question of methodology. Testing as understood by CTMF is pre-deployment testing.
- Efficient testing methods in order to deal with the complexity of distributed systems: The generic test architecture has been invented with this requirement in mind. In the following section it is shown how problem specific test architectures are built from the basic components of the generic test architecture.

5 TEST ARCHITECTURES FOR DISTRIBUTED SYSTEMS

For proving the generality of our approach the different test architectures described in Section 2 have to be mapped to the generic model. The mapping of the conformance testing architectures according to CTMF is straightforward because the development of the generic model starts with the CTMF concepts and there is a one-to-one mapping for most of the CTMF concepts onto the generic model. Therefore, in the following an extended interoperability testing architecture and a performance testing architecture are described by using our generic model.

5.1 Interoperability Testing Architecture

Figure 10 maps the interoperability testing architecture proposed by the ATM Forum (Section 2.4.1 Figure 5) to the generic model. The CoPs A, B, C specify explicitly the points where communication is monitored. For the task of the monitoring TCs MA, MB and MC have been introduced.

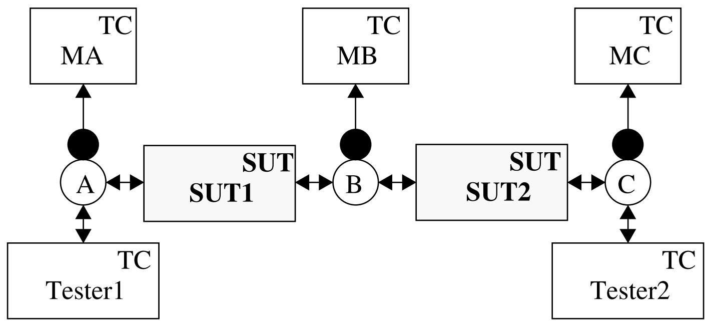


Figure 10 ATM Forum interoperability architecture.

In Section 2.4.1 an extended test architecture for interoperability testing is proposed. Figure 11 shows a transformation of the interoperability test architecture to the generic model. In order to have control over the communication of the IUTs which have to interoperate, the used communication service is emulated by the CC Emulator. The component Emulator communicates with TC Monitor, thus reporting all communication between the IUTs and to receive commands in case the emulator should actively influence the communication, e.g., decrease the performance or corrupt data packets.

5.2 Performance Testing Architectures

Figure 12 shows an instance of the performance testing architecture sketched in Figure 6 (Section 2.4.2) by using the generic model. Background Tester (BT1, BT2) are meant to be load generators and therefore are mapped to CCs. The CC Monitor measures of the real network load during a test run. The Foreground Tester (UFT1, UFT2, LFT1, LFT2) are mapped onto TCs. PCOs and measurement points are mapped onto CLs, their physical interface is described by using CoPs. Additionally, an MTC communicates with the other components in order to start and stop a test run.

6 CONCLUSIONS

In this paper a generic test architecture for conformance, interoperability, perform-

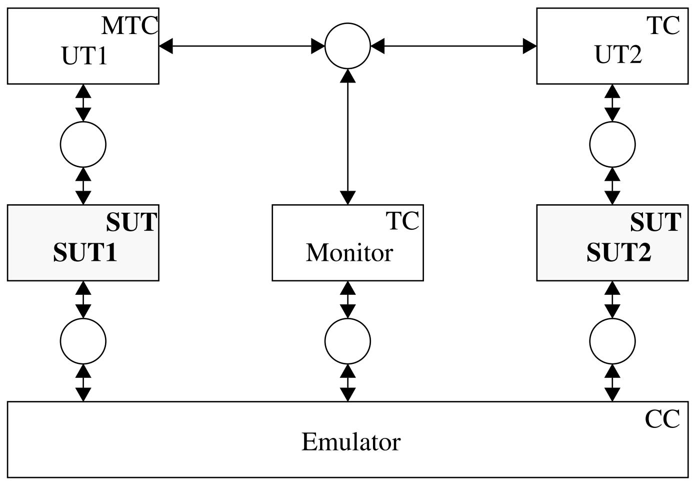


Figure 11 Interoperability Test Architecture.

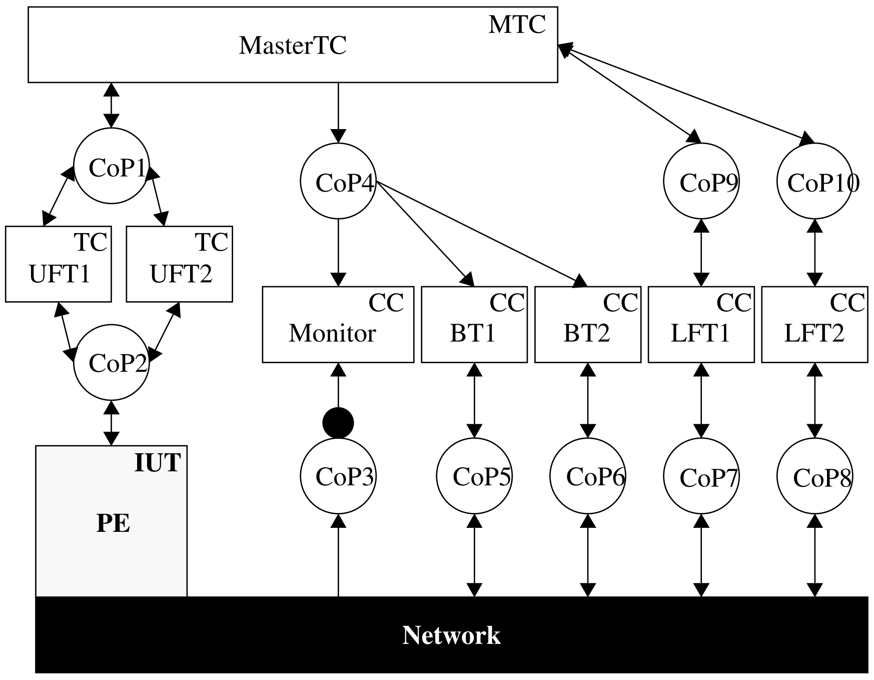


Figure 12 Performance test architecture.

ance and real-time testing has been proposed. The development of the generic test architecture has been motivated by the observations that (1) the abstract test methods defined in the CTMF context are too restrictive with respect to the types of systems that can be tested; and (2) the up-coming new kind of advanced distributed (object, real-time, safety-critical) systems require a flexible and adaptable test architecture. The proposed generic test architecture has been designed as a toolbox whose components can be configured as needed and, thus, provide the required flexibility to set up any required test system configuration.

The discussion of the state-of-the-art in test architectures has been done looking at a test system as a distributed system that consists of active components, named test components, service provider or IUT in CTMF, and passive components PCOs or CPs. The latter define a static communication structure over the active components. In the generic test architecture this point of view has been developed further. Its applicability to different test scenarios has been demonstrated.

For the future a language support for the new test architecture will be investigated.

7 REFERENCES

ATM Forum (1994) *Introduction to ATM Forums Test Specifications*. af-test.0022.pdf, http://www.atmforum.com/atmforum/specs/approved.html.

Baumgarten, B. and Giessler, A. (1994) *OSI Conformance Testing Methodology and TTCN*, Elsevier.

Blair, G.S. and Stefani, J.-B (1998) *Open distributed processing and multimedia.* Addison Wesley Longman Ltd.

Buehler W. (Ed.) (1994) *Introduction to ATM Forum Test Specifications, Version 1.0.* ATM Forum Technical Committee, Testing Subworking Group, af-test-0022.000.

Danthine, A., Baguette, Y., Leduc, G., Léonard, L. (1992) The OSI 95 Connection-Mode Transport Service - The Enhanced QoS, in *High Performance Networking* (ed. A. Danthine and O. Spaniol), IFIP.

Danthine, A., Bonaventure, O. (1993) From Best Effort to Enhanced QoS. CIO Deliverable, No. R2060/ULg/CIO/DS/P/004.

Digital Equipment Corporation (1992) *Distributed Computing Environment Application Development Reference.* Maynard, Maryland, U.S.A.

European Workshop for Open Systems ETG 028 (1993) *Interoperability Classification*. Brussels.

Gadre, J., Rohrer, C., Summers, C., Symington, S. (1990) A COS Study of OIS Interoperability. *Computer Standards & Interfaces*, **9**, 217-237.

Grabowski, J. and Walter, T. (1995) Testing Quality-of-Service Aspects in Multimedia Applications, in *Protocols for Multimedia Systems 2nd Workshop*, Salzburg, Austria.

Hogrefe, D. (1990) Conformance testing based on formal methods, in *FORTE 90 Formal Description Techniques* (ed. J. Quemada, A. Fernandez).

Hopkinson T. (Ed.) (1996) *Scoping Further Activity for Interoperability Testing*. European Workshop for Open Systems EWOS/TA PT N034, Final Report Version 1.0 EWOS EGCT/96/130 R1, Brussels.

IETF Differentiated Services Working Group (1998) *An Architecture for Differentiated Services*. Internet-Draft, draft-ietf-diffserv-arch-00.txt, http://www.ietf.org/html.charters/diffserv-charter.html.

IETF Integrated Services Working Group (1998) *Integrated Services* http://www.ietf.org/html.charters/intserv-charter.html.

ISO International Standard 7498 (1984) *Information processing systems - Open Systems - Basic Reference Model.*

ISO International Standard 9646-1 (1995) *Information technology - Open Systems Interconnection - Conformance Testing Methodology and Framework Parts 1: General concepts*.

ISO International Standard 9646-2 (1995) *Information technology - Open Systems Interconnection - Conformance Testing Methodology and Framework Parts 2: Abstract Test Suite specification*.

ISO International Standard 9646-3 (1996) *Information technology - Open Systems Interconnection - Conformance Testing Methodology and Framework Parts 3: The Tree and Tabular Combined Notation (TTCN).*

ISO International Standard 10746 (1991) *Information processing systems – Open Systems Interconnection: Reference Model for Open Distributed Processing, Part 2: Descriptive Model.*

ITU-T Recommendation X.200 (1989) *Data Communication Networks Open Systems Interconnection (OSI) Service Definitions Recommendations X.200 - X.219*.

Lima Jr., L.P. and Cavali, A. R. (1997) Test Execution of telecommunications services, in *Proc. of IFIP FMOODS '97*, Canterbury, UK.

Linn, R. (1989) Conformance Evaluation Methodology and Protocol Testing. *IEEE Journal on Selected Areas in Communications*, **7**, 1143 – 1158.

Lockhart, H.W. (1994) *OSF DCE – Guide to Developing Distributed Applications.* McGraw-Hill, New York, U.S.A.

Myungchul K., Gyuhyeong K., Yoon D. C. (1996). *Interoperability Testing Methodology and Guidelines*. Digital Audio-Visual Council, System Integration TC, DAVIC/TC/SYS/96/06/006.

Object Management Group (1995) *The Common Object Request Broker: Architecture and Specification; Revision 2.0.* Framingham, Massachusetts, U.S.A.

Open Software Foundation (1998). *http://www.opengroup.org*.

Probert, R. and Monkevic, O. (1992), TTCN: The International Notation for Specifying Tests of Communications Systems. *Computer Networks and ISDN Systems*, **23**, 111 – 126.

Rao, S., BeHanna, Ch., Sun, M., Forys, F. (1997) *CORBA Service Test Environment*. NEC Systems Laboratory, Inc.

Sarikaya, B. (1989) Conformance Testing: Architectures and Test Sequences. *Computer Networks and ISDN Systems*, **17**.

Schieferdecker, I., Stepien, B., Rennoch, A. (1997) PerfTTCN, a TTCN language extension for performing testing, in *Testing of Communication Systems Volume 10* (ed. Myungchul Kim, Sungwon Kang, Keesoo Hong), Chapman & Hall, 21-36.

Schill, A. (1996) *DCE - das OSF Distributed Computing Environment, Einführung und Grundlagen*. Springer Verlag.

Tennenhouse, D., Smith, D., Sincoskie, D., Wetherall, D., Minden, G. (1997) A survey of active networks research. *IEEE Communications Magazine*, **35**, 80-86.

TINA-C (1998) *http://www.tinac.com.*

Walter, T. and Grabowski J. (1997) Real-time TTCN for testing real-time and multimedia systems, in *Testing of Communication Systems Volume 10* (ed. Myungchul Kim, Sungwon Kang, Keesoo Hong), Chapman & Hall, 37-54.

Zeng, H.X., Chanson, S.T., Smith, B.R. (1989) On Ferry Clip Approaches in Protocol Testing. *Computer Networks and ISDN Systems*, **17**, 77-88.

Schütz, W. *On the Testability of Distributed Real-Time Systems*. Report of the ESPRIT Basic Research Project 3092 *Predictably Dependable Computer Systems*.

8 BIOGRAPHY

Thomas Walter received his Diploma in Informatics from University of Hamburg and his Doctorate degree in electrical engineering from the Swiss Federal Institute of Technology in Zurich (ETHZ) in 1987 and 1993, respectively. Since 1986 he is with the Computer Engineering and Networks Laboratory (TIK) of ETHZ, and in 1994 he became lecturer at ETHZ. His current research interests include formal methods for specification and validation of real-time systems. Besides this he is also active in setting up an infrastructure for teleteaching at ETHZ.

Ina Schieferdecker studied mathematical computer science at the Humboldt University in Berlin and received her Ph.D. from the Technical University in Berlin in 1994. She attended the postgraduate course on open communication systems at the Technical University in Berlin. Since 1993, she is a researcher at GMD Fokus and a lecturer at Technical University Berlin since 1995. She is working on testing methods for network components, and carries out research on formal methods, performance-enhanced specifications and performance analysis.

Jens Grabowski studied computer science and chemistry at the University of Hamburg, Germany, where he graduated with a diploma degree. From 1990 to October 1995 he was research scientist at the University of Berne, Switzerland, where he received his Ph.D. degree in 1994. Since October 1995 Jens Grabowski is researcher and lecturer at the Institute for Telematics in Lübeck, Germany.