An Approach for Performance Tests of CORBA based Systems

Ina Schieferdecker

ABSTRACT

This paper presents means to test the functionality, scalability and performance of distributed telecommunication applications based on CORBA ORBs. The presented performance test approach is generic and can be used to cope with various test objectives. For example, it can be used to evaluate the maximal performance of a server device in a networked environment. It is also of use to evaluate response times of servers and the supported number of simultaneous clients. The test manager Ttman presented here is a tool to setup and parameterize distributed test configuration and to control the test execution. Exemplarily, a TINA access session which is part of the TINA platform developed by GMD FOKUS is considered.

KEYWORDS: Distributed Testing, Test Synchronization, Test Coordination, TINA, CORBA

1. INTRODUCTION

Due to the highly increasing complexity of new telecommunication services and the need for more scalable and manageable as well as flexible, run-time configurable execution environments for telecommunications services (telecommunication platforms) new technologies for such platforms are needed. The next generation of telecommunication platforms is based on distributed object technology. A general framework for telecommunication and information retrieval services based on distributed object technology is defined by the Telecommunications Information Networking Architecture Consortium (TINA-C).

Performance testing of real applications in realistic test scenarios and test configurations is of major importance for the assessment, evaluation and overall acceptance of TINA technology (and of distributed object technologies in general). This paper presents a general approach to testing the performance, robustness and scalability of distributed systems. The TINA access session which is part of the TINA platform developed by GMD FOKUS was the TINA implementation which has been tested. To enable distributed performance testing a flexible test architecture has been implemented based on Common Object Request Broker Architecture (CORBA) technology [2]. This test architecture allows to start and configure any number of TINA test clients simultaneously on any network node and to collect their results after the tests have been completed. By increasing the number of simultaneously working test clients the performance characteristics of the TINA access session server can be tested.

Performance testing with a high number of test components requires a flexible solution for establishing different configurations, initiation of the tests and the evaluation of the results. The normal case is that even for a single test case the test configuration is not fixed, but is modified to in order to determine the maximal capacity of the system under test. A continuously increasing number of test components is used to identify the response time of the access session server under increasing load conditions.

The test components themselves form a distributed application which has to be managed. The distribution of the test components is necessary because they run concurrently and should not influence each other - this cannot be guaranteed if the amount of test components becomes too high on a single node. In a distributed test setup two synchronization aspects can be identified: time and functional synchronization for test setup, maintenance and clearing, test execution, and test reporting. Test setup is required to bring all involved entities, like communication channels, testing devices, test components, etc. into a well defined state, so that the test operator is able to execute the test. Possibly, a set of parameters required for proper execution of a test suite have to be distributed to the testing components. The test execution is controlled via coordination messages such as 'start test' and 'report test results'. After a test suite has been completed, the testing devices and the communi-

cation channels have to release occupied resources, so that the testing devices are able to perform another testing session. The process of gathering results produced by a test or a test campaign is denotedby the term test reporting. A test operator can request traces produced by the test components at the testing devices. This information has to be delivered to the test operator using the desired granularity. Either all test devices have to report the traces, or only a specific one. The test result is considered to be transmitted to the test operator via the notification of a completed test case.

Basic Concepts of the Test Synchronization Protocol TSP1 [3] by ETSI have been used to implement the main and parallel test components for the performance tests of the service access session. The presented performance test approach is generic and can be used to cope with various test objectives. For example, it can be used to evaluate the maximal performance of a server device in a networked environment. It is also of use to evaluate response times of servers and the supported number of simultaneous clients. The performance test results can be used to determine optimal parameter settings for the server such as thread count, main memory, etc. The performance test results can also be used as a basis for the decision whether a server device has to be upgraded or whether a better parameter tuning is sufficient to come up with better server performance.

2. A DISTRIBUTED AND SCALABLE PERFORMANCE TEST ARCHITECTURE

RM-ODP [10, 11] describes principles for conformance assessments of ODP specifications and implementations so that implementations of different vendors can interoperate. These conformance assessments made in RM-ODP are essential and the testing of the implementations has to be performed for increasing the probability that applications can operate in the environment and interoperate with each other. Depending on the objectives of testing, distributed applications can be tested with a local or distributed testing configuration. Looking at networking testing similar issues can be identified. For example, if the routing capabilities of a network has to be tested, access to the network at the remote side is needed. The distribution of the test components to the remote side can give the desired access.

In the area of Open Systems Interconnection (OSI) protocols, the Conformance Testing Methodology and Framework (CTMF) is well established and widely used. It is defined in the multipart standard IS 9646 [6]. CTMF was not aimed at describing tests for distributed systems but rather for OSI communication protocols. Although multiple upper and lower testers are supported, one assumption in CTMF is that the Implementation Under Test (IUT) is not distributed. This leads to the fact that the test coordination procedure needed between the several testers are not explicitly specified. However, concurrent TTCN (C-TTCN) [7], which enables the use of independent and concurrent test components (TC) in a test description, gives direct raise to distributed test setups. A Main Test Component (MTC) communicates with Parallel Test Components (PTC) over Coordination Points exchanging Coordination Messages (CM). C-TTCN gives convenient means to describe abstract tests for distributed systems, however, distributed test setups and the synchronization of distributed test components are subject of current research.

Testing distributed applications results in general in distributed test setups, where test components have to be distributed to gain access at the remote ends of the tested application. The distribution of the test components is also needed because they should not influence each other - this cannot be guaranteed if the amount of test components becomes too high on a single node, what is e.g. of particular importance for performance tests.

Performance testing with a high number of test components requires a flexible solution for establishing different configurations, initiation of the tests and the evaluation of the results. The normal case is that even for a single test case the test configuration is not fixed, but is modified to in order to determine the maximal capacity of the system under test. For example, a continuously increasing number of test components is used to identify the response time of ta server under increasing load conditions. The test components themselves form a distributed application which has to be managed. The distribution of the test components is necessary because they run concurrently and should not influence each other - this cannot be guaranteed if the amount of test components becomes too high on a single node.

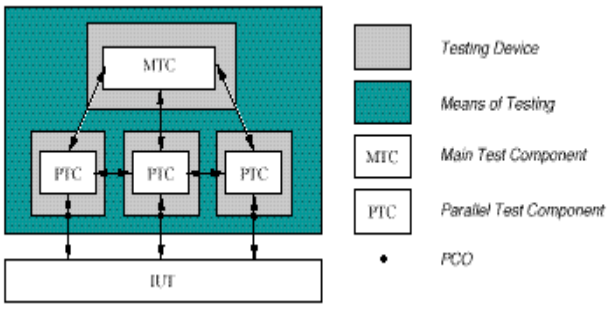


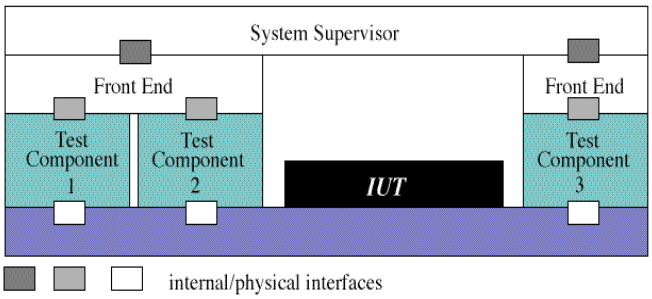
Fig. 1 Generic Distributed Test Architechture

Figure 1 displays the generic configuration for performance testing of distributed system. Each performance test is realized by a set of parallel test components realizing the individual test behaviour such as the emulation of client behaviour, and by a main test component, which controls and coordinates the other parallel test components. Every test component and the test manager, i.e. every test entity, may reside on a separate tester. No resource sharing except of sharing of communication links can take place. For example, the resource time, one of the most important resources can not be shared between two entities that do not reside on the same testing device. Time synchronization needs to take place. The fact that the coordination message exchange may cross the bound-aries of a single tester requires internetworking between the single testers. Reliability of the inter-network is a assumed.

In a distributed test setup two synchronization aspects can be identified: time and functional synchronization. In the following we will describe the functional synchronization only1.
Functional synchronization is needed to perform:
test setup, maintenance and clearing
test execution and
test reporting.

Test setup is required to bring all involved entities, like communication channels, testing devices, test components, etc. into a well defined state, so that the test operator is able to execute the test. Possibly, a set of parameters required for proper execution of a test suite have to be distributed to the testing components. The test execution is controlled via coordination messages such as 'start test' and 'report test results'. After a test suite has been completed, the testing devices and the communication channels have to release occupied resources, so that the testing devices are able to perform another testing session.

The process of gathering results produced by a test or a test campaign is denoted by the term test reporting. A test operator can request traces produced by the test components at the testing devices. This information has to be delivered to the test operator using the desired granularity. Either all test devices have to report the traces, or only a specific one. The test result is considered to be transmitted to the test operator via the notification of a completed test case. At any stage during the test execution it has to be assured that all test components are in a known and stable state.



[1] Time synchronization is of less complexity for the described performance tests, since each PTC calculates its execution time locally and reports it to the main test component. So, the weak requirement of equal time progress in each test device has to be assumed only.

Fig. 2: Architechture of TSP1

Basic Concepts of the Test Synchronization Protocol TSP1 [3] by ETSI have been used to implement the main and parallel test components for the performance tests of the TINA service access session [4]. The Architecture of TSP1 is presented in Figure 2. The purpose of the TSP1 protocol is to achieve functional coordination and time synchronization between two or more Test Synchronization Architectural Elements (TSAEs). TSAEs are Front Ends (FE), Test Components (TC) and the System Supervisor (SS). A Test Component is executable, i.e. it realizes the logical test behaviour and contains also hardware dependent parts like protocol emulations of the underlaying layer, access to Line Interfaces, etc. The Front End is a server process on each testing device which is involved in the test configuration. It is responsible for delivery of control messages between test components and the system supervisor. One front end is the interface for all test components on a testing device. The System Supervisor takes care of distributing control messages to the appropriate test component via the respective Front End. In fact, the complete test configuration and the distribution of the test components is only known to the System Supervisor.

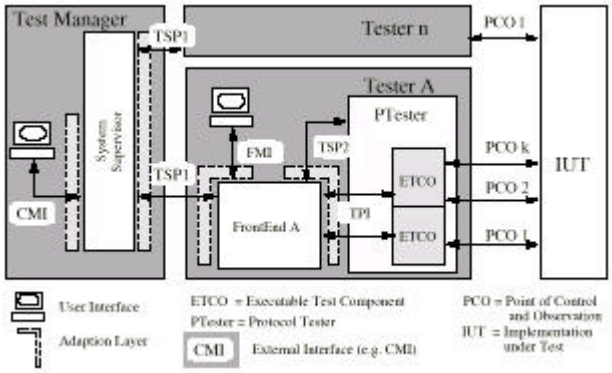


Fig. 3: Library Structure of TSP1

Physical interfaces are used where the two communicating entities do not reside on the same hardware. Mainly this will be the case between the System Supervisor and the Front Ends and the case between the executable test components and the IUT. Internal interfaces will be used were the communicating entities reside on the same hardware. In fact this is true between the Front End and the Executable Test Component. Normally, the Front End will run as a server process on the testing device where the local Executable Test Components also reside.

The exchange of Coordination Messages is managed via the Front Ends in conjunction with the System Supervisor. Every executing test component needing to exchange coordination messages with an other ETCO sends them to the Front End. The Front End then forwards the CM to the System Supervisor if the message's destination is an ETCO not controlled by the Front End. The System Supervisor then takes care of distributing the message to the right Executable Test Component via the appropriate Front End. In fact, the complete test configuration and the distribution of the test components is only known to the System Supervisor.

The functionality of the System Supervisor includes the management of the test execution. It does not provide any support for implementing the necessary configuration on the testing device. The test configuration, i.e. the distribution and availability of test devices must be known and identified in advance, and set up manually or using a Telecommunication Management Network (TMN). The System Supervisor (SS) manages the address table list of the test components, i.e. the mapping between each test component and its FE, has routing capabilities towards the FEs, communicates with the FEs, and manages the test session.

The Front End has two functions. Firstly, it decodes and translates received messages from the System Supervisor to the target testing device. Secondly, it distributes only those messages which are destined for test components not controlled by the Front End. If messages are received from a Test Component local to the Front End and having as destination a Test Component controlled by the same Front End, these messages stay local to the Front End, i.e. they are not sent to the System

Supervisor. The Front End has to have routing capabilities towards its Test Components; communicate with the SS, and to communicate with the testing device.
Executable Test Components are able to handle the test interfaces and are executing the (logical) test component. They operate in the environment provided by the testing device.

As monolithic solutions that incorporate communication services, adaption to the testing devices, etc. are very inflexible, the TSP1 protocol was, firstly, split into a System Supervisor side and a Front End side [1] and, secondly, only the dynamic behaviour of the protocol was formulated in C. Well defined external interfaces provide access to the TSP1 library. The TSP1 library can be written without containing any OS or machine depended code. It has been designed to be compilable at any ANSI C-capable OS. The TSP1 library was already successfully compiled on a SUN ULTRA 10 running Solaris 5.6 and a PC running LINUX with the kernel version 2.0.35. On both systems the GNU C-compiler (GCC, ver 2.8.1 (SUN) and ver. 2.7.2 (LINUX)) was used. The interface at the system supervisor interface is depicted in Figure 4.

Fig. 4: TTman Interface

3. TINA PLATFORM UNDER TEST

This section is to describe the TINA [5] platform implementation at GMD FOKUS which was the system under test outlined in this paper. The platform was designed according to the TINA architecture and consists in principal of access session and subscription components. They are implemented in C++ and run under Windows NT 4.0. The communication between the distributed components is done by means of CORBA mechanisms which are provided by the commercial product Visibroker 3.2. The following subsections describe the structure of the implementation of both components and their environment in more detail.

The access session component is of major concern for the overall system performance. It is forming one process running on Windows NT and consists of implementations for the computational objects defined in the TINA architecture like Initial Agent (IA) and User Agent (UA). That means there is a decomposition of these objects in several C++ class declarations and definitions. Furthermore these computational objects are supporting interfaces according to the Retailer Reference Point (RET-RP) defined by TINA-C like i_RetailerInital and i_RetailerNamedAccess as well as proprietary interfaces which are used internally (see figure). In order to fulfil its task the access session needs information from subscription. Therefore another process is running on the same node containing the subscription component. It contains implementation for several computational objects whereas one of them the Subscription Coordinator (SC) is of main interest for the access session. It supports an interface which provides all the necessary information to the access session. Subscription itself retrieves these information from an object-oriented database realized with Versant.

In order to make some interface references from subscription known to the access session and known to the test component a CORBA name service has to be executed. In the relevant test configuration the name service coming with Visibroker for C++ 3.2 was used. It is running on the same node like the other components under test.

The access session uses another component (UADB) to get access to the already mentioned object-oriented database, where all user information are stored. This component runs in a separate process on the same node and is also implemented in C++.

4. PERFORMANCE TESTS

Testing distributed applications encompasses two steps. In a first step the functional aspects of the system under test is verified, i.e. it is checked whether the system behaves in the target environment like expected and whether it is conform to reference points. Once the conformance of the system under test is checked, performance and robustness tests can be performed to determine whether the system also behaves correct under load. This paper concentrates on performance testing in the second step. In essence, the performance test is an evaluation of the responsiveness of the access session server of the TINA platform and of its scalability. Therefore, parallel test components are used to emulate the behaviour of clients individually and to emulate the simultaneous access of several clients to the access session server.

A number of such parallel test components (PTCs, for short referred to as TC), which emulate the client behaviour, will be triggered to run the test behaviour when the system under test is up and running and the test configuration is set up.

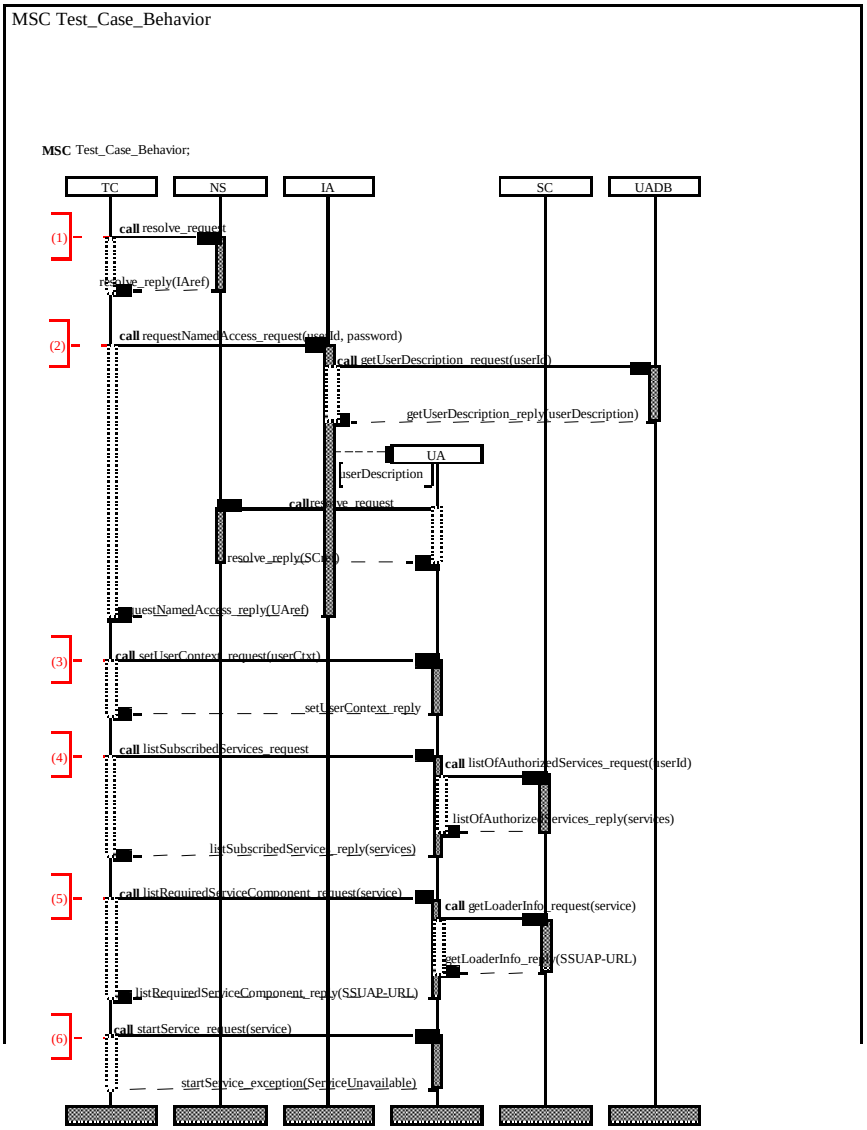


Fig. 5: The Test Behaviour for Emulated Clients

Figure 5 depicts the test behaviour as a Message Sequence Chart (MSC [8]).

1. The test component (TC) resolves a name context at the name service to retrieve the interface reference (*i_RetailerInitial* interface) to the Initial Agent (IA).
2. This interface reference is used to call the *requestNamedAccess* operation at that interface. The parameter *userId* has the value *anonymous*, the password is an empty string. This operation request causes

the IA to initiate a database request to the UADB object to get some properties for that user (*userDescription*). In the case that the *userId* is *anonymous* the IA instantiates a new User Agent (UA), initializes the UA with the user description and returns the interface reference (*i_RetailerNamedAccess* interface) of the UA to the TC. In its initialization phase the User Agent resolves a name context at the name service to retrieve the interface reference to the Subscription Coordinator (SC).

3. The TC calls the operation *setUserContext* at the *i_RetailerNamedAccess* interface.
4. In order to retrieve the available services for that anonymous user the TC calls the operation *listSubscribedServices* at the *i_RetailerNamedAccess* interface. Then the UA sends the request *listOfAuthorizedServices* to the SC which provides the information with the help of the underlying database back to the UA. The UA replies the list back to the TC.
5. The TC which acts like a Provider Agent in the TINA architecture needs some information about the service specific user application of the selected service in order to start the service. Therefore it calls the operation *listRequiredServiceComponent*. This causes the UA to send the request *getLoaderInfo* to the SC which retrieves this information from the database. In the current implementation this information consists of an URL to a JAVA applet which implements the service specific user application.
6. After the TC has got the information about the service specific user application it calls the *startService* operation for the selected service. Since no service is running on the platform the UA responds with a *ServiceUnavailable* exception.

We use the upcoming version of Message Sequence Charts (MSC'2000) to identify the events to observe and to define the measurements to be taken in a performance test. An example is given below.

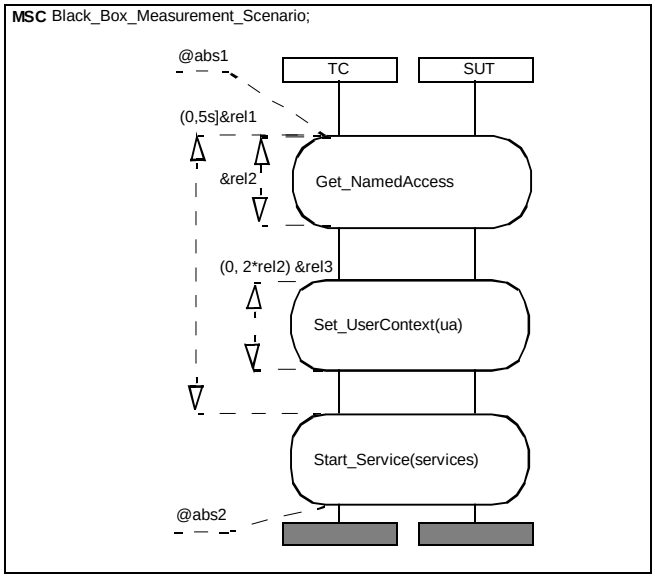


Fig. 6: Defining the Measurements to be taken in a Performance Test

The MSC *Black_Box_Measurement_Scenario* in Figure 6 uses absolute measurements (indicated by @) and relative measurements (indicated by &). Absolute measurements observe the global time. The absolute measurements *@abs1* and *@abs2* give the absolute start and end of the test described as a sequence of *Get_NamedAccess*, *Set_UserContext* and *Start_Service* (we divided the sequence given in Fig. 5 into subsequence in order to have a modular design for the test cases). In addition, relative measurements are used to measure the time distance between pairs of events.
For example, *&rel1* measures the distance between the start event of *Get_NamedAccess* and the start event of *Start_Service*. This measurement is combined with a constraint *(0,5s]* meaning that the duration between start of *Get_NamedAccess* and start of *Start_Service* is constrained and the time it takes really (within the given bounds of the constraint) is observed by means of the relative measurement.

*(0, 2*rel2) &rel3* is also a relative time constraint combined with a relative measurement. This constraint refers to a measurement on the duration between start of *Get_NamedAccess* and its end taken before.
For the concrete test environment we installed the executables of all TINA components on a Windows NT 4.0 PC with 256 MB memory and a dual processor board with 2x Intel Pentium 266 MHz. This PC contains also the database with the user and service information (subscription). The MTC and the Test Manager are implemented as Windows applications as well. They are executed on a Windows NT laptop. The emulated clients are executed on PCs and/or Workstations. First results of the performance tests are given in [9]. A next series of performance tests is under work with an updated version of the TINA Access session server.

5. CONCLUSION

Performance testing of real applications in realistic test scenarios and test configurations is of major importance for the assessment, evaluation and overall acceptance of distributed systems. In addition, performance tests and systematic approaches towards performance tests gain in acceptance.
The presented performance test approach is generic and can be used to cope with various test objectives. For example, it can be used to evaluate the maximal performance of a server device in a networked environment. It is also of use to evaluate response times of servers and the supported number of simultaneous clients.
The performance test results can be used to determine optimal parameter settings for the server such as thread count, main memory, etc. The performance test results can also be used as a basis for the decision whether a server device has to be upgraded or whether a better parameter tuning is sufficient to come up with better server performance.
The performance test approach is in general applicable to any CORBA based distributed system: the mechanism for implementing, setting up, configuring and distributing the emulated clients which give the load to the server, are based on standard CORBA features.
It should be noted at this point, that in the future new features such as on-line performance monitoring and load balancing in distributed process environments will support the routing of client requests as well as the migration to less loaded servers.

ACKNOWLEDGEMENT

The work described in this paper has been performed in a joint effort with Marc Born and Theofanis Vassiliou-Gioles. Valuable insights and fruitful discussions were given by Andreas Hoffmann and Mario Winkler.

REFERENCES

[1] M. Heinrich, E. W. Jüngst. A Resource-Based Paradigm for the Configuring of Technical Systems from Modular Components, in Proceedings Seventh IEEE Conference on Artificial Intelligence Applications, S.257-264. IEEE, 1991
[2] OMG: The Common Object Request Broker Architecture and Specification, Version 2.1. Aug. 1997.
[3] ETSI TC-MTS: Methods for Testing and Specification (MTS); Test Synchronization; Architectural reference; Test Synchronization Protocol 1 (TSP1) specification, ETSI Technical Report ETR 303, Sophia Antipolis, January 1997.
[4] Farley, P.; Hogg, S.; Kristiansen, L. et al.: Ret Reference Point Specifications, Snapshot 1, Version 0.4, TINA-Consortium, May 14, 1997.
[5] TINA-C: Service Architecture Version 4.0, Oct. 1996.
[6] ISO/IEC 9646: Information technology - Open systems interconnection - Conformance testing methodology and framework, International Standard, Geneva,1991.
[7] ISO/IEC 9646-3: Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part3: The Tree and Tabular Combined Notation International Standard, Second Edition, Geneva, 1997.

[8] [Z.120] ITU-T Z.120: Message Sequence Chart (MSC), ITU-T Recommendation, Geneva, 1996.
[9] M. Born, A. Hoffmann, M. Winkler, I. Schieferdecker, Th. Vassiliou-Gioles: Performance Testing of a TINA Platform. - TINA'99, Hawai, May 1999.
[10] ITU-T Rec. X.903 | ISO/IEC 10746-3: 1995, Open Distributed Processing - Reference Model Part 3
[11] ITU-T Rec. X.904 | ISO/IEC 10746-4: 1995, Open Distributed Processing - Reference Model Part 4

AUTHOR

Dr. Ina Schieferdecker, GMD FOKUS Kaiserin-Augusta-Allee 31, D-10589 Berlin, Tel. +49 30 3463 7241, Fax. +49 30 3463 8241, email: schieferdecker @fokus.gmd.de
graduated from Humboldt University Berlin in Mathematical Computer Science and received her Ph.D. from the Technical University Berlin in 1994. She is a researcher at GMD FOKUS since 1993 and a lecturer at Technical University Berlin since 1995. In 1997, she had a research stay at the International Computer Science Institute in Berkeley, California.
As the head of Competence Center for Testing, Interoperability and Performance (TIP) at GMD FOKUS she is responsible for development support for distributed systems and for measurement and testing methods for distributed systems in particular. She carries out research on testing methods, and performance-enhanced specifications, primarily in the context of ODL, MSC, and TTCN applied to ATM, Internet, and CORBA technology. She is involved in ITU-T and ETSI standardisation on MSC and TTCN.