5

CONFIGURATION AND EXECUTION SUPPORT FOR DISTRIBUTED TESTS

Theofanis Vassiliou-Gioles, Ina Schieferdecker,
Marc Born, Mario Winkler and Mang Li
GMD FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin
Tel. + 49 30 3463 7346, Fax + 49 30 3463 8346
email: {vassiliou I schieferdecker I born I winkler I m.li} @fokus.gmd.de

Abstract This paper presents means to test the functionality, scalability and performance of distributed telecommunication applications based on CORBA ORBs. It presents generic tools for testing the functionality, performance, robustness and scalability of distributed systems. The tools cover the test suite simulator TSsim for validating concurrent test suites, the TTCN/CORBA gateway TCgate for automated execution of the test cases (including a generic coder/decoder component), and the test manager TTman to setup and parameterize the test configuration and to control the test execution. The emphasis of this paper is on TTman and its scripting facilities. Exemplarily, a TINA access session which is part of the TINA platform developed by GMD FOKUS is considered.

Keywords: Distributed Tests, Test Scripting, TINA, TSP1

1. MOTIVATION

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. Current telecommunication platforms are mostly based on intelligent networks (IN) technology and do not meet the new requirements any more. In the last few years a lot of research efforts have been made in the research labs all over the world to find new solutions to fit the new requirements of today's telecommunication market. The next generation of telecommunication platforms is based on distributed object technology - a key enabling factor for future telecom-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35567-2_25 munication systems. In order to define a general framework for all kinds of telecommunication and information retrieval services based on distributed object technology most of the large telecommunication companies in all over the world founded the Telecommunications Information Networking Architecture Consortium (TINA-C). Technologically, TINA systems can be implemented on top of Object Request Brokers (ORB) of OMG CORBA.

This paper presents means to test the functionality, scalability and performance of distributed telecommunication applications based on CORBA ORBs. It presents generic tools for testing the functionality, performance, robustness and scalability of distributed systems. Exemplarily, a TINA access session which is part of the TINA platform developed by GMD FOKUS is considered. The test tools allow to start and configure any number of TINA test clients simultaneously on any network node and to collect the results of them after the tests have been finished.

RM-ODP 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 test 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 [8]. 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) 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.

The paper is structured as follows: Section 2. discusses synchronization aspects for distributed test components, gives an overview on the Test Syn-

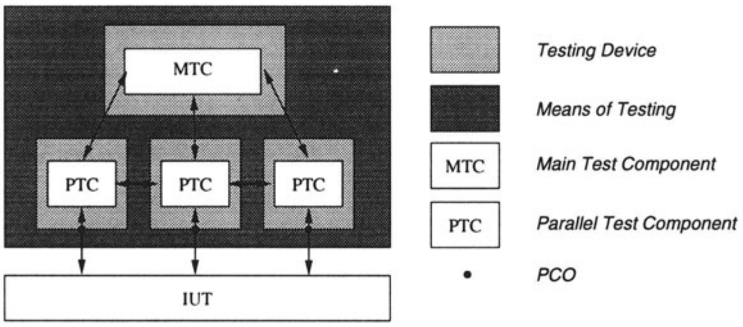

Figure 1 Generic Distributed Test Architecture

chronization Protocol TSP1 and describes an implementation of TSP1 called TTman. Section 3. identifies shortcomings of TTCN test suites in terms of dynamic test case selection and test group verdict assignments and proposes a solution to this: the Test Session Specification Language TSSL. In Section 4. an application example (a TINA Access Session) is described: its conformance tests are defined in C-TTCN and their execution make use of TTman and TSSL. Conclusions finish the paper.

2. A TEST MANAGER FOR DISTRIBUTED TESTING

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. The test system in a distributed test context itself forms a distributed application which has to be managed.

Figure 1 presents the generic configuration of a distributed test system. A distributed test is realized by a set of parallel test components performing the individual test behaviour such as e.g. the emulation of client behaviour for a service under test, 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.

Therefore, two synchronization aspects have to be covered in a distributed test setup: time and functional synchronization. Between and during the execution of test cases it has to be assured that communication links are available and local and remote test components are still "on duty" at the testing devices.

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, where the following techniques could be used:

- synchronization via the Low-Frequency transmitter of PTB or similar institutions (DCF77) [7]
- synchronization via Global Positioning System [6]
- synchronization via Network Time Protocol [11].

In the following, we will concentrate on functional synchronization aspects. Functional synchronization[1] 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 test components. The test execution is controlled via coordination messages such as 'start test' and 'report test results'. After a test suite has been finished, the testing devices and the communication channels have to release occupied resources, so that the testing devices are able to perform another test 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 testing 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.

2.1 OVERVIEW OF TSP1

ETSI MTS has defined the Test Synchronization Protocol 1 (TSP1) for synchronization issues in distributed test setups [1]. 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). For example, in a multi-party testing method (MPTM) according to [8], each lower tester can be defined by a single Parallel

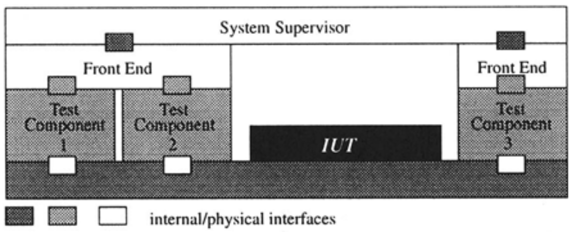

Figure 2 Architecture of TSP1

Test Component (PTC) and the Lower Testing Control Function (LTCF) can be specified by the Main Test Component (MTC). Coordination points between the MTC and the PTC enable communication between the MTC and the PTCs. With TSP1, the System Supervisor has the function of the Lower Tester Control Function (LTCF) and each test component (TC) is a Lower Tester (LT).

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 (ETCO). 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 via Coordination Points (CP) as defined in an abstract test suite 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 testing 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,

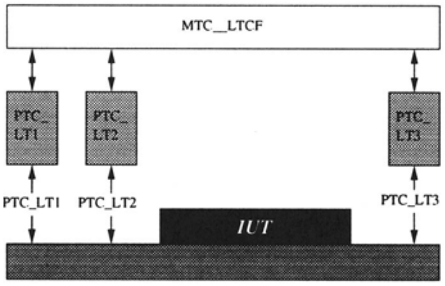

Figure 3 Multi-Party Testing Method, [8]

- 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.

2.2 THE IMPLEMENTATION OF TSP1

For implementing the Test Session Manager TTman on the basis of the TSP1 protocol, the library concept (see also Figure 4) is chosen as portability TSP1 was one of the implementation goals: A TSP1 implementation, either the System Supervisor or the Front Ends must be able to run on different operating systems (OS). Additionally, a TSP1 implementation should be able to use different communication media and/or different transport mechanisms for the communication between the System Supervisor and the Front Ends. Intentionally, [1] does not define the communication services that should be used for carrying the TSP1 PDUs as the choice of the service depends on availability but also on performance requirements. The only requirement that is formulated by [1] is that the underlying service provider shall be reliable. If performance aspects have not the highest priority but low costs of test setup are desired, TCP/IP connections might be chosen as transport layer if IP connectivity exists. If performance requirements have the highest priority, e.g. running performance tests, an ATM connection or an ISDN connection are more appropriate to provide the required service quality.

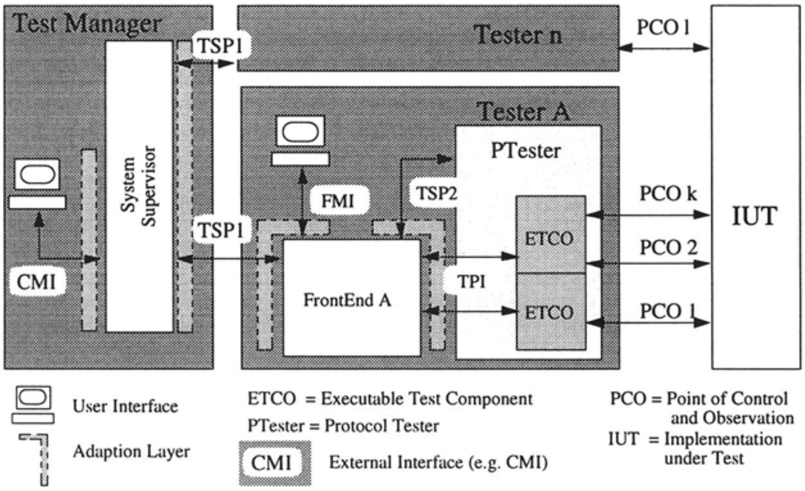
Test Manager
System Supervisor
CMI
TSP1
TSP1
Tester n
Tester A
FMI
TSP2
FrontEnd A
PTester
TPI
ETCO
ETCO
PCO 1
PCO k
PCO 2
PCO 1
IUT
User Interface
Adaption Layer
ETCO = Executable Test Component
PTester = Protocol Tester
CMI External Interface (e.g. CMI)
PCO = Point of Control and Observation
IUT = Implementation under Test

Figure 4 TSP1 Library structure

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 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 5.

3. TEST SCRIPTING FOR EFFICIENT TEST EXECUTION

Testing is a very time consuming process. Often it happens that test cases fail where their successful execution is a prerequisite for other, depending test cases. That means that other test cases have to fail, or at least come to an inconclusive verdict if the prerequisite test cases fail. The execution of such dependent test cases is most often a waste of resources and time.

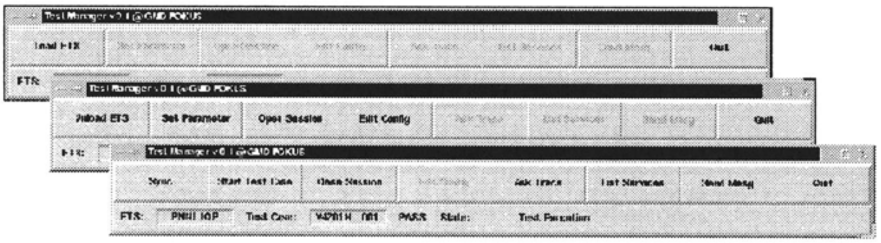

Figure 5 Control Management Interface of TTman

Another aspect paid attention in testing is reusability of test cases, i.e. that test cases might be reused in a context other than they were written primarily for. For reusing a set of test cases (i.e. a test suite) a selection and regrouping of appropriate test cases might be necessary. TTCN does not provided any means for controlling the progress of a test campaign in dependence of test results of already executed test cases. Only the dependence on static requirements, the so called selection expressions, can be used to select and deselect test cases for execution. Also, no means for different views (in terms of groups of test cases and hierarchies of test groups) of a test suite are provided by TTCN.

Finally, TTCN does not support the assignment of verdicts to test groups, which could be defined to be the accumulated verdicts of the contained test cases and/or test groups. Hence, also the assignment of the overall verdict to test suite execution in a test campaign is done in course of a subsequent evaluation of the test case verdicts, rather than via a final calculation of the verdicts of the top most test groups. In particular, there are no specified rules how to interpret the individual test case verdicts with respect to the overall test suite verdict, what opens the door to ambiguous and even contradicting overall assessments of tested systems. In order to overcome these deficiencies, a scripting language Test Session Specification Language (TSSL), will be presented that accommodates the need for dynamic test selection and execution and that enables the test suite to produce different views on the same set of test cases by simply changing the associated TSSL script. The combined use of TSSL and TTman leads to an increased grade of automated test execution and to an improved support for the evaluation of test verdicts.

The use of TSSL is not limited to TTCN test suites, but its application to TTCN is straightforward. TSSL is described subsequently in combination to TTCN in order to ease the reading. TSSL is designed to reside on top of a test suite, i.e. a TSSL script controls the execution of a TTCN test suite. TTCN is used to formulate the test cases, to group them and to define static requirements like parameters. TSSL is an add-on to an existing test suite. It does not replace the test suite or any concepts in the test suite, like selection expressions.

TSSL is based on the concept of GROUP objects which are constituted by the test suite itself, test groups and test cases (which are singleton GROUP objects). The latter two can be declared and modified. The test suite object is implicitly instantiated when the script is executed.

A test session script, that is a script written in TSSL, consists of two parts. A declaration part and a dynamic part, which define the dynamic behaviour of the test session which is based on a test suite. The first declaration of a TSSL script is the reference to the associated test suite. The import of a test suite has two effects. At first, all test cases and groups defined in the test suite can be referenced. Secondly, a predefined variable of type VERDICT, called verdict is instantiated. An instance of a GROUP object has three attributes:

- `[ref | list]`
- `verdict`

An object is either defined by a reference to an existing group in the test suite or by an explicit list of test groups/test cases. A reference to an existing group references all test cases in this group in the order in which test cases appear in the test suite (provided that they are selected due to their static selection expressions). The list attribute can be used to group test cases or groups to a new group not specified in the test suite for easier execution afterwards. In verdict the actual test group verdict is stored. It it accessible from outside the group, after the execution of the test group has finished.

Two methods are provided for GROUP variables. The first one is the exec method (shorthand for execute). The method is run if the test group is executed. Inside the execute method, individual test cases can be executed, the group verdict can be set, and decision and loop constructs enable the granular application of execution and verdict assignment. For example, a group verdict should only be set to PASS if all test cases in the group have been successfully executed a defined number of times. Also, the selection and execution of other test cases based on the result of previously executed test cases can be performed. If no execute method is defined a default execute method applies, i.e. every selected test case in the group will be executed. The group verdict will be calculated according the TTCN rules for a verdict assignment. The verdict can get only "worse", not better.

The second method that can be defined for a GROUP object is the eval method (shorthand for evaluate). This method defines a rule that will be evaluated after each and every executed test case of the group. Depending on the conditions in eval, the execution of the group can be aborted. Using this possibility, the test effort can be reduced as not every possible executable test case needs to be executed. Maybe, the execution of every test case might be not desirable if more than say for example 75% percent of the executable test cases of a group have already failed. TSSL has a loop and a decision construct. They can be used to control the test group and test case execution in either from the main level (i.e. for the test suite) or in the exec method of a GROUP object. A decision can be constructed with an if..else expression, a loop using a while style construct. Boolean expressions in a decisions and loops can consist of relations between the standard data types and verdict attributes. Two different types of functions can be used to control the execution of a TSSL script:

- execution of an object
- gathering of information about an object

An object is executed by use of its exec methods and yields its verdict attribute. If an object has no explicit exec methods, the default behaviour applies, which is to execute every test case within this group where the selection expressions hold, and where the test case was not executed before.

In a given exec method, each test case to be executed is named explicitly and will be executed despite of the verdict it has. If the test suite designer wants to avoid the repeated execution of a test case he can check the test case verdict. The exec method of a test group containing other test groups has to execute each test case of the contained groups explicitly if the groups are included by reference. Only if a test group is included by a test group variable and not by a reference it can be executed according to the test groups exec method. If every test case of a test group is deselected by its selection expressions, a NONE verdict is returned.

The attribute verdict of an object returns the basic information of an object, its verdict. Other functions exist that give a more abstract view on a GROUP object:

- `count_tc`
- `count_verdict`
- `r_count_verdict`

The function `count_tc` returns the number of test cases in an object. The `count_verdict` function return the number of test cases having the specified (i.e.different to NONE) verdict. The function `r_count_verdict` returns the percentage of test cases in an object having a specified verdict. As basis for the calculation either all test cases in an object can be used or only test cases whose selection expression hold. The default basis for calculation are the selected test cases.

4. TINA PLATFORM UNDER TEST

This section describes the TINA platform implementation [3] 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 Visibroker 3.2. The following subsections describe the structure of the implementation of both components and their environment in more detail.

4.1 ACCESS SESSION AND SUBSCRIPTION

The access session component is of major concern in this paper. 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.

4.2 ENVIRONMENT

In order to make some interface references from subscription known to the access session and known to the test component a CORBA Naming Service (NS) has to be executed. In the relevant test configuration the NS 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++. Figure 6 shows the configuration of the platform.

As a precondition for the whole platform the Visibroker 3.2 Smart Agent has to run on the node as well as the Versant demon to use the database which also runs on the same node. This is not depicted in the figure.

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.

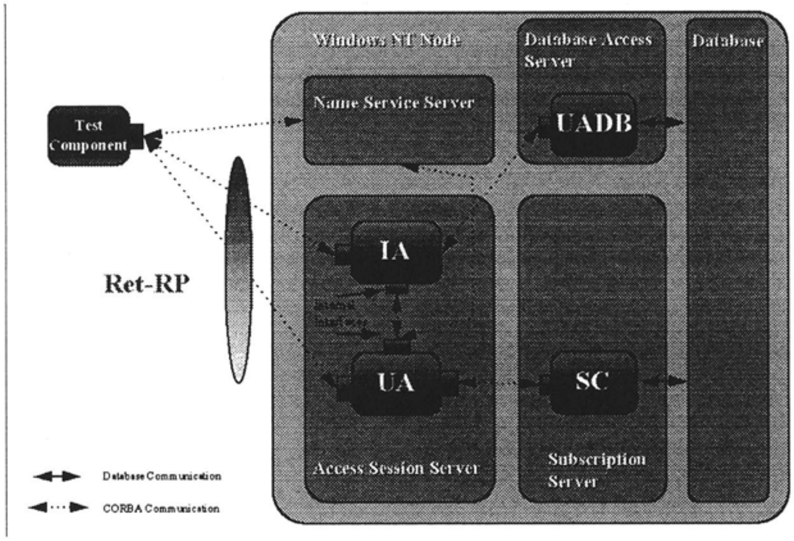

Figure 6 Configuration of the TINA Platform under Test

- 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.

The conformance tests for the TINA platform under test have been made first, the performance tests in a second step. This papers describes in more detail the use of TSSL for the efficient execution of the conformance tests, the performance tests and their results are described in [15].

Figure 7 depicts the test behaviour as a Message Sequence Chart (MSC) diagram in parallel to the following description:

1 The test component (TC) resolves a name context at the NS 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 NS to retrieve the interface reference to the Subscription Coordinator (SC).

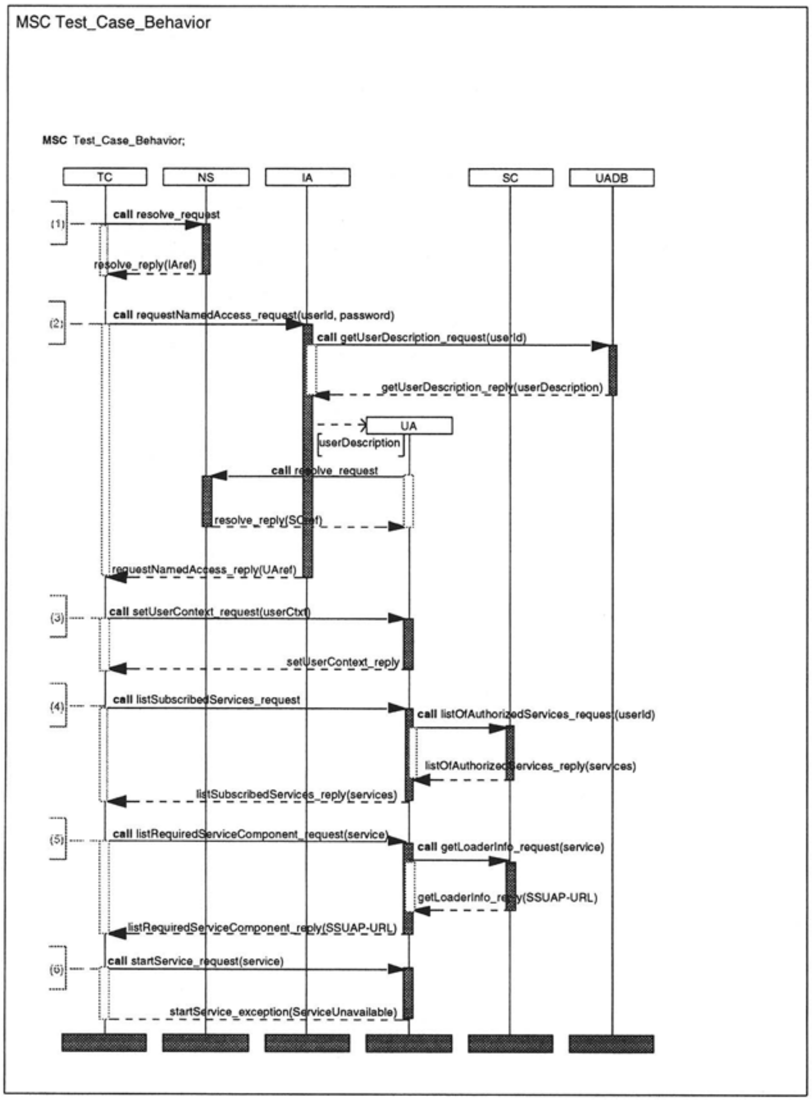
MSC Test_Case_Behavior
MSC Test_Case_Behavior;
TC
NS
IA
SC
UADB
UA
(1)
(2)
(3)
(4)
(5)
(6)
call resolve_request
resolve_reply(IAref)
call requestNamedAccess_request(userId, password)
call getUserDescription_request(userId)
getUserDescription_reply(userDescription)
userDescription
call resolve_request
resolve_reply(SCref)
requestNamedAccess_reply(UAref)
call setUserContext_request(userCtxt)
setUserContext_reply
call listSubscribedServices_request
call listOfAuthorizedServices_request(userId)
listOfAuthorizedServices_reply(services)
listSubscribedServices_reply(services)
call listRequiredServiceComponent_request(service)
call getLoaderInfo_request(service)
getLoaderInfo_reply(SSUAP-URL)
listRequiredServiceComponent_reply(SSUAP-URL)
call startService_request(service)
startService_exception(ServiceUnavailable)

Figure 7 The behaviour of the example test case

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.

The test cases make use of PTCs which describe the test behaviour with respect to the individual interfaces of NS, IA, and UA. TTman is used to setup and parameterize the executable test components in the distributed test setup.

The test cases for Figure 7 are defined step-wise so that every test case makes use of test cases which reflects the preceding operation requests. For example, the test case for *RequestNamedAccess* makes use of the test case for *Resolve_Request*. This makes the usefulness of test case execution depending on the success of the respective preceding test cases, what is reflected in the TSSL script given in Figure 8.

5. CONCLUSION

The presented paper investigated the synchronization of parallel and distributed test components in order to obtain means for the synchronization of distributed test components. There is no standardized framework for distributed testing, however different testing techniques such as conformance testing, performance and interoperability testing require in general distributed test setups for testing distributed systems. The time and functional synchronization of distributed test components are still only partially covered issue in research. The main focus is on functional synchronization of distributed testing components. The test synchronization protocol TSP1 was described, its relevance

```
Basic_capabilities_Access_Session = {
  IMPORT "AS_Tests";
  GROUP InitTest, AccessTest, ServiceStartTest, ...;
  TESTCASE Resolve_Request, RequestNamedAccess, SetUserContext, ...;
  INTEGER tries;
  Resolve_Request.ref = "AS_Tests/GENERAL/Init_Service/Valid/U0";
  RequestNamedAccess.ref = "AS_Tests/GENERAL/Init_Service/Valid/U1";
  SetUserContext.ref = "AS_Tests/GENERAL/Init_Service/Valid/U2";
  ...;
  InitTest.list = { Resolve_Request };
  AccessTest.list = { RequestNamedAccess, SetUserContext, ...};
  ServiceStartTest.list= { InitTest, AccessTest };
  ServiceStartTest.exec()
  {
    if( InitTest.exec ())
     if (AccessTest.exec()) verdict = PASS;
     else { verdict = FAIL; break; }
    else { verdict = FAIL; break; }
  };
 main () {
   tries = 1;
   /* tries 10 times service setup */
   while((ServiceStartTest.verdict) && tries =< 10)
   {
     if (NOT ServiceStartTest.exec())
     { verdict = FAIL; break; }
     tries = tries + 1;
   }
   verdict = PASS;
  }
}
```

Figure 8 The Example TSSL Script

to distributed testing and the testing notation TTCN was analysed. The Test Session Manager TTman, which is based on TSP1, has been described.

Testing might be a time consuming process. In distributed testing, the optimization of testing time is especially desired as connections have to be maintained for the synchronization of the distributed test components and test personal have to operate at the remote site. Therefore, a careful selection of test cases that have to be executed has to be done. As no standardized notation for specifying the dynamic test case selection exists, a test session specification language TSSL was presented.

The usage and application of TTman and TSSL are demonstrated for an exemplarily test for a TINA Access Session implementation.The use of test setup and synchronization features of TTman to the conformance tests of the TINA Access Session eased the test execution. TTman's features are of particular importance for repeated test executions as in e.g. regression tests. Due to the small number of test cases for the Access Session, the performance gain in test execution from the TSSL script was not that high, but the script helped to control the test execution. The test suite as well as the scripts will be further extended in order to investigate the applicability, usefulness and efficiency of scripting techniques in distributed test executions. The translation of a TSSL script into a target language has still to be performed by hand. Automated support for the use of TSSL is proposed. Incorporating a TSSL interpreter into the presented test session manager TTman will increase the usability of TSSL.

Notes

1. The fact that the coordination message exchange may cross the boundaries of a single tester requires internetworking between the single testers. Reliability of the internetwork is assumed.

References

[1] 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.

[2] Farley, P.; Hogg, S.; Kristiansen, L. et al.: *Ret Reference Point Specifications*, Snapshot 1, Version 0.4, TINA-Consortium, May 14, 1997.

[3] Project "TINA Platform" by GMD FOKUS/Deutsche Telekom Berkom, http://www.fokus.gmd.de/research/cc/platin/projects/, 1998.

[4] TINA-C: Service Architecture Version 4.0, Oct. 1996.

[5] Eurescom Project 412: *Methodology and Tools For ISDN Network Integration and Traffic Route Testing*, Deliverable 3, Test Specifications for ISDN Network Integration Testing, EURESCOM, Heidelberg, August 1996.

[6] *Global Positioning System Standard Positioning Service Signal Specification*, Second Edition, U.S. Department of Defense at the U.S. Coast Guard Navigation Center, Alexandria, VA, June 1995.

[7] Hetzel, P.: *Zeitinformation und Normalfrequenz von der PTB › Über den Telekom-Langwellensender DCF77*, telekom praxis, Heft 1, 1993, pp. 25-36.

[8] ISO/IEC 9646: *Information technology - Open systems interconnection - Conformance testing methodology and framework*, International Standard, Geneva,1991.

[9] 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.

[11] RFC1305 (Mills, D. L.): *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, DARPA Network, University of Delaware, March 1992.

[12] Ousterhout, J.K.: *Tcl and the Tk Toolkit*, Addison-Wesley, April 1994.

[13] ITU-T Z.100: *CCITT specification and description language (SDL)*, ITU-T Recommendation, Geneva, March 1993.

[14] ITU-T Z.120: *Message Sequence Chart (MSC)*, ITU-T Recommendation, Geneva, 1996.

[15] Born, M.; Hoffmann, A.; Winkler, M.; Schieferdecker, I.; Vassiliou-Gioles, Th.: *Performance Testing of a TINA Platform.* - Accepted to Appear in TINA'99, Hawai, May 1999.