Performance Testing of a TINA Platform

Marc Born, Andreas Hoffmann,
Ina Schieferdecker, Theofanis Vassiliou-Gioles, Mario Winkler

GMD FOKUS Kaiserin-Augusta-Allee 31, D-10589 Berlin
Tel. +49 30 3463 7385, Fax. +49 30 3463 8385
email: {born | a.hoffmann | schieferdecker | vassiliou | winkler} @fokus.gmd.de

Abstract - **The TINA architecture specifications are well-known and available for many years. Experiences have been made with prototypical implementations of the TINA architecture or parts of it. These implementations have shown the benefits of the generic concept of TINA for surrounding distributed telecommunication services. Most of the implementations are using new object-oriented implementation languages like C++ or Java and communication architectures like CORBA. However, up to now the question of the performance and satiability of TINA conform implementations, i.e. there eligibility for real applications with thousands of simultaneously acting users still remains.**

This paper reports performance test results made with the TINA platform implementation at GMD FOKUS, which is based on C++, CORBA and Windows NT. The tests concentrate on the access session part of the service architecture (including subscription).

The paper discusses basic concepts of a flexible performance test architecture for distributed systems based on CORBA technology. The concrete performance test results are given, analysed and evaluated.

I. 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 telecommunication 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).

In recent years in many labs telecommunication platforms based on the principles of the TINA-architecture have been developed. However, up to now only a few of them have left the labs. One reason is that there is a lot of scepticism to object technology and especially to distributed systems. Often it is argued that distributed object-oriented systems in general and especially TINA-based systems are less performant and scalable than conventional systems. Hence, an evaluation of real TINA implementations is needed to show that current TINA systems are ready to come to practise.

This paper shows that telecommunication platforms based on TINA technology are scalable and performant enough to meet the requirements of today's telecommunication market. It 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 in our lab. To enable distributed performance testing a flexible test architecture has been implemented based on Common Object Request Broker Architecture (CORBA) technology. 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, scalability and robustness of the TINA access session server can be tested.

It should be noted that distributed object technology also allows that several TINA servers run simultaneously on different network nodes in order to increase the performance and to minimize the response time for the user. Performance testing of simultaneously running TINA servers is a subject to future testing and not covered in this paper.

A. TINA Platform Under Test

This section is to describe the TINA 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.

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

0-7803-5785-X/$10.00 © 1998 IEEE

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.

2. Environment

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.

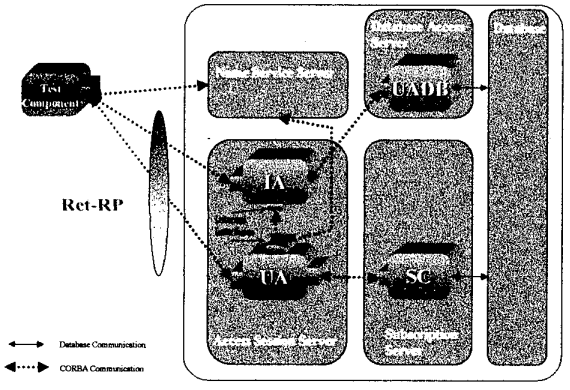

Fig. 1. Configuration of the TINA Platform under Test

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 1 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.

II. Test Objectives

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.

The conformance tests for the TINA platform under test have been made in the past and are not part of this paper. These tests have shown that the TINA platform to be tested conforms to the TINA retailer reference point. The general approach for distributed conformance testing and an example taken from the TINA access session can be found in [6]. This paper concentrates on performance testing in the second step. The goal is to check:

- the performance and partially the robustness and scalability of the TINA access session,
- QoS issues like response time of the TINA access session server to the user.

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.

1. Figure 2 depicts the test behaviour as a Message Sequence Chart (MSC) diagram in parallel to the following description:
2. 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).
3. 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).
4. The TC calls the operation *setUserContext* at the *i_RetailerNamedAccess* interface.
5. In order to retrieve the available services for that anonymous user the TC calls the operation *listSubscribedServices* at the *i_RetailerNamedAccess*

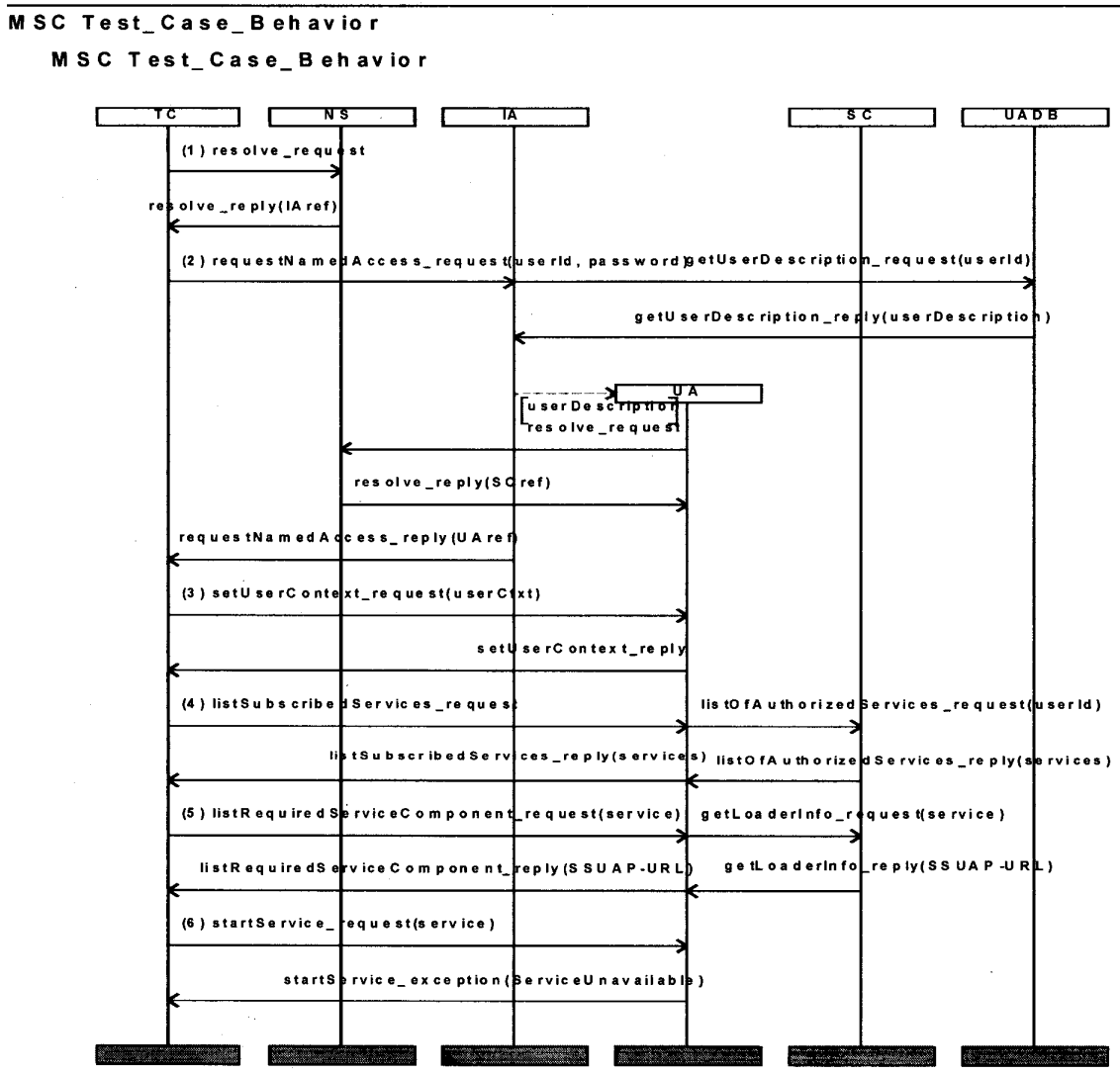

Fig. 2. Behavior of the PTC Emulating Client

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.

6. 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.
7. 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.

III. Performance Testing of Distributed Systems

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.

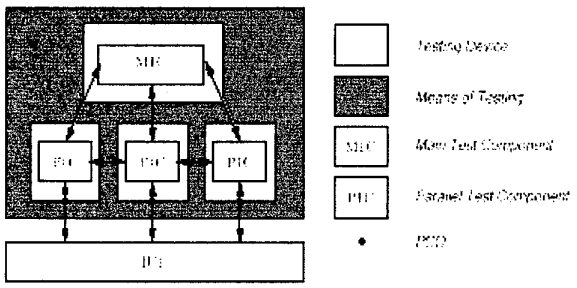

Fig. 3. Generic Distributed Test Architecture

Figure 3 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 only[1].

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.

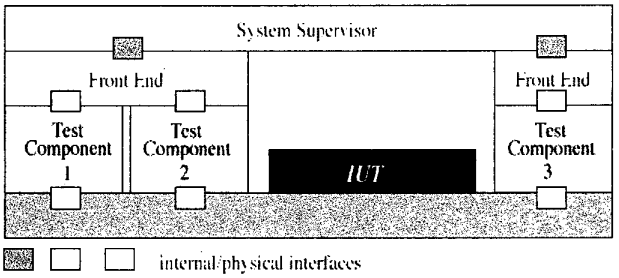

Fig. 4. Architecture of TSP1

At any stage during the test execution it has to be assured that all test components are in a known and stable state.

Basic Concepts of the Test Synchronization Protocol TSP1 by ETSI [5] have been used to implement the main and parallel test components for the performance tests of the service access session. The Architecture of TSP1 is presented in Figure 3.

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 underlying 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.

[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.

IV. The Concrete Performance Test Architecture and Configuration

The concrete performance test architecture for the performance test of the access session consists of the following computational objects which interact via well-defined interfaces (see Figure5):

- TestManager - this component establishes the required configurations and passes the necessary information (like the name service IOR);
- TCAgent - this component runs on every node where test components are to be installed, it acts as a daemon to start the test components on behalf of the Test Manager;
- MTC (Main Test Component) - this component manages the specific test. It allows to set the test parameters, select the test case to be executed, initiates the test and collects the results;
- PTC (Parallel Test Component) - this component executes the test itself.

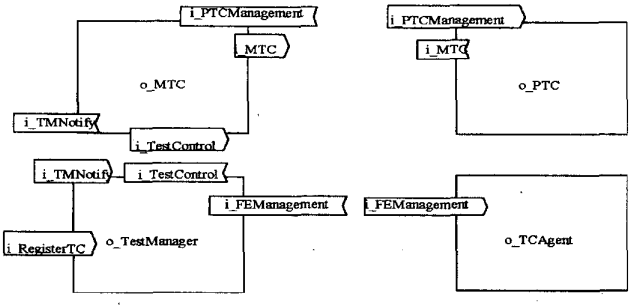

Fig. 5. The Test Components

The MTC and PTC have to be implemented for each specific test whereas the Test Manager and the TCAgent are generic components which are test independent. The implementation of a PTC is very easy, just one virtual method has to be overloaded and implemented with the test behaviour. All other behaviour is implemented in a base class.

Thus, the scenario for implementing and executing a test is the following:

1. Implementation of the PTC by overloading the test_execute method of the PTC class.
2. Implementation of the MTC.
3. Instantiation of the TCAgent on each node on which PTCs or the MTC should run.
4. Instantiation of the Test Manager. The Test Manager has an integrated editor which allows to specify a script containing the test configuration, i.e. what test components are to be started on which node. (Example line: „start 10 ptc.exe on node_2"). This script is then interpreted by the Test Manager.
5. The Test Manager contacts the TCAgents to start the test components. The MTC has to be the first started component. After being started the MTC registers itself at the Test Manager. A reference to the MTC is then being passed to all afterwards installed PTCs.
6. The PTCs register themselves at the MTC.
7. The MTC initiates the tests by calling an operation on each PTC.
8. The PTCs start the test. That means, the overloaded procedure containing the test behaviour is executed.
9. The test results from the different PTCs is transmitted to the MTC and can be evaluated afterwards.

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 TCAgent and the specific PTCs (also called test client) are available for both Windows NT and Solaris systems. One test client for Windows NT has 416KB size, one test client has 5,3MB size.

The performance tests are ongoing work. First results are available and are reported in the paper. For the PTCs we use a pool of 5 Sun Sparc 10, 20 and Ultra 1 workstations with at least 64 MB memory. These machines are connected via fast ethernet with the PC, where the TINA components of the system under test are running and via ethernet with the laptop with the Test Manager and the MTC. Figure 5 shows the concrete test configuration. It should be noted that the network connectivity between the PTCs and the tested system is not below 100 MB/s.

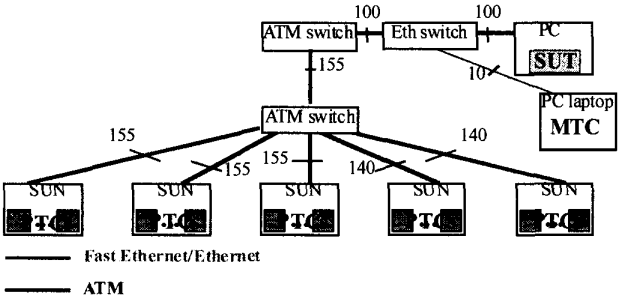

Fig. 6. Performance Test Setup

This configuration ensures, that the PTCs are able to access the TINA system almost simultaneously and that the performance metering is not influenced by the network capacity.

Our specific MTC implementation enables that the PTCs start the test at a specific time. This is done by passing a time parameter from the MTC to all PTCs. The advantage is, that the establishment of the configuration does not overlap with the test itself.

An additional feature of our MTC is, that it allows to specify an interval to start the test components not

simultaneously but with a well-defined gap in between each of the test components.

V. Test Results and Evaluation

Test results have been generated so far for two cases:

(A) local testing only, i.e. both the test system and the system under test are executed on the same computer

(B) distributed testing including the PC laptop for the system under test and the main test components and up to two SUN workstations for up to 40 PTCs.

In both configurations, a single test (i.e. from 'resolve_request' to the final 'startService_exception') takes approximately 1200 ms. In the case, that several test clients are started simultaneously, the response time is degraded.

The test results of configuration (A) are presented in Figure 8. In the local test, best results are achieved with parallel test clients, which are gaped with 1000ms. Then, the response time is in between 800 and 950ms. This holds for 5 up to 40 test clients, i.e. the number of clients is not the restricting size but rather the computational power of the PC laptop. The smaller the gap between the test clients, the longer are the response times. In the case of 40 test clients, the longest response time has been measured.

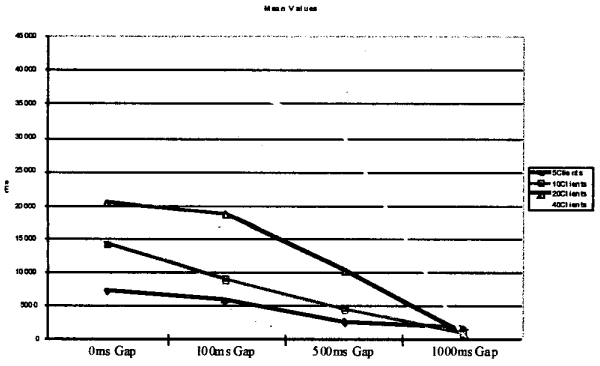

Fig. 7. Mean Values for Response Times for Configuration (A)

In the networked case, i.e. configuration (B) in Figure 7, the measurements show a strong dependence on the number of clients. These test results are rather independent of the gap between individual test clients. It has to be analysed whether the transmission delay of the network supersedes the differences, which are observed in the local configuration. In particular, since the test results in Figure 7 have been obtained within the FOKUS infrastructure network during normal working time.

Additional analysis for the case of simultaneous access of several clients to the access session server has shown, that the underlying data base of the TINA access session is a performance bottleneck. Also, the handling of multiple threads in the access session can be improved. A performance increase is also expected with a more powerful PC (in terms of speed and main memory), on which the access session is executed.

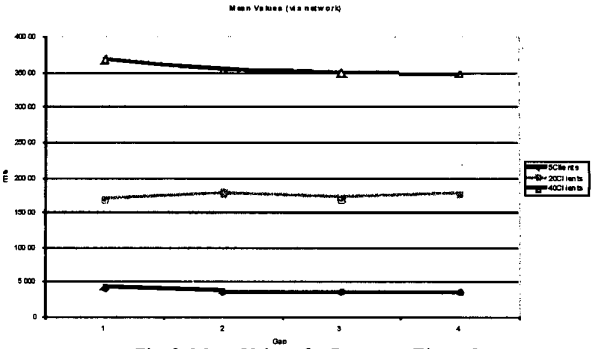

Fig. 8. Mean Values for Response Times for Configuration (B)

VI. 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 TINA technology (and of distributed object technologies in general).

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.

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. Also in this scenario, performance testing can be used to fine-tune the load balancing and migration algorithms but also to evaluate their efficiency.

References

[1] Fischer J., Fischbeck N., Born M., Hoffmann A., Winkler M.: Towards a behavioral Description of ODL, TINA'97 conference

[2] M. Heinrich. Ressourcen-orientierte Modellierung als Basis des Konfigurierens modularer Technischer Systeme, Beitrag zum 5. Workshop Planen und Konfigurieren. Daimler Benz AG. Hamburg, 1991

[3] 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

[4] OMG: CORBA Component Model RFP, orbos/97-06-12, 1997

[5] ITU-T Rec. X.903 | ISO/IEC 10746-3: 1995, Open Distributed Processing - Reference Model Part 3

[6] ITU-T Rec. X.904 | ISO/IEC 10746-4: 1995, Open Distributed Processing - Reference Model Part 4

[7] OMG: The Common Object Request Broker Architecture and Specification, Version 2.1. Aug. 1997.