CONFORMANCE TESTING OF CORBA SERVICES USING TREE AND TABULAR COMBINED NOTATION

ALEXEY MEDNONOGOV[1], HANNU H. KARI[1],
OLLI MARTIKAINEN[1], JARI MALINEN[2]

[1] *Telecommunications Software and Multimedia Laboratory*

[2] *Laboratory of Information Processing Science*

Helsinki University of Technology, P.O. Box 5400 , FIN-02015 Helsinki, FINLAND
Email: mednonog@lut.fi, hhk@cs.hut.fi, Olli.Martikainen@hut.fi, jtm@cs.hut.fi

Abstract This paper presents a formal approach to conformance testing of CORBA-based distributed services using TTCN framework. It discusses mapping of CORBA IDL to TTCN, concentrating on the obstacles and the design issues to be considered. The paper overviews the architecture of the CORBA/TTCN gateway, which acts as an intermediary between test environment and system under test (SUT). It goes through an example of test session and distinguishes the typical stages of dynamic behaviour. The results of the study indicate that TTCN framework especially facilitates testing of active service components, although conformance testing of reactive parts is also possible.

Keywords: CORBA, TTCN, conformance testing, distributed systems

1. INTRODUCTION

The introduction of Internet based services suggests a major change in the future telecommunications service provisioning. The amount of Internet traffic will outweigh the amount of traditional telephone traffic, and new services and service architectures will evolve. The transition from the existing telecommunications services to Internet based ones will change the service infrastructure as well as service creation and management structures [9]. New solutions for service creation, testing and management will be a key success factor for distributed Internet based service provisioning. The distribution of services has been particularly fuelled by the appearance of Common Object Request Broker Architecture (CORBA) in 1991, a technology that addresses the need for interoperable, object-oriented, robust and high-performance distributed solutions in the era of global networks and communications [4].

Surprisingly, the issue of conformance testing of CORBA-based distributed services remains relatively underdeveloped. To our knowledge, application of TTCN to testing CORBA interfaces has been addressed only in few papers (among them, [1] considers testing of TINA service components and [2] describes application of TTCN in the context of test case derivation from SDL models created within a framework of TOSCA project). At the same time, TTCN as a part of ISO-9646 standard [3] is a mature framework, widely recognized for its flexibility and industrial strength. Covering this gap, we share our experiences in testing CORBA objects described in terms of IDL interfaces using TTCN language. We limit ourselves to conformance testing of application-level CORBA servers and clients without considering other issues like testing of GIOP/IIOP protocols, performance testing, interoperability testing, and so on. For the purposes of the present study we assume that all other components of ORB are already tested and are functioning properly. The paper is organized as follows: In section 2, we present an overview of OMG Object Management Architecture and CORBA middleware architecture. In section 3, we define the mapping of CORBA Interface Definition Language (IDL) to TTCN, focusing on the obstacles of such conversion. In section 4, an overall architecture of the CORBA/TTCN gateway is presented and its capabilities are explained. Section 5 goes through a test session example and provides recommendations on test suite coverage. Finally, in section 6 our conclusions are presented.

2. CORBA MIDDLEWARE ARCHITECTURE

The timely creation of interoperable, distributed and high-performance applications capable of communicating transparently in the heterogeneous network environment has grown to be an increasingly challenging task in the telecommunications industry. To alleviate the problem, Object Management Group (OMG), a consortium consisting of several hundreds of member organizations world-wide, has devised a CORBA middleware architecture, the first version of which has been adopted in 1991. CORBA seamlessly interconnects multi-vendor applications and services and let them communicate with each other irrespectively of their location in the network, the operating system they use or the programming language utilized at the implementation stage. CORBA is the communications heart of the Object Management Architecture (OMA) which acts as a higher level reference model (RM) for CORBA framework.

OMA RM classifies all system components into four categories as shown in Figure 1, namely: the Object Request Broker (ORB); CORBA Services (COS); CORBA Facilities; Application Objects. The ORB provides generic low-level communication services to all other OMA components and is responsible for transparent distribution and communication of objects in the network. CORBA Services define a set of system-level interfaces complementing the basic functionality of ORB, and CORBA Facilities (classified into horizontal and vertical ones) provide standard frameworks responsible for defining the universal rules of engagement for collaborating objects [5].

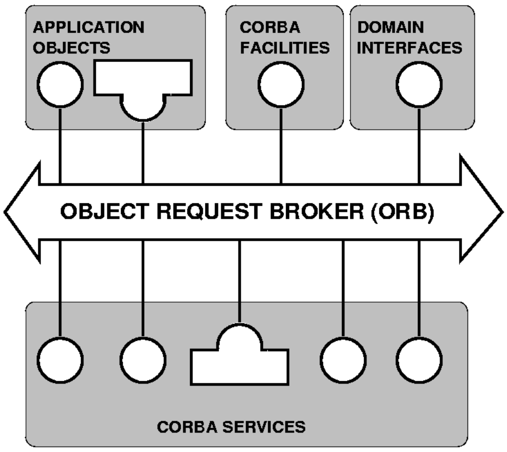

Figure 1. OMA Reference Model

CORBA specification constitutes a set of well-defined rules describing how OMA-oriented architecture shall be actually constructed. Its core part discusses the syntax and semantics of Interface Definition Language (IDL). Object declaration in terms of IDL acts as a contract between a concrete object implementation (server) and its potential clients which is independent from the programming language, platform and location of the contracted parties. IDL declaration provides the formal means for declaring the structure of CORBA object, abstracting from the implementation details. In particular, it defines object location within hierarchy of IDL modules and interfaces, operations accepted by the object, exceptions it raises and the data types of input and output parameters used in operation calls. By putting an IDL declaration into Interface Repository (IFR), the server advertises its capabilities to all interested clients which can introspect the contents of IFR and find out how invocation requests are handled by a specific server, thus turning CORBA into a self-described component architecture. Combined with Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI), Interface Repository provides powerful and flexible means for dynamic invocation of server operations or dynamic processing of client requests. In case a distributed application or service platform does not need this flexibility, invocation requests can be processed directly using precompiled static IDL stubs and skeletons that act as a wrapper for low-level core part of the invocation request. CORBA also standardizes the way objects establish a communication path with each other by associating each server with an Interoperable Object Reference (IOR). As soon as the client finds out the exact contents of IOR, it may freely start communicating with the corresponding server. The overall architecture of CORBA ORB is shown in Figure 2.

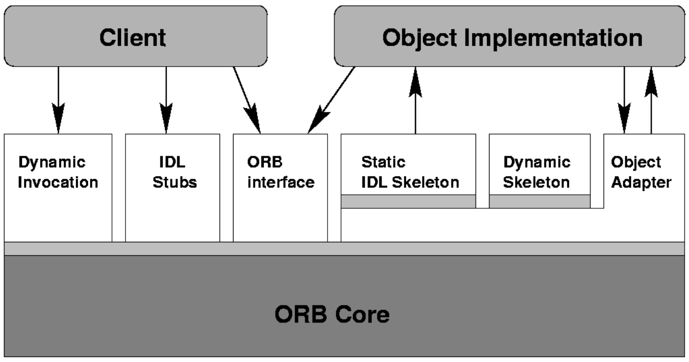

Figure 2. Common Object Request Broker Architecture

In TTCN-oriented conformance testing of CORBA clients and servers the power of DII, DSI and Interface Repository introspective capabilities is fully employed. Basically, TTCN ASP is converted to CORBA dynamic request or to dynamic response and vice versa, and appropriate generic clients and servants are created on the test environment side for that purpose. According to CORBA specification, a System Under Test (which is viewed as a collection of client and server objects) cannot distinguish whether a static or dynamic invocation scheme has been used, so the implementation details of the test environment are completely transparent from the SUT perspective.

3. MAPPING OF CORBA IDL TO TTCN

The rules for mapping CORBA Interface Definition Language to TTCN have been presented in the paper of GMD FOKUS [1], although the related work of X/Open and Network Management Forum (NMF) conducted by the JIDM (Joint Inter Domain Management) working group shall be acknowledged, too [8]. However, JIDM mainly concentrates on the reverse task of mapping GDMO/ASN.1 to IDL, so JIDM mapping rules have an indirect impact in the present paper; on the other hand, the mapping discussed in [1] has contributed most to our study. In particular, [1] defines the rules for mapping IDL operations, exceptions, constants, attributes and type definitions to TTCN, and we adhered to the same concepts as long as it has been practical. Yet, amendments and supplements to the approach discussed in [1] proved to be inevitable, either in cases when level of details in the original paper was insufficient or when alternative design of the mapping rules has been viewed as beneficial for improving the flexibility of the whole testing scheme. To avoid overlapping with [1], hereafter we provide a brief overview of the basic principles used in conversion of IDL to TTCN (mostly derived from [1]) and then concentrate on our findings in this area, as well as on the obstacles and the design issues to be considered.

The conversion rules are as follows: (a) one CORBA instance of IDL interface maps to one PCO declaration, which may be reused in different test cases to represent several CORBA objects; (b) declaration of one operation maps to a pair of ASPs, namely Call ASP and Reply ASP, the identifiers of which are prefixed by pCALL_ and pREPLY_ respectively; (c) one actual operation call maps to a pair of ASP constraints; (d) IDL data types are mapped to ASN.1 types; (e) IDL specification itself maps IDL attributes to normal operations, so additional arrangements are unnecessary; (f) one Exception ASP (identified as pRAISE) is defined for all possible CORBA exceptions, both system exceptions and user-defined, so that different constraints may be introduced as necessary for individual exceptions or for groups of them; (g) to facilitate debugging of an Abstract Test Suite (ATS) at design stage, CORBA/TTCN gateway introduces its own gateway-specific exceptions defined in terms of IDL to signal e.g. incorrectly constructed ASP and any other exceptional situations not directly related to SUT, and a separate namespace is reserved by the gateway for that purpose; (h) IDL-to-TTCN mapping implies an effective solution for IDL name resolution and inheritance mechanism without declaring that specifically, as both issues are more relevant to concrete language-dependent implementations of CORBA objects.

The mapping rules defined in our gateway specification ensure that every distinct entity of IDL language (interface, operation, type definition, identifier of enumerated type etc.) possesses a one-to-one mapping in TTCN namespace of identifiers. For that purpose, we use a symbolic chain "_i" as a separator and duplicate all occurrences of underscore characters ("_") in names of original IDL entities while mapping them to TTCN. For example, interface "IntC" defined within IDL module with scoped name "::ModA::ModB" according to these rules may be mapped to PCO identifier "PCO1_iModA_iModB_iIntC", and oneway operation "opD" defined within interface "::ModA_::_ModB::IntC" is mapped to Call ASP identifier "pCALL_iModA___i__ModB_iIntC_iopD". Apparently, we have to concatenate several IDL identifiers to form one TTCN identifier so that initial IDL entities can be reconstructed from the resulting TTCN aggregate, otherwise the conversion rules would not guarantee a one-to-one mapping, which can lead to possible collisions in TTCN namespace. To address this issue, we use a concept similar to "bit stuffing", a well-known pattern utilized in low-level telecommunication protocols to mark boundaries of frames sent over the communication medium. In our case, a similar idea is introduced to label margins of module, interface and operation names.

We also introduce an additional field CALL_ID of ASN.1 type INTEGER into every Call ASP, Reply ASP and Exception ASP to uniquely identify operation calls sent to CORBA servers or received from clients, which is necessary for support of concurrent invocations within one test case. It is required that concrete values of CALL_ID shall be unique within a test case, although set of CALL_ID values allocated for servers may overlap with values used by clients. Without having a CALL_ID field, it would be an ambiguous action to invoke the same operation while the previous operation call is still pending, as in this case ATS would be unable to distinguish which operation completed its execution once one of them returns. Moreover, CALL_ID field is of vital importance for determining which operation call has thrown an exception if pRAISE ASP is received or sent.

Our specification of IDL/TTCN mapping intentionally avoids use of TTCN operations, although their use in concrete test suites is not restricted. One situation when TTCN operations could be used is related to associating PCO with a concrete CORBA object, and [1] seemingly uses operations for that purpose. However, this approach does not fit well with good TTCN design practice, as TTCN operations shall be normally limited to calculating the return value from the set of input parameters, and use of operation side effects for performing an association procedure does not strictly follows the formal part of ISO 9646-3 standard. Instead, we most naturally (and more formally) delegate the responsibility for associating PCO with a CORBA object to the CORBA/TTCN gateway by sending to it an ASP of a special format as needed.

For that purpose, we define several forms of Registration ASP which is sent to the gateway every time ATS is willing to bind PCO with a client or server object. For instance, ASP identified as pSREG_IOR sent through PCO in question will bind this PCO to the CORBA server advertised by IOR (Interoperable Object Reference); another form of Registration ASP identified as pCREG_NSERV is used to instruct the gateway to create a generic servant on its side and advertise its presence to all interested clients by putting this servant into object hierarchy in the Naming Service. Since then, all operation calls received from clients will be relayed to the PCO through which original pCREG_NSERV ASP has been issued. Gateway will inform ATS of whether the binding procedure failed or succeeded by sending a gateway-specific Exception ASP to the test environment. Gateway specification requires that all bindings mentioned above must be removed (and generic servants destroyed) without any special notification coming from ATS after test case completes its execution.

Our mapping of CORBA exceptions to TTCN allows catching group of exceptions in one TTCN constraint. As noted previously, we define one Exception ASP for all possible CORBA exceptions. This ASP contains three fields: (1) CALL_ID of the operation that has thrown an exception; (2) absolute scoped name of an exception thrown, as defined in IDL interface; (3) exception body encapsulated into separate PDU. Absolute scoped name is encapsulated into ASN.1 SEQUENCE OF IA5String, so that exception having scoped name "::ModA::IntB::ExcC" may be constrained by construct {"ModA", "IntB", "ExcC"}. At the same time, by defining construct {"ModA", "IntB", ?} we are capable of catching a group of exceptions defined within interface "::ModA::IntB". We assume that CORBA system exceptions are defined within interface "::CORBA::SystemException" and gateway-specific exceptions are defined within namespace "::GatewayException". The gateway-specific exceptions usually signal a receipt of structurally invalid ASP; inopportune ASPs may in some cases cause a gateway exception too, for instance when Call ASP has been issued before sending Registration ASP. However, the gateway will handle most of inopportune ASPs of all other kinds, as well as structurally correct ASPs containing invalid values.

It was already mentioned that IDL data types map almost naturally to ASN.1 types, as shown in Figure 3. Yet, there are two essential exceptions from this rule. One of them regards to IDL discriminated union. A straightforward solution implies mapping of IDL union directly to ASN.1 CHOICE. However, a direct conversion may lead to a loss of information about the exact value of the discriminator, since several values of the discriminator may refer to one branch of IDL union. On the other hand, CORBA language mappings allow a direct access to the discriminator, hence the result of such access is implementation-dependent. To eliminate this ambiguity, the final mapping for IDL union proposes encapsulation of discriminator field and ASN.1 CHOICE field into one ASN.1 SEQUENCE structure, where a field of type CHOICE contains the actual branch activated by union and an additional field carries the exact value of the discriminator.

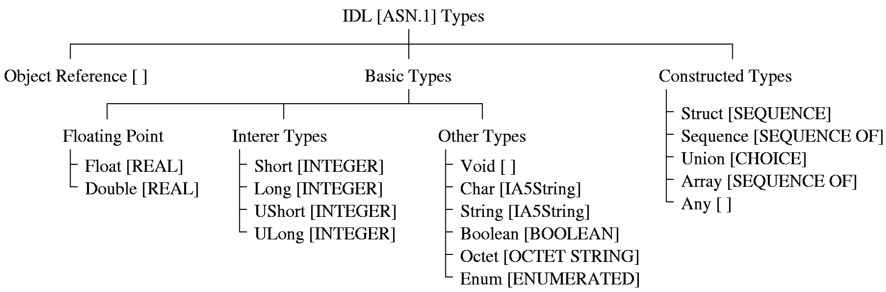

Figure 3. Mapping of IDL Data Types to ASN.1

More serious restrictions concern mapping of IDL Any type. Mapping of IDL to TTCN does not imply a one-to-one conversion from IDL data types to ASN.1 types, that is one ASN.1 type may correspond to several IDL types. For instance, ASN.1 INTEGER may correspond to IDL "unsigned short" or "long" or "unsigned long", and ASN.1 IA5String may correspond to IDL "char" or IDL "string". For this reason, CORBA/TTCN gateway internally uses Interface Repository (IFR) as its structural backbone, so every time it receives Call ASP going to server or Reply ASP or Exception ASP going to client, it performs a lookup of IFR to clarify how the structured data of the operation call must be actually processed. However, if the gateway meets the definition of Any type in IFR, it may not have any hint how to process the respective part of the received ASP, as the structural information extracted from the ASP is insufficient to provide a reliable guidance to the gateway, due to one-to-many nature of TTCN-to-IDL mapping of data types. If "any" value has been received from SUT, then its conversion to ASP is trivial, but in this case ATS shall contain multiple ASP definitions per one operation, what contradicts to the mapping rules described above. Due to all these complexities, current version of the gateway does not support mapping for Any type. Three possible solutions are anticipated, all three having their own advantages and drawbacks:

(1) Type information is explicitly inserted into corresponding identifiers of ASP fields in form of prefixes, e.g. "s" for strings, "c" for characters etc. However, this approach requires defining several pairs of ASPs per one operation call if the latter happens to contain "any". Moreover, since then ASP identifier itself must obey additional rules, because gateway must know exactly how ASP name is constructed at run-time, and if there are several possible ASPs per response from the same operation, a kind of operation key is definitely needed. At the same time, we cannot encapsulate a variable of Any type into a separate PDU and then append prefixes only to what is inside (what would have eliminated the need for more than one pair of ASPs per operation), since ISO 9646-3 standard prohibits use of non-ASN.1 types inside ASN.1 data. Finally, it remains unclear how the whole scheme would address cases like "union" inside "any" or other challenges, so this approach shall be examined with caution.

(2) Values of "any" type are mapped to ASN.1 OCTET STRING type, having part of the marshalling buffer as a content filler. Although this approach seems to be the most natural solution from the gateway perspective, it puts a heavy burden on the designer of test suite who is since then responsible for manual construction of part of GIOP/ESIOP/IIOP stream, which might be practically unacceptable.

(3) The set of IDL interfaces describing SUT is supplemented with manually implemented decorators that do not contain "any", so that operation calls are routed through them. Decorator is a well-known design pattern, the main function of which is attaching additional responsibilities to the core system [7]. In our case, if we assume we know in advance all variations of typed information encapsulated into "any" variables, we can then make an equivalence transform of the operation containing "any" into several operations containing only conventional types. A set of these "wrapper" operations may be defined within one decorator object acting as a relay point between MOT and core SUT. This idea could be extended to mapping of one server operation containing "any" into two sets of decorating oneway operations, one set responsible for decorating a direct part of operation call and another one responsible for decorating operation return. This would help if operation call and operation return both contain "any"

parameters. The similar idea could be also applied to operation calls coming from clients. The technique of decorators is a good trade-off between two previous approaches, although it requires additional effort to implement decorating CORBA objects.

4. ARCHITECTURE OF CORBA/TTCN GATEWAY

The CORBA/TTCN gateway acts as an intermediary between the test environment and SUT. The gateway itself is a CORBA-based Java application, enjoying all the benefits of both CORBA distributed nature and Java platform independence. The ORB of our choice was ORBacus for Java from Object Oriented Concepts Inc, which fully implements the core functionality of CORBA 2.0 in accordance to the OMG standard [11]. CORBA mapping to Java is the most natural and frequently used solution, as Java takes care of transparent object removal and has a comprehensive exception handling mechanism. Although Java is an interpreted language and is relatively slow, this issue is not of crucial importance as long as performance measurements in conformance testing are not usually involved. Java is not perfectly suitable for testing low-level protocols as Java API does not provide a direct access to protocol layers lower than TCP and UDP, but this could be solved by implementing a gateway logic on Java and a system-level part on C++. The two parts could be, again, combined using CORBA or Java native calls. Nevertheless, in case of testing CORBA services it is sufficient for the gateway to have indirect access to the protocol stack.

The gateway communicates with the test environment using CORBA ORB, thus making it possible to have both subsystems located on different machines. This architectural design was technically feasible, as we have used OpenTTCN from Open Environment Software Oy as our target test environment which internally uses ORB as its communication backbone [10]. The distributed nature of the means of testing (MOT) may significantly facilitate testing, especially if CORBA services are tested in conjunction with some other telecommunication software, what may require the physical distribution of multi-purpose gateways in the network. The interpreted nature of the test environment combined with a simple and developed-friendly API has noticeably reduced the time and budget spent on the gateway design, allowing the design process to be iterative, i.e. when the gateway components are designed and their functionality is tested immediately. The absence of the overhead related to compiling ATS into ETS was another speed-up factor at the design stage.

The overall structure of the gateway is shown in Figure 4. The core part of it consists of BuilderAdapter, VisitorAdapter, ClientCallBuilder, ServerReplyBuilder, ServerCallVisitor and ClientReplyVisitor. It performs a conversion from ASP format into format acceptable for making an operation call and vice versa. Here we have used three well-known design patterns, namely Builder, Visitor and Adapter [7]. Visitor performs introspection of ASP coming from the test environment, Builder constructs ASP corresponding either to client operation call or to server operation response and Adapter performs a technical conversion of information obtained from the Interface Repository into internal format acceptable for processing either by Builder or by Visitor.

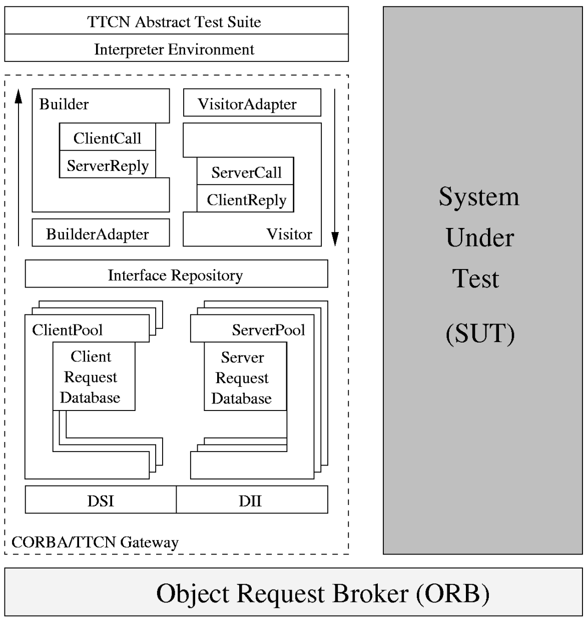

Figure 4. CORBA/TTCN Gateway Architecture

The gateway uses the asynchronous mode of DII invocation mechanism, what naturally correlates with asynchronous nature of TTCN operational semantics. The asynchronous mode enables the gateway to process new invocation requests coming from the test environment while other operation calls are still pending. This is also valid for concurrent invocations of one explicit operation attributed to the same CORBA object. This feature allows design of concurrent test suites with several parallel test components (PTC) or even simultaneous execution of several independent test suites served by one gateway component. In the latter case, the only restriction introduced by the gateway concerns prohibitive use of overlapping PCO identifiers in different test suites, as the TTCN test tool in use currently does not provide means for distinguishing between different test sessions as they are seen from the gateway perspective. Subsystem named "ServerRequestDatabase" is running in a dedicated thread and regularly polls for response from incomplete requests, accepting at the same time registration of new operation invocations issued by ServerPool. Client side of the gateway operates in a similar way, with "ClientRequestDatabase" subsystem designed to notify waiting threads in the ClientPool in case a response on client request has been issued by ATS. Client operation calls can be executed concurrently, too.

5. TEST SUITE EXECUTION

For the purposes of verifying a gateway functionality, two sets of test cases have been developed. Each set contains nine test cases, one set for testing servers and another one for testing clients. This collection constitutes a minimum orthogonal set of tests of the gateway states and can therefore be recommended as a skeleton for developing more detailed test groups aimed at practical testing of ORB implementations. The test cases and their objectives are classified as follows: (1) *"prototype"* tests the very basic functionality of the gateway: it performs (or accepts) the only one operation call with "in" and "out" parameters of primitive IDL types; (2) *"primitive"* tests all IDL primitive data types as input and output parameters; (3) *"oneway"* tests correctness of executing oneway operations; (4) *"struct"* tests all IDL structured types (struct, sequence and array) except union; (5) *"choice"* tests IDL union, including recursive unions (i.e. union inside union); (6) *"context"* tests operations declared with context clause; (7) *"concurrent"* tests multiple invocation requests coming from clients or sent to servers; (8) *"exception"* tests handling of both system and user-defined exceptions; (9) finally, *"abnormal"* tests various invalid situations like incorrectly constructed ASP, most of which shall normally be observed on the ATS side in the form of gateway-specific Exception ASP.

A simplified example of a typical test session is shown in Figure 5. The session dynamic behaviour follows two stages of testing a CORBA service: (a) registering CORBA object, and (b) issuing (or accepting) actual operation calls. The third stage of deregistering CORBA object is implicitly performed by the gateway upon completion of test case and hence need not be reflected in test suite. The Interoperable Object Reference (IOR) used for registering CORBA server object is supplied to ATS in form of a PIXIT parameter.

<table>
<tr><td colspan="6">Test Case</td></tr>
<tr><td colspan="3">Test Case ID:</td><td colspan="3">tcINVOKE_COUNT</td></tr>
<tr><td colspan="3">Test Group Reference:</td><td colspan="3">Operations/</td></tr>
<tr><td colspan="3">Test Case Purpose:</td><td colspan="3">To test that after CORBA server object is registered by its IOR and "increment" operation is invoked, it returns the argument of the operation incremented by one.</td></tr>
<tr><td>N</td><td>L</td><td>Behaviour Description</td><td>Constraint Ref</td><td>V</td><td>C</td></tr>
<tr><td>1</td><td></td><td>PCO1_iArithmetic_iCount ! pSREG_IOR START tSREG</td><td>cSREG_IOR (xIOR1_iArithmetic_iCount)</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>PCO1_iArithmetic_iCount ? pRAISE</td><td>c1RAISE_iGatewayException _iRecoverable_iGeneral</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>PCO1_iArithmetic_iCount ! pCALL_iArithmetic_iCount _iincrement START tREPLY</td><td>c1CALL1_iArithmetic_iCount _iincrement</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td>PCO1_iArithmetic_iCount ? pREPLY_iArithmetic_iCount _iincrement</td><td>c1REPLY1_iArithmetic_iCount _iincrement</td><td>P</td><td></td></tr>
<tr><td>5</td><td></td><td>? TIMEOUT tREPLY</td><td></td><td>F</td><td></td></tr>
<tr><td>6</td><td></td><td>PCO1_iArithmetic_iCount ? OTHERWISE</td><td></td><td>F</td><td></td></tr>
<tr><td>7</td><td></td><td>? TIMEOUT tSREG</td><td></td><td>F</td><td></td></tr>
<tr><td>8</td><td></td><td>PCO1_iArithmetic_iCount ? OTHERWISE</td><td></td><td>F</td><td></td></tr>
</table>

Figure 5. Test Session Dynamic Behaviour Description

It shall be acknowledged that the least formal part of CORBA services practical testing (requiring in some cases a manual control over SUT) relates to establishing a communication path between a CORBA object and the means of testing. In the presented example this is achieved by supplying an IOR of the object, but finding out the exact content of IOR may itself require knowledge about location of IOR file in e.g. file system or in the network. Moreover, an object may be located in the Naming Service or in the Trading Service what makes registration procedure even more complicated. The client side of SUT may be even harder to deal with, as ATS must explicitly instruct the gateway to create a generic servant and advertise its presence to the rest of the CORBA world. If for example SUT client is willing to immediately obtain a reference to the corresponding servant emulated by MOT before it becomes available, then such client shall be subject to manual control of the operator, and a corresponding IMPLICIT SEND statement shall be present in ATS.

6. DISCUSSION

This paper discusses the practical application of TTCN framework to conformance testing of CORBA services. An alternative approach to the same task would require a manual design of CORBA-based subsystem using language for which CORBA mapping specification exists, for instance C++, Java or Smalltalk. This manually implemented subsystem would then perform test invocations of SUT operations and in its turn respond to requests coming from SUT clients. Several issues shall be considered while making a choice between these two alternatives: (1) TTCN is a formal standardized framework specifically designed for reusable and automated testing of telecommunication systems; (2) TTCN facilitates practical testing of active components (servers) of CORBA services, as it clearly outperforms the above mentioned conventional technique due to ad-hoc and informal nature of the latter; (3) Yet, TTCN does not show up enough flexibility in emulating a CORBA servant, as TTCN framework may require implementation of TTCN operations for this purpose, what makes the processing of requests coming from reactive components of SUT (clients) look relatively awkward. To sum up, in case of testing CORBA clients TTCN obviously looses a comparison with its non-TTCN conventional counterpart. However, if SUT mostly contains active parts (which is normally the case in testing CORBA services), then a TTCN framework shall undoubtedly be preferred.

Several aspects constitute the grounds for the further research. First, the use of Modular TTCN for supplementing the existing IDL-to-TTCN mapping rules shall be investigated. The second aspect addresses the need for automating the process of test suite derivation from IDL interfaces and SDL models, as generating ASN.1 type definitions and ASP declarations from IDL is a relatively routine operation and is subject to automation. Finally, the third edition of TTCN, although not officially published yet, promises to be an innovative continuation of the TTCN standard which may bring a fresh breath to conformance testing of CORBA services and distributed applications. All these issues are planned to be addressed in our future work.

REFERENCES

[1] I. Schieferdecker, M. Li, A. Hoffmann, Conformance Testing of TINA Service Components - the TTCN/CORBA Gateway, *Fifth International Conference on Intelligence in Services and Networks IS&N'98*, Antwerp, Belgium, May 25th - 28th, 1998.

[2] R. Sinnott, M. Kolberg, Creating Telecommunication Services based on Object-Oriented Frameworks and SDL, *Proceedings of the Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99)*, Saint Malo, France, 1999.

[3] ISO/IEC 9646-3, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)*, International Standard, second edition, 1994.

[4] Object Management Group, *The Common Object Request Broker: Architecture and Specification, Revision 2.3.1,* OMG Document 99-10-08, October 1999.

[5] R. Orfali, D. Harkey, *Client/Server Programming with Java and CORBA,* John Wiley & Sons, Inc., 1998.

[6] Object Management Group, *IDL-Java Language Mapping,* OMG Document 99-07-53, June 1999.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, 1995.

[8] The Open Group, *JIDM Specification Translation*, Technical Document P509, February 1997.

[9] O. Martikainen, J. Karvo, Internet Based Service Development, *Fifth International Conference on Intelligence in Networks (Smartnet '99)*, Thailand, November 22nd-26th, 1999.

[10] Open Environment Software, *OpenTTCN Tester,* http://www.oes.fi.

[11] Object Oriented Concepts, *ORBacus Manual,* http://www.ooc.com.