Experience report on Conformance Tests for CORBA ORBs

M. Li, A. Rennoch, I. Schieferdecker,
D. Witaszek, O. Halabi, A. Vouffo, and A. Yin

FOKUS, Fraunhofer Institute for Open Communication Systems

Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
phone: +49 30 3463-7000, fax: +49 30 3463-8000,
email: corval2@fokus.fhg.de
www.fokus.fhg.de/tip

Abstract

Middleware platforms are in wide spread use for distributed systems. Their quality is key to the stability and interoperability in multi-vendor heterogeneous environments. It is the aim of the EC IST project CORVAL2 to enhance the techniques used to validate the conformance of OMG's CORBA technology. This paper investigates testability aspects of CORBA ORBs and considers CORBA based systems both from a theoretical and practical view on testing. Test strategies are proposed and a conformance test suite presented.

Keywords: CORBA, Conformance testing, Testability, Static and dynamic analysis, Testing tools.

1 Introduction

Distributed software applications cover a broad range of user domains: e-commerce and co-operative working are only some catchwords. It is middleware, which enables applications to be independent from different operating systems, languages, data communication, databases, I/O interfaces or heterogeneous computer hardware. Obviously, the quality of the programming platform is a key issue in software development. An increasing number of vendors of middleware together with a huge number of applications arise the serious question on interoperability. In order to avoid a blind one-to-one interoperability testing of each pairs of server/customer configuration, it is proposed to follow the methodological approach of conformance testing. Conformance to standard specifications ensures interoperability and portability between products of different vendors, what is essential for the openness of distributed systems.

A well established programming platform is the Common Object Request Broker Architecture (CORBA) standardized by the Object Management Group (OMG) software consortium. CORBA allows software components to communicate with each other independent from their location and their details of implementation such as programming language and operating system. The communication between software components follows a client/server paradigm. The interaction between clients and servers is based on the Object Request Broker (ORB).

In this paper we discuss the technical approach to conformance testing of CORBA v2.3 ORBs [19]. The specification consists of the CORBA Core, CORBA Interoperability and CORBA Interworking[1] parts. The CORBA Core defines the Interface Definition Language (IDL) and the Application Programming Interface (API) used by CORBA applications. IDL allows for the description of IT services and applications in a language and implementation-independent way. Different programming language mappings for IDL are provided. Conformance to the IDL and API in a selected programming language ensures the portability of an application on different CORBA implementations. In

1. CORBA Interworking is intended for the communication between CORBA and Microsoft's COM systems and is not considered in this paper.

0-7695-1287-9/01 $17.00 © 2001 IEEE

addition to the portability, the CORBA Interoperability defines protocols to support the interoperability between CORBA ORBs (Object Request Brokers). Mandatory for a CORBA conformant ORB is the combination of the General Inter-ORB Protocol (GIOP) and its specialization with the transport protocol TCP, the Internet Inter-ORB Protocol (IIOP).

The paper considers the testability concept and approaches to test the different requirements of the CORBA core specifications. It is structured as follows: In Section 2 the testing target, i.e. the special nature and requirements for CORBA ORBs, is presented. An implementation oriented view on testability is given in Section 3. This section informs on the selected test implementation strategies, too. A short overview on the developed conformance test suite for CORBA v2.3 ORBs is given in Section 4. Finally, conclusions for CORBA test specifications and test ORB interoperability protocols are drawn.

2 The testing target: CORBA ORBs

The Object Management Group (OMG) was formed to provide an architectural framework together with detailed specifications to "drive the industry towards interoperable, reusable, portable software components based on standard object-oriented interfaces"[19]. The key component of the framework is the Common Object Request Broker Architecture (CORBA). It supports a distribution transparent communication between clients and server objects in a heterogeneous environment. Since its introduction, the architecture and specification of CORBA have been improved by several revisions. The first specification and implementation were available at the beginning of 90's. The Interface Definition Language (IDL) and some basic CORBA interfaces were defined there. The revision 2.0 published in 1995 contains for the first time the interoperability protocols GIOP and IIOP. This was significant for the wide acceptance of CORBA, because GIOP and IIOP ensure interworking between CORBA-based systems supplied by different vendors. In the subsequent revisions 2.1 and 2.2, an interface for dynamic management of Any values (DynAny) and the concept of the Portable Object Adaptor (POA) were added. The POA extends and replaces the Basic Object Adaptor (BOA). POA was further improved in the 2.3 revision that was standardized in 1999. New concepts included in this revision are Value Type and Abstract Interfaces. In addition, the number of supported languages has been increased continuously. Among the already covered languages are C, C++, Java and Smalltalk.

While CORBA is maturing and more and more CORBA implementations and CORBA-based products emerge, the attention on interoperability and portability increases. The key of interoperability and portability is the conformance to standard specifications. CORBA specifications place conformance requirements in terms of *Compliance Points*. One compliance point is, for example, the whole CORBA Core that contains basic interfaces used by CORBA-based applications.

A Means of testing (MOT) is needed to evaluate the interoperability and portability. It must be vendor independent and it should provide reproducible evaluation results. The EC IST project CORVAL2 develops and provides such a MOT. A major input was the CORBA Verification Suite (VSOrb) [23]. VSOrb is an extensive test suite for the C and C++ mappings of the CORBA 2.1 (abbr. as v2.1 in the following) specification. It has been applied to different ORB implementations. VSOrb makes use of the Assertion Definition Language (ADL[22]) Translation System and Test Environment Toolkit (TET[24]) in order to automate the creation and execution of tests. A further input to the v2.3 test development origins from a recent work on testing techniques for CORBA-based systems. It resides on the Conformance Testing Methodology and Framework (CTMF) [17] of ITU-T/ISO. CTMF is widely used in the industry for testing various communication protocols. Its usability for object-oriented systems is shown in [13]. A test management environment supporting CTMF-based test systems is also part of the input. It covers means for test management, including setup and control of tests, as well as test reporting.

The testability of object-oriented client/server systems such as CORBA systems depend on different aspects covering general concepts of distributed object technology, the interface architecture and specification of objects and components, and the specific realization of implementations based on different programming languages.

The analysis has been structured according to the major implementation aspects of a CORBA ORB, i.e. syntax and semantics of IDL definitions, syntax and semantics of the API, and the definition of GIOP/IIOP (see also Figure 1).

The testability analysis of CORBA systems adopts this structuring principles, i.e. the major conformance requirements according to the product profile [20]. It distinguishes the following system aspects

- language mapping
  mapping of interface descriptions (i.e. with an IDL compiler) to programming languages (e.g. C++ and Java) and generated interfaces (i.e. IDL stubs/ skeletons)

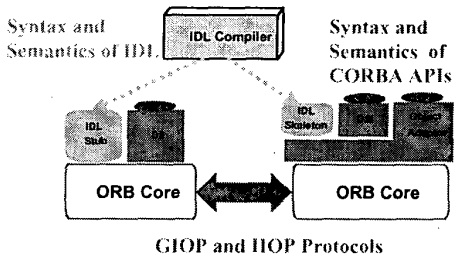


Figure 1: Different system aspects

- operation based interface
  API (e.g. ORB interface, DII, DSI, object adapter, interface repository)[2]
  CORBA services (e.g. name service)
- message based interface
  CORBA interoperability between different ORB core implementations: GIOP/IIOP

In the following we discuss more detailed the individual aspects.

2.1 Language mapping

Tests on language mappings concern in fact the test of compilers (and interpreters). Compiler tests cover both static aspects on the error-free syntax of the generated code and dynamic aspects on the correct behavior of this code. It has to be noted that different compilers may conform to a language mapping specification, by producing different code with equivalent behaviour. A language mapping compiler differs from classical programming language compilers, which translate a programming test to machine code. Testing e.g. an IDL compiler has the following advantages in comparison to the latter one:

- it produces an output which can easily be inspected (files with programming language code)
- the input language specification (IDL) does not contain any behavior (e.g. state transition)

Thus, language mapping tests are controllable (e.g. IDL input) and observable (by the resulting files with programming language code). A core problem is how to analyze the results and how to assign a test verdict, i.e. how to compare the output with the specification of the language mapping, especially when taking into account the various possible mappings for a given language.

Additionally, when testing a code in programming languages, like C++ and Java, the object-oriented paradigm has to be considered. In the context of object-oriented programming, special emphasis is needed for inheritance, polymorphism, late binding, and encapsulation [4].

2.2 Operation based interface

Requirements on the testability of CORBA specifications for operation based interfaces address basic requirements such as existence and completeness of provided operations, but also further desirable specific, testing oriented details:

- the prerequisites for operations and parameters,
- the dependencies between operations under test and other operations/parameters,
- the dependencies between parameters under test and operations/other parameters,
- sample structures proposed for testing of data types,
- sample values or value ranges proposed for testing of parameters, and
- sample parameter combinations proposed for testing for operations under test.

The fault model should consider both, valid and invalid values, structures, operation orders etc. The test approach can follow a typical 'service testing' approach, i.e. a classical request/reply (incl. exceptions) communication at the interface under test. The controllability of tests for operation based interfaces results from the underlying client-server principle, by which operation under tests can be invoked from the test system. The observability is restricted to the direct reactions on the operation invocation (i.e. reply or exceptions). Internals that results from an operation invocation are typically not observable.

2.3 Message based interface

The requirements for testing a message based interface with an underlying protocol specification address five distinct parts to be considered [7]:

- the service that is provided by the protocol,
- the assumption about the environment in which the protocol is executed,
- the vocabulary of messages used to implement the protocol,

2. Note: Figure 1: does not include all operation based interfaces. Further interfaces exist between the client or servant and the ORB core, interface repository etc.

- the format of each message in the vocabulary, and
- the procedure rules guarding the consistency of message exchanges.

Only the direct reactions are observable but not internals that are caused by incoming messages. The fault model of message related system aspects corresponds to classical conformance testing approaches[17], which focuses on the identification of wrong messages, which are caused by coding errors, faulty protocol data or inconsistent behavior with respect to the protocol rules.

3 Test strategies for ORB implementations

As seen from the previous sections the testability of complex systems like CORBA is depending on a series of requirements on the system's model, specification and implementation details. This section discusses testable features for CORBA and proposes appropriate test strategies and approaches.

3.1 Language mapping

OMG-IDL has a formal syntax and not formal but well understood semantics. The target language is a programming language with defined syntax and semantics. The mapping describes how the IDL syntax and semantics can be reflected in the syntax and semantics of a given programming language (considering available concepts in the used version of the programming language). The mapping gives to the ORB vendors (to a certain degree) freedom for the form of the resulting code.

The IDL compiler is testable in the sense that it can be assessed (using different IDL input) and observed (if output in readable form is available). Due to the informal character of the mapping description and the variety of possible mappings, the full automation of the test case generation and result analysis is practically not possible. Therefore semi-automated generation together with manual addition is advisable.

For the development of language mapping tests we performed the following steps to achieve a representative set of test cases:

- define a set of IDL specifications to be tested,
- based on the mapping rules, the features of the target programming code should be specified manually,
- compare the output obtained from the test execution with the expected output, and
- assign a verdict.

Test model

In the context of compiler tests, there is no complete test theory available which solves the language mapping problem for IDL. Nevertheless, there is some general work on compiler tests [3]: language mapping (i.e. compiler) tests are black box tests, since there are no possibilities to inspect a compiler, but only to observe and run the output of the compiler for a given input specification. Following this viewpoint the output of the compiler has to be verified according to the input. With respect to the static aspects, the output of the "compiler under test" can be validated by the programming language compiler of the target language.

An illustration on the general process and the relationship of the involved documents in given in Figure 2. The IUT is an IDL compiler under test. The *Test Sequence Generation* has to produce (on a base of the IDL Grammar) a set of IDL test sequences covering all IDL constructs used in all possible context (or a representative subset of it). An output of the Test Sequence Generation is a (set of) specification(s) in IDL which will be used as an input for the IUT (these are the test cases). The IUT produces its output in a target programming language, for which the mapping rules are defined.

The following activities produce means for the test result analysis: The mapping rules are an input to *Test Program Generation*, which has to produce test programs. Further input is the IDL grammar and the test input in order to generate the test programs only for constructs (and their values, if any) which are currently used to test the IUT. For each IDL construct, the Test Program Generation produces test programs (including the expected results and verdict assignments). The test programs and the compiler output are used by a Test Result Analysis to compare them, assign a result and produce a test report.

The developed test cases (i.e. test programs) can become part of a tool for an automatic test case execution and verdict assignment for IDL compiler test. Test specifications for compiler tests (i.e. the test input) can be generated from an input language syntax to achieve a nearly complete coverage of the language constructs. The test programs cannot by produced automatically: the mapping rules for IDL to a programming language are defined in a form of examples and many "equivalent" mappings are possible.

In CORBA it is due to the unavailability (i.e. there is no clear access point for the test system) of a well defined interface between the generated interface code (stubs and skeletons) on one side and any API on the ORB side, that the behavior can be tested only by a combined test of two corresponding interface programs or substitutes.

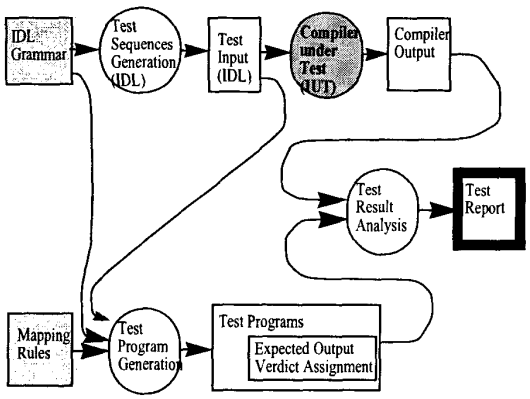


Figure 2: The development of IDL compiler conformance tests

3.2 Operation based interfaces

The use of IDL for interface definitions gives a clear specification for the operation signatures only and leaves open the operation semantics etc. Furthermore, proposals for testing (sample parameter structures, values and combinations) are missing in the CORBA specification. With the definition of the mapping rules of IDL to programming languages like C++, guidance is given on how to implement CORBA applications but also on the test system implementation. In particular, concrete information on the interface characteristics is given, which can be tested. It defines not only the proposed mapping, but also illegal application examples, possibilities, exceptions etc. Unfortunately, the majority of this information is informal only.

Test model

The question on generating operation based interface tests targets three major aspects, namely the definition of the ordering, the test pattern, and the test values. The ordering aspects covers both, the order of interfaces under test and the test ordering of interface features (e.g. basic types, operations, exceptions,...). Since there is less economical pressure (time and costs) to optimize a test campaign, there are no constraints on the order of tests on the static aspects of the API. However, any inheritance structure should be considered in order not to perform meaningless test cases, i.e. not to execute tests which depend on test which already failed. The same applies for the relationship between operations at different interfaces.

Test patterns, i.e. generic and universal test case structures, have been used to automate the test development process. Sample test patterns on the static tests have been described in [15]. The test body of semantic tests for operations are simpler and could be expressed using a case distinction for normal and exceptional behavior. This is already supported by some test tools like ADL[22].

More efforts are needed with respect to the selection of test values. Appropriate test data filtering techniques [11] could express the requirements and allow the translation into the test programming code. Unfortunately, developments in this research areas are still at the beginning, so that a lot of manual work is needed.

3.3 Message based interfaces

As explained in the previous section, the CORBA message based interfaces are not precisely defined. For example, the behavior of a client when receiving an unexpected message is not clear. As formal descriptions of CORBA interoperability protocols are not available, again manual test development is used mainly.

Test model

Testing of CORBA interoperability protocols follows classical OSI protocol conformance testing [17]. Testing the server side ORB is straightforward, since the server application is on top of the IUT (ORB under test), which can be implemented directly based on the server side ORB API. The client behavior can be a part of the means of testing (e.g. realized by a TTCN based test program).

Testing the client side requires an additional information exchange: after starting a client, the application must send a request to its ORB (under test) in order to enforce it to send an GIOP request to the server side test component. After this request, the server test component will send a reply message.

Another issue is that the GIOP/IIOP communication protocol does not support status requests. Thus, there is no possibility to verify whether the client ORB (under test) has accepted and transmitted any reply to the client application, which has been received from the server side. That means that the client application (or any other component) may require some suitable means to check the client ORB (under test) status. Further, the client application must support the server side test component, which gives the final verdict, i.e. the client application has to send back any received reply (using some request message). To avoid a deadlock, a timer must be introduced

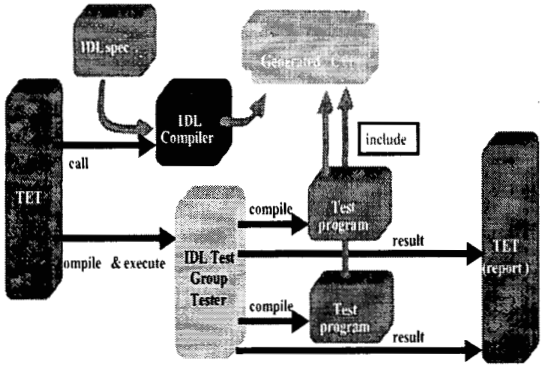
under the control of the client application and a corresponding signal for the server tester side in case of a time-out has to be send to the server side.

4 The Conformance Test Suite for CORBA ORBs

The conformance test suite for CORBA v2.3[3] has been developed on the basis of the analysis presented above. Tests for language mappings are covered by **IDL tests** to verify the syntactic aspects of the code generated by the ORB's IDL compiler and by **Stub and skeleton tests** - to verify the runtime behavior of the code generated by the ORB's IDL compiler. Tests for operation based interfaces are contained in **API declaration tests** to verify the declarations of the CORBA APIs (e.g. in form of C++ header files) provided with the ORB implementation and in **API behavior tests** to verify the behavior of the CORBA APIs provided with the ORB implementation. Message based interfaces are tested with **GIOP/IIOP tests** to verify the ORB's capability in terms of the syntax of GIOP messages and their exchange over IIOP. In the following, details on structure and techniques used by each test group are given.

4.1 IDL Compiler Test (IDL)

The tests are structured into test groups, test sub-groups and test case groups according to the logical structure of the language mapping. Each test case group consists of an IDL specification and a set of test cases, whereas each test case is a list of features to be tested. Tests for the new features of CORBA v2.3, e.g. value type, abstract value type and abstract interface, have been developed. There are about 400 IDL test cases available.

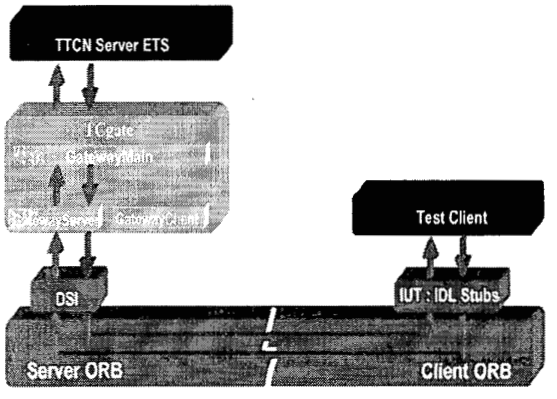


3. The conformance test suite for the CORBA v2.3 was released in Jan. 2001 and can be downloaded at http://www.opengroup.org/corval2/download.html

The test suite is executed with the TET[24] testing environment with a Java based GUI in the professional edition. The tests for C++ as target language are based on compiler error checks: for each test sub-group an IDL file is used as input to the IDL compiler under test. The generated code is then included in a codelet file and compiled using a target language compiler (e.g. a GNU C++ compiler). The assigned verdict depends on the success of the compilation.

4.2 IDL Stub and Skeleton Test (SII)

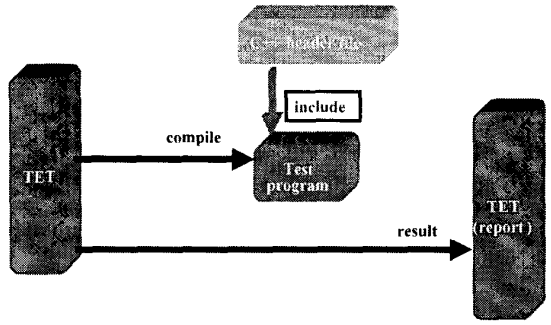
The tests are separated into two test groups for stubs and skeletons, respectively. Subgroups for tests on substitutability of value parameters and on passing instances of different valuebox types as parameters are defined. The stub test group and the skeleton test group contain appr. 380 test cases.



The test notation TTCN (Tree and Tabular Combined Notation[17]) in combination with the TTCN/CORBA gateway TCgate[9] is used. The test suite can be executed using command-line mode or using TTman[14].

4.3 API Declaration Tests (DECL)

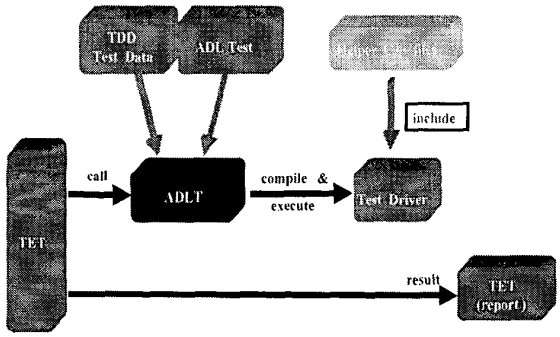
The structure of the declaration tests corresponds to the CORBA specifications. From the CORBA v2.1 test suite [23], a number of tests have been updated according to modifications of parameter types, structure extensions etc.



For the C++ declaration tests, a pattern-based approach is used. The tests use a compiler-error check, like the IDL tests, and are executed with TET[24]. There are about 1240 API declaration tests.

4.4 API Behavior Test (API)

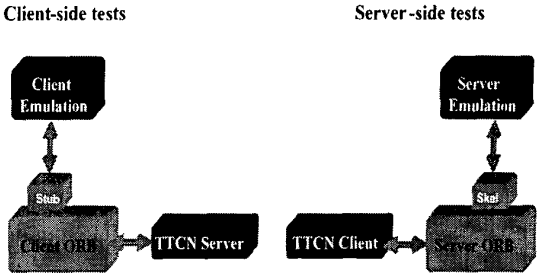
The tests cover operations and attributes of the considered APIs. There are available 470 test cases.



The tests are defined using the Assertion Definition Language (ADL) including the Test Data Description Language (TDD) [22]. The tests are executed with TET[24].

4.5 GIOP/IIOP Test

The GIOP tests are divided into server site and client side tests. Each group contains another three sub-groups on message ordering, on Common Data Representation (CDR) and on pseudo object types. The tests contain 360 test cases for different GIOP versions (1.0, 1.1 and 1.2).



The test notation TTCN[17] is used for the definition of GIOP/IIOP tests. The test suite can be executed using command-line mode or using TTman[14].

4.6 Java ORBs

In CORVAL2, a separate test suite is under development, which offers conformance tests for an ORB implementation with Java based interfaces. The test suite structure has been kept but some of the test techniques are changed. A new approach has been selected with respect to the static syntax tests, i.e. IDL compiler and API declaration tests. In Java, they are based on a comparison of Java classes with Java classes of a reference implementation. Here, the standard reflection package in Java is used. API behaviour tests are implemented with the new version ADL 2.0, which supports testing of object-oriented Java interfaces, too. In case of TTCN based tests, the TTCN-based test components could be reused, but the client and server emulation have been ported to Java.

4.7 General remarks

Test development

Due to the variety of aspects of the CORBA conformance test several technics has been applied. Some test has been written in target programming language, some in an abstract way using ADL and TTCN. This had an impact on the test development time.

The test written in a programming language (C++) had to be totally rewritten, when mapping to Java has been considered.

For the tests developed in TTCN, only small parts coded in C++ had to be ported to Java. For the stub and skeleton tests, a TCgate[9] written in C++ could be reused for Java ORB: the tester contains a reference ORB (C++), which communicates via GIOP/IIOP with a tested (Java) ORB.

However, to develop a test for CORBA in TTCN, a mapping of IDL (CORBA) types to programming language types has to be provided and made available during the execution of the tests. Additionally, the gateway TCgate is needed to transform the TTCN messages to CORBA calls. Thus, the usage of TTCN consumed more time for test development at the beginning, but required less effort when porting to Java.

ADL is more abstract then a programming language, but uses many constructs from the programming language - there exists dialects of ADL for particular languages. Therefore, only some parts of tests developed for testing C++ ORB could be reused to test Java ORBs.

The CORBA specification is informal, what makes it more difficult to develop a list of test objectives. Misunderstandings on the standard or insufficient information in the standard led sometimes to the need for clarification with ORB vendors and the OMG.

Test campaign

To test different aspects of CORBA ORBs, suitable methods have been developed and applied, which provide means and tools for test execution. This led to a variety of methods and tools, which can be used for ORB conformance tests. Some of the methods (like IDL compiler test, API declaration test) depend on the target programming language and e.g. differ for C++ and Java tests.

At the beginning of the project, the user which performs a test campaign, had to deal with two test management tools (two GUIs), one for TTCN tests and one for other tests, and different test report formats. Both GUIs have been compared and it has been decided to run the TET test under the control of TTman, too. Now, a user is able to run the complete test suite with one GUI only. Since execution of the large number of tests and the analysis of the results is very time consuming, additional utilities for automated test setup (e.g. test scripting) and reporting (e.g. test result summary) are provided.

It is recommended to start a test campaign with the tests to check the syntax, like the IDL compiler and API declaration tests, because already in this phase some missing/wrong code can easily be discovered.

5 Conclusions

Testability of complex systems is a key aspect for the quality assurance of such systems. According to [8], an ideal development of object-oriented systems is iterative and incremental. It must be accompanied by short code/ test cycles to allow iterative and incremental test planning, design and execution. This approach supports the testability of a system under development, but cannot be taken in all cases (e.g. if larger subsystems have to be integrated).

This paper gives an insight into some theoretical and practical questions on testability of CORBA systems, covering CORBA system aspects on language mappings, operation based and message based interfaces. In addition, considerations on practical problems with respect to test implementation and test derivation are included.

The C++ test suite presented in this paper comprises more than 2500 tests and have been applied to different ORB implementations, i.e. Orbacus (IONA), InterStage (Fujitsu) and MICO (Open Source). Due to its volume and coverage, the test suite helps to discover and correct specific failures in the ORBs. The successful application of this conformance test suite is a definitive step towards an interoperable CORBA infrastructure. In comparison to usual CORBA application test tools, it is the broad range of details which enables the system implementor not only to know about possible failures within a product but also to get detailed knowledge on the specific error type and location.

System specifications, informal and formal parts, can be used to establish a catalogue of fixed test objectives. However, they do not contain sufficient information needed to develop unique tests. For example, the variety of system parameters may lead to a wide range of possible test instantiations and implementations. Therefore, there is a need to provide clear and unambiguous system specification and to enhance them with test related information. In the context of language mapping system aspects, an informal approach for test generation has to be taken since in CORBA developers are free in the design and realization of their implementations. Also, the review of available abstract interface specifications leads to the understanding that none of the currently used description techniques covers the full information required for testability, i.e. sufficient specification of data and dynamic behavior. However, there is an approach (the reference point facet approach [9]) for refining the specification of ODP reference point specifications. This will be used to enhance the specification and testability of object-oriented interfaces. Further, the specification of test data needs to be restricted to a practical scope using some filtering specification techniques [11]. If this information will be specified by appropriate test description techniques, it can be used during the test generation phase, too.

It can be concluded that in all different areas, there is a need to close the gap between a system requirement specification and its implementation in order to strengthen

testability. The UML approach, which combines several viewpoints (including implementation) in one framework, seems in particular to be valuable in order to enhance testability and the confidence in product quality. A UML test profile is of specific importance to enable the direct test case development in the context of UML specifications.

6 Acknowledgments

The work is partially supported by the European IST Project Corval2 (IST-1999-11131, Enhanced Techniques for CORBA Validation, http://www.opengroup.org/corval2).

We want to thank our project partners for the fruitful cooperation in Corval2: The Open Group, IONA Tech., Fujitsu, Eric Leach Marketing Ltd., and Object Management Group. Special thanks for their contribution to the development of the test suite are given to Dr. Uwe Seimet, Cormac McKenna and John Wigram.

7 References

[1] B. Baumgarten, H. Wiland: Qualitative notions of testability. In: A. Petrenko, N. Yevtushenko (eds.): Testing of communication systems. Kluwer Academic Publishers, 1997.

[2] B. Baumgarten, O. Henniger: Testability with unbounded testing strategies. In: G. Csopaki, S. Dibuz, K. Tarnay (eds.): Testing of communicating systems, Kluwer Academic Publishers, 1999.

[3] A.S. Boujawah, K. Saleh: Compiler test case generation methods: a survey and assessment. Information and Software Technology 39 (1997) 617 - 625, Elsevier Science B.V. Amsterdam 1997.

[4] R. v. Binder: Testing Object-Oriented systems. 1999.

[5] O. Charles, R. Groz: Basing test coverage on a formalization of test hypotheses. In: M. Kim, S. Kang, K. Hong (eds.): Testing of Communicating systems, Vol. 10, Chapman & Hall, 1997.

[6] R. Gotzhein: Open Distributed Systems. Vieweg Verlag, Wiesbaden 1993.

[7] G. Holzmann: Design and validation of computer protocols, Prentice-Hall, London 1991.

[8] H. König, A. Ulrich, M. Heiner: Design for testability: a step-wise approach to protocol testing. In: M. Kim, S. Kang, K. Hong (eds.): Testing of Communicating systems, Vol. 10, Chapman & Hall, 1997.

[9] M. Li, I. Schieferdecker, A. Rennoch: Testing the TINA Retailer Reference Point.- ISADS'99, Fourth International Symposium on Autonomous Decentralized Systems, Tokyo (Japan), 1999. (describes TCgate).

[10] R. Orfali, D. Harkey, J. Edwards: The essential client/server survival guide. 2nd edition, Wiley computer publishing, New York 1996.

[11] A. Rennoch, J. de Meer, I. Schieferdecker: Test Data Filtering. FBT'99, Munich, June 1999.

[12] J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language. Reference Manual. Addison-Wesley, Reading 1999.

[13] I. Schieferdecker, M. Li, A. Rennoch: Incremental Testing at System Reference Points. - The IFIP 13th Intern. Conf. on Testing of Communicating Systems, Ottawa (Canada), Aug. 29 - Sept. 1, 2000.

[14] T. Vassiliou-Gioles, M. Li, I. Schieferdecker, M. Born, M. Winkler: Configuration and Execution Support for Distributed Tests. - IFIP 12th International Workshop on Testing of Communicating Systems (IWTCS'99), Budapest (Hungary), Sept. 1999. (describes TTman)

[15] CORVAL2: Deliverable 1 - Evaluation of the Existing Concepts and Methods Applicable for CORBA Validation, Mar. 2000.

[16] CORVAL2: Deliverable 11 - Specification of the CORBA v2.3 Conformance Test Suite, May 2000.

[17] ISO/IEC 9646-3: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN), edition 2, Dec. 1997.

[18] ITU-T Rec. X.901 | ISO/IEC 10746-1: Information Technology - Open Distributed Processing - Reference Model: Overview, Aug. 1997.

[19] OMG: The Common Object Request Broker: Architecture and Specification, v2.3, Jul. 1999.

[20] OMG: Product Profile, CORBA v2.3, psdef/99-11-01, Nov. 1999.

[21] OMG: CORBA C++ Language Mapping Specification, v2.3, Jun. 1999.

[22] The Open Group: Assertion Definition Language (ADL): http://adl.opengroup.org

[23] The Open Group: CORBA Verification Suite User's Guide, v1.1.1, Sep. 1999.

[24] The Open Group: TETware: http://tetworks.opengroup.org.

[25] TINA Consortium: Object Definition Language Manual. v2.3, July 1996.