Operation-based Interface Testing on Different Abstraction Levels[1]

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

Fraunhofer Institute for Open Communication Systems (FOKUS)
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
phone: +49 30 3463-7000, fax: +49 30 3463-8000

Email: corval2@fokus.gmd.de

www.fokus.fhg.de/tip

Abstract: Tests for Operation-based Interfaces can be implemented with concrete programming languages like C++ or Java. To have test system independent test descriptions and to gain better test reusability, in many companies more abstract test specifications are applied in this context. This paper discusses our experiences from a test suite development project for the OMG CORBA specification. In particular, the applicability of two major techniques - ADL and TTCN - has been investigated. This paper reports on the experiences and gives engineers some help to decide between the approaches for test development for operation-based interfaces.

Keywords: Conformance Testing, ADL, TTCN, Test Specification, CORBA

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

1 Introduction

The acceptance of system and software engineering methods in the developer's community is different, and even difficult if we think about "formal" methods. In most cases there is a common understanding that engineering methods, techniques and tools are very useful in system and software developments. It is clear that each engineer applies some of the existing IT engineering approaches. They may be characterized as more or less tricky or brilliant - and also as more or less abstract.

In the context of system and software development the meaning of *abstract* is used in opposite to the notion *concrete*. For example, consider a description of an action sequence: If due to the development level of its realization it has not been defined in all details it is called abstract (e.g. OMG's Object Constraint Language). On the other side, a C++ or Java program implementation of this behaviour, which can be compiled and executed, will be regarded as being concrete. The relationship of an abstract description towards a corresponding executable implementation is used to indicate its degree of abstraction.

We will not discuss the advantages and disadvantages of the abstract viewpoint in general. But we will offer our observations and experiences from an industrial oriented project on the development of CORBA (Common Object Request Broker Architecture) conformance tests using test development techniques on different abstraction levels.

The CORBA conformance test suite (so-called VSOrb) has been developed over years. It started in 1997 with the CORVAL project. An extension of the test suite has been required since the CORBA specification itself has been extended significantly by the OMG (Object Management Group, an industrial consortium). This work[2] is currently ongoing in the CORVAL2 project. Due to the long history of the test suite development, different approaches in the design and realization of the test implementation have been adopted.

In this paper, we will introduce the techniques used by the different approaches for operation- based interface (OBI) testing. In particular, we will discuss our observations on the design and realization of test cases implemented in pure C++ or Java, the Assertion Definition Language (ADL) [2], and the Tree and Tabular Combined Notation (TTCN) [1]. ADL and TTCN are different in the extent of their abstraction. A comparison of our experiences with those two main techniques ADL and TTCN provides an insight into the qualification of these approaches for testing of operation-based interfaces.

2 Testing target and methods

The CORVAL2 project addresses testing of all CORBA conformance aspects: language mapping (compiler) testing, classical protocol testing, and operation-based interface testing. In the following we restrict to the operation-based interface testing.

An interface definition describes the access to an object, which has a behaviour and contains data. An interface which is defined by a set of possible operations on the object is called an Operation-Based Interface (OBI) in this work. Any parameter data used with the operation is also defined at the interface. At the user's side (at the client), the operation terminates with either a reply or an exception raised at the interface by the object, i.e. an OBI acts in a synchronous mode.

From the methodological viewpoint we follow the conformance testing approach [5] to test the functionality and correctness of a given implementation against a specification. It uses black- box testing, i.e. the internal structure

2. The test suite for a C++ interface is free available and can be download at http://www.opengroup.org/corval2/download.html. The test suite for a Java interface is subject to licensing.

of the Implementation Under Test (IUT) remains hidden. Only at given Points of Control and Observation (PCOs) the tester is able to stimulate the IUT and to observe the resulting behaviour, i.e. the signal flow across the PCO. The generic test architecture for conformance testing in a client/server architecture is shown in Figure 1. In general the tester is the software application that implements the test purposes. The communication between the tester and the IUT is described by test events. Two communication modes have to be distinguished:

- an asynchronous mode, i.e. signals between tester and IUT are independent, there is no need to receive any response (reply or exception) due to a request, and
- the synchronous mode, i.e. the sender of a signal is blocked until it receives a response from the receiver.

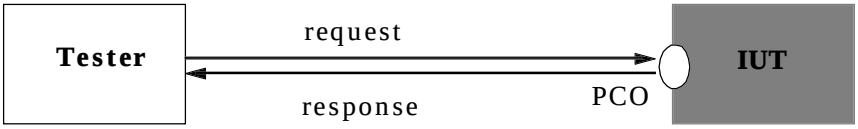

Figure 1 Conceptual test architecture

CORBA API - An Example for OBI-testing

The CORBA specification defines the object model and the OMG Interface Definition Language (IDL). In the object model, an object is an encapsulated entity with a distinct immutable identity whose services can be accessed only through well-defined interfaces, where an interface is a description of a set of possible operations on the object and it is specified in IDL. An operation is an identifiable entity that denotes a service that can be requested. IDL allows interfaces of objects to be defined independently from the object implementation.

The specification of an OBI in CORBA comprises IDL definition and semantics specification:

- IDL definition, which includes the definition of operations with in/out parameters and return values, and definition of types, structures, constants, enumerations, exceptions, further interfaces etc.
- semantics specification (in natural language),
  which denotes the normal behaviour and exceptions of the operations at the interface.

In addition, the mapping of IDL to programming languages such as C++, Java will be considered, too.

In CORBA object model, an object system is a collection of objects and provides services to clients. A client request services by issuing requests. A request consists of the operation, the parameters, the target object reference and an optional request context. Hence, the OBI testing approach in general follows a typical object service testing, i.e. a classical request / reply (incl. exceptions) communication at the IUT interface.

3 Testing techniques

In a first approach many of the CORBA tests have been implemented with the target programming language only. Such an approach appears to be obvious since in any case the access to the IUT has to be implemented using the target language. It is an advantage that the test engineer do not has to apply any additional test specific language, and therefore no translator or compiler is required. Test programs can be adopted by anybody who is familiar with the API and the required programming language. There are some general software development strategies to support the understanding and to keep the overview on a test:

- strict application on naming conventions, e.g. to separate tester and IUT related variables or objects,

- appropriate separation of source code in several files to allow both easy access to different test case parts (preamble, test body, etc.) and a suitable usage of line indentations (which appears difficult if too many parenthesis levels are in one file)

Both requirements have been fulfilled in the CORBA test (v2.1) written in pure C++ and Java. At the end of the implementation process a huge of source code files have been introduced even to implement only one test. The separation of different aspects was driven by the wish to reuse many parts of the source code in several tests and to allow quick access to specific parts of a test. For example, a simple Java test requires the following files:

- a Java file to implement the call of the operation under test and the analysis of the results,
- a Java file for the definition of concrete parameter value within the test,
- the “main” test case scenario to coordinate the usage of parameters within the test.

A series of class definitions are required to

- organize common operations on the IUT (e.g. initialization activities on the ORB),
- to implement reporting (e.g. infos at runtime or test results).

ADL

The Assertion Definition Language (ADL) [2] provides formal syntax for the functional testing of software components. ADL version 1.1, which has been used for Operation-based interfaces implemented in C++ was originally designed to test the behaviour of C routines. The usage of ADL 1.1 for C++ requires the implementation of wrapper routines[3]. ADL specifications address the definition of the interface to be tested with both

- a description of the behaviour expected at the interface, i.e. the test behaviour, and
- a description of the data parameters used in combination with the test behaviour.

The behaviour definition can be seen as an extension of the definition of the operation to be tested from the perspective of a client, e.g. only public methods and fields can be tested. In this sense these expectations at the interface are semantic annotations to the IUT. They are separated from the Test Data Description (TDD) which is used to describe the test data and its usage with the test cases. Both ADL and TDD specifications are input to the ADL Translator (ADLT) which generate a programming language version of the tests.

Further it is allowed to apply programming language constructs in ADL and TDD definitions. Thus an ADL specification depends on the target language.

The basic scheme of an ADL test reflects the classical structure of a standardized conformance test case:

- determination of some prerequisites (the test preamble),
- call of the operation under test,
- optional: restoration of the initial state of the SUT (the test postamble)
- calculation of boolean (assertion) values based on the outcome of the test,

3. The new object-oriented version ADL2 avoids wrapper routines and is applicable to Java, too. However, the basic testing related concepts are equal to ADL 1.1.

- generation of one (boolean) report value based on a group of assertion values, similar to a test verdict.

In a single ADL assertion it is allowed to combine two types of expressions: i.e. expressions may be evaluated before (indicated by the call-state-operator @) or after the operation under test has been performed. Further, there is a language constructs available to catch and handle exceptions thrown by the IUT. ADL supports the distinction of normal and abnormal behaviour.

An ADL test description may require some helper functions, e.g. to check any feature status or specific result. The additional functions are implemented in the target programming language.

TTCN

TTCN [1] defined by ISO and ITU-T is a well established test notation for OSI protocol conformance testing [5]. In the context of CORBA testing TTCN has been applied in message- based interface testing only [7] [8]. Currently, the third edition TTCN (TTCN-3) [6] is developed by ETSI [3]. TTCN-3 is a flexible and powerful language for specifying many types of system tests over a variety of communication interfaces. The standard has been prepublished by ITU-T [10].

TTCN-3 is on syntactical level a major change to TTCN-2, however, the main concepts of TTCN have been retained and improved. New concepts have been included, so that TTCN-3 will be applicable for a broader class of system. Typical areas of application will be protocol testing (including mobile and IP protocols), service testing (including supplementary services), module testing, testing of CORBA based platforms, APIs etc. In addition, it allows the use of different presentation (display) formats, such as the tabular presentation format known from previous TTCN versions and the MSC based presentation format.

TTCN-3 is applicable for specification of reactive system over a variety of communication interfaces. It is designed to be used as a programming language. However it has its own data type concept independent from a target language, i.e. the test developer can define and use individual data structures (records etc.).

The abstract data type concept allows an integration (using the TTCN-3 *import* feature) of predefined data type definitions. The correspondence of OMG IDL types to TTCN-3 data types have already been defined. In [10] the mapping from IDL to TTCN-3 is provided. This allows a test developer to use e.g. CORBA_UShort (which maps to *Integer*) or CORBA_Object; (which maps to *address*).

In TTCN-3 the application to an operation under test requires the usage of several language features, at least:

- a special function “testcase”,
- introduction of test components types (at least the main test component MTC),
- interfaces, data types etc. form the target language have to be defined in TTCN-3,
- definition of a “port”, the abstract mechanism to communicate with the test system interface,
- “map” and “unmap” operations have to be used to connect and disconnect a port,
- encoding/decoding for the exchange of data with the SUT is required.

Please note that the last two features have to be implemented by the test engineer.

The last Synchronous communication in TTCN-3 is supported by the application of “.call(*operation_under_test*)” and “.getreply(*check*)” operations on the port. The usage of some *check* operation is subject to an user supplied implementation.

Relation to target language

An overview on some major differences of ADL and TTCN-3 w.r.t. their relationship to a target implementation language is given in Table 1. ADL can be considered as an annotation for different programming languages while TTCN-3 claims to be programming language independent. Due to these distinction we explain that the abstraction level of TTCN-3 is higher than ADL.

Another important aspect is the time required to develop the test suite specification and implementation for a specific test programming environment. Due to the language dependence of ADL the migration of the test implementation to another target programming language is not straightforward. On the other hand most parts of a TTCN test suite are language independent and do not require a migration e.g. from C++ to Java.

The different abstraction levels of ADL and TTCN have been illustrated in Figure 2. Please note that this in this paper is not intended to give a precise measure or means of measurement for the level of abstraction. This section should emphasize the different nature of the two testing techniques.

Features	**ADL**	**TTCN-3**
target language scope	C, C++, Java	depending on compiler
IUT reference	ADL(adlclass)	port (with signature)
result types	boolean	verdict: pass, fail, inconc, none
data type	target language	TTCN specific
adaptation	include	Real Test System Interface (tbp)
inclusion of programming statements	in ADL and TDD	“external” operations
exception handling	“try ... catch” structure	explicit with “.catch”
Abstraction level	+	++

Table 1 ADL and TTCN-3 relationship to programming languages

Figure 2 Abstraction level of ADL and TTCN-3

4 Evaluation of ADL and TTCN-3 for OBI testing

Based on an in depth analysis of the test specification developed in ADL and TTCN-3, some advanced features for test description are selected and compared in the following.

Description of test system configuration

TTCN-3 supports data types, but also special types associated with the configuration such as *address*, *port* and *component*, which are used to define the architecture of a test system. A test configuration in TTCN-3 consists of a set of inter-connected test components with well- defined communication ports and an explicit (abstract) test system interface which defines the border of a test system. The type *port* provides an abstract mechanism facilitating the communication between test components, and between test components and the test system

interface. An abstract interface[4] of a test system component can be specified with the type component, which contains the communication ports. The test case is connected to the IUT through the ports of the test system interface. In addition, TTCN-3 provides the configuration operations such as create [3], which supports the dynamic configuration.

ADL does not provide any configuration data type such as test component and port. The test system configuration can not be described by ADL.

Test execution control

An important aspect of test execution control is the description of the relations between test cases and test groups, e.g. sequences, repetitions and dependencies on test outcomes. A test case in TTCN-3 is like a function and is defined in the definition part. The test case is executed due to the control part of a TTCN-3 module. The result of an executed test case is always a value of the type *verdicttype*, which can be: *pass*, *fail*, *inconc*, *none* and *error*. The basic program statement such as *if-else* can be used in the control part of a TTCN-3 module. Test cases in TTCN-3 are defined independently from the control part which is added to the TTCN-3 module. The control part provides the high-level logic within TTCN-3. An equivalent feature for this logic is not supported in ADL.

Test cases in ADL are independent. The ordering of their execution is not part of ADL.

On the other hand, in ADL, the test input data is described separately from the semantics specification of the function under test. The test directive defines a set of points of an input grid. The input grid is the set of test data points generated by ADLT for a specific test of a function. It is calculated as the cross product of all values of all properties of all test variables involved in a given test directive.

ADLT generates a test driver for each test directive. The test driver iterates over the grid of test data input and calls provide functions to get values for symbolic test variables. It passes the values to the Assertion-Checking Function (ACF) for the function under test. The ACF compares the actual behaviour of the function under test to the specified expectations. A test case (test directive) is *pass* if the function under test behaves correctly on all test data points. Going from this aspect, ADL provides the control of execution for the test data points. In comparison to TTCN-3, ADL has a partial control of test execution within a test case. On the other hand, this aspect of ADL facilitates the flexibility in the test data definition. It is one of the advantages of ADL.

Reuse the test behaviour

A test case in TTCN-3 is a special kind of function. It may be parameterized with their directed parameters. The reuse of tested behaviour can be realized by the parameterization of a test case.

Since there is no relation between test cases in ADL, in general the tested behaviour can not be reused. However some reusing features are supported. A test description of a class which is derived from another base class can inherit from a test description of the base class (ADL2.0 feature). A behaviour specification can be used in more than one test directives. Reuse of the test data description indicates that a test variable in TDD can be used in many test directive. This aspects are also supported in TTCN-3, e.g. a test case can be executed in the TTCN-3 control part for many times.

Time restriction for test execution

It is useful that a test case is executed within limited time to prevent infinite waiting. TTCN-3 supports the timer type with keyword *timer*, which can be declared and used in a test scenario definition.

ADL does not support the description of such aspects.

4. The real physical connection is outside the scope of TTCN-3. Its realization is considered within the implementation of the final Executable Test Suite (ETS).

Flexibility of test data

ADL separates the semantics specification from Test Data Description (TDD) for a function under test. TDD is designed for the description of test data and the specification of test directives. It is used to describe test inputs in a structured and coherent way. The test data can be abstract by specifying symbolic, independent properties. Or, it can be concrete by using literal test data description.

In addition, ADL provides the refinement of symbolic test variables. A test variable is composed of properties. It can be derived from a parent test variable. Any restriction to a symbolic test variable also limits the input for a test without any modification of an existing functions to create/destroy the variables.

In TTCN-3, testing completely involves specifying tests by their inputs and outputs. There is no flexibility in test data definition as on TDD for the data sent to the IUT. But TTCN-3 provides the matching mechanisms, which supports the test developer with a high flexibility to describe expected data.

Test reporting

The test outcome is important for the test result analysis. The invocation of the operation under test reported in both test techniques. In addition TTCN-3 provides test scenarios in a graphical presentation format (according to standardized Message Sequence Charts), too.

Based on the analysis, the comparison of ADL and TTCN-3 is summarized in Table 2.

Features	**ADL**	**TTCN-3**
Test system description	--	++
Control of test cases	+	+
Reusing of test behavior	+(restricted)	++
Time restriction for test execution	--	++
Flexibility of test data	++	--
Test reporting	+	++ (e.g. MSCs)

Table 2 A testing specific comparison of ADL and TTCN-3

5 Conclusions

This presentation provides an insight to the development of Operation-based Interface tests in the context of CORBA conformance testing. There are several techniques - on different abstraction levels - which have been used for the test specification and implementation. Two major approaches have been selected to demonstrate the suitability w.r.t. a set of test cases on the CORBA portable object adapter API: ADL as a proved industrial representative for an implementation oriented language and TTCN-3, the latest standardized test notation from the conformance testing community.

TTCN-3 is on a higher abstraction level than ADL. A comparison of both approaches explains their differences in details. Further testing specific differences are found due to the test configuration (covered by TTCN-3) and the flexibility of test data generation (covered by ADL). Both approaches are similar in their goal to be as much as possible concrete like a programming language but to satisfy the specific requirements to be a means for testing.

Test engineers who have to decide on any of the test techniques may base their decision on any existing requirement to implement the tests for different target programming environments. ADL tests have to be modified if the environment changes. It requires more work to implement OBI tests with TTCN-3, but their migration to other programming environments appears easier.

References

[1] ITU-T X.903, ISO/IEC 10746-3, “Open Distributed Processing - Reference Model, Part 3”, Geneva, Swiss, 1997.

[2] The Open Group: Assertion Definition Language. http://adl.opengroup.org. Reading (UK), 1997 - 2001.

[3] ETSI: Tree and Tabular Combined Notation, version 3. http://www.etsi.org/ptcc/ ptcc_ttcn3.htm, Sophia Antipolis (F), 2001.

[4] OMG, “Common Object Request Broker Architecture (CORBA)”, ver. 2.3, 1999.

[5] ISO/ITU-T/ETSI, “Conformance Testing Methodology Framework (CTMF)”, 1997.

[6] J. Grabowski et al.: On the design of the new testing language TTCN-3. Testcom 2000, Ottawa (CA), 2000.

[7] M. Li et al.: Testing the TINA Retailer Reference Point. ISADS, Tokyo (J), 1999.

[8] A. Mednonogov et al.: Conformance Testing of CORBA Services using TTCN. Testcom 2000, Ottawa (CA), 2000.

[9] A. Yin: Testing Operation-Based Interfaces. Diploma-thesis TU Berlin, 2001.

[10] ITU-T: Tree and Tabular Combined Notation version 3 (TTCN-3): Core language. http://www.itu.int/itudoc/itu-t/rec/z/index.html, 2001.

Authors

The authors are members of the Testing, Interoperability and Performance research group (TIP) at GMD FOKUS. They are developing advanced methods and tests to validate system quality from a user perspective.