Conformance Testing of TINA Service Components - The TTCN/CORBA Gateway

Ina Schieferdecker, Mang Li, Andreas Hoffmann
GMD FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
e-mail: {Schieferdecker, M.Li, A.Hoffmann}@fokus.gmd.de,
URL: http://www.fokus.gmd.de/step/

Abstract. This paper considers conformance testing of TINA service components in the realm of the ITU-T conformance testing methodology, which provides the only standardized, well accepted and widely used test methodology and test notation TTCN. It discusses the use of TTCN for testing computational objects of service components and proposes an implementation approach to derive executable tests from TTCN, which can be executed on any CORBA compliant ORB (with minor efforts for ORB specific adaptation) to check the functional correctness of deployed service components in distributed object environments. The TTCN/CORBA gateway is a general approach for testing distributed systems. TINA is taken as an example in this paper only. The work presented has been partially supported by the EC ACTS project TOSCA -TINA Open Service Creation Architecture, AC237.

1 Introduction

With the development of TINA (Telecommunications Information Networking Architecture) and the provision of new and complex TINA services such as telecommunication, management and information services, which may be deployed in the context of various distributed object computing environments like the OMG CORBA (Common Object Request Broker Architecture [8]) or OSF DCE (Distributed Computing Environment [10]), the need to be able to validate and test large and heterogeneous object systems is becoming increasingly important. In particular, it is not sufficient to validate TINA services on specification level (with one of the formal based object-oriented modeling techniques such as the SDL - System Description and Specification Language - based Object Modeling Technique [11]) only, but also to test them in the target environment. Testing may be used to check

- service components individually,
- conformance to TINA reference points,
- the individual service components working together as well as
- to check individual services working together in a multi-service environment.

A service component and service test is intended to determine whether deployed service components and services are ready to ship by observing how the service and service components perform in the target environment while attempting to emulate their real use. Hence, testing should encompass functional, performance, and robustness tests. It should also cover operational, installation and usability aspects of the service and service components. Testing is an essential activity in the pre-deployment phase of the TINA service life cycle

- for determining, whether services and service components conform to their specification,
- to simplify service procurement,
- to reduce the risks and costs related to purchasing services,
- to allow for unbiased comparison of various implementations and repeatability of tests using different means of testing and
- to increase the likelihood that services from multiple service providers are interoperable.

The paper concentrates on testing the functional aspects of service components and services. Functional testing is also referred to as conformance testing. The widely used methodology for testing the conformance of protocol implementations is described in [15], where also the only standardized test notation TTCN (Tree and Tabular Combined Notation) is defined. It should be noted that conformance testing can only show the presence of functional errors and cannot guarantee their absence. Consequently, conformance testing is not a means to prove a 100% compliance to a specification. In particular, exhaustive testing is not practical and generally too expensive. Therefore, any testing should concentrate on the essential and indispensable aspects of the tested system.

The paper is structured as follows: Section 2 gives a short overview on the TINA service creation process as defined by TOSCA and the role of conformance testing in this process. Section 3 considers related work on testing of distributed systems, while the specifics of testing TINA service components are discussed in Section 4. The mapping rules of TINA ODL specifications to TTCN declarations is described in Section 5, which is followed by an example in Section 6. The TTCN/CORBA gateway being an approach for executing TTCN test cases on CORBA platforms is presented in Section 7. Conclusions finish the paper.

2 Conformance Testing in the TOSCA Approach for TINA Service Creation

The increasing complexity of TINA services leads to the needs of a methodology for the creation of validated TINA services. An important goal of the TOSCA project [20] is to develop a methodology that supports the validation of TINA services in course of their creation process, i.e. before service deployment. Currently, TOSCA

works on an integrated tool set to automate the service creation process as much as possible.

TINA services are designed to run in a distributed service environment, which is defined by the general service architecture [4]. It consists of three layers: the access session, which provides an uniform access to all TINA services, the service session, which is started after an access session has been established for a customer and a service has been selected, and the communication session, which manages the network resources for the service. The service architecture defines a set of components, which provide means to segment the functionality of TINA services. A TINA Service Component (SC) encapsulates data and functionality. Service components are defined by means of an ODP (Open Distributed Processing [9]) computational viewpoint specification and are mapped to computational objects (COs) or computational object groups as defined in [6].

The work of TOSCA focusses mainly on the service session, but considers also aspects of the access session. To enable rapid service creation, the TOSCA project has adopted and extended the framework idea known from object-oriented technology. A TOSCA-framework is used to describe a family of similar services (so called service class) in a generic manner. It provides reusable classes of computational objects *and* well-defined configurations for them. It consists of a generic part, which is fixed for the service class, and a flexible part, which contains flexibility points. Flexibility points are place holders in the descriptions of computational objects for things like parameters, types, operations, and behaviour. Constraints can be used to restrict the set of possible replacements. Flexibility points have to be filled during the service creation process. For some of them default behaviours may be defined which may be replaced during the service creation process.

TOSCA-frameworks are intended to support both service designers, who are programmers with technical knowledge, and business consultants, who are not very familiar with technical details, in the service creation process. A service designer may create a new service by filling the framework's flexibility points manually with its technical expertise. This will be done mostly by specialization of abstract classes of the framework.

In addition, TOSCA proposes an abstract and less technical view on frameworks in order to ease the instantiation of frameworks by business consultants. Each abstract view on a framework is called a paradigm. Paradigm tools may offer predefined building blocks as plug-ins for flexibility points and provide user-friendly interfaces.

The TOSCA service creation process from the testing point of view (i.e. the creation of frameworks is not fully reflected) is shown in Figure 1. A TOSCA framework may be developed either from the scratch or through the generalization of an already existing service, i.e. the service is made more abstractly by adding flexibility points.

A TOSCA-framework has two representations: as implementation code, e.g. written in C++, and as SDL specification. The specification has to be a compliant model of the implementation code and is needed for validation. Validation methods will be

used for the abstract framework as well as for the specialized framework, where the flexibility points have been filled. It should be noted that during the framework specialization process, the flexibility points of the framework's implementation as well as those of the SDL model have to be completely filled (at least those flexibility points which do not have any predefined default behaviour). Once the specialized framework has been validated, TTCN test cases can be (semi-)automatically derived from the SDL model. The executable TTCN test system is used to check whether the running service that resulted from the framework's implementation code, meets the conformance requirements and is or is not ready for deployment.

An automation of the above described testing process is needed to make it efficient and repeatable and to make test results comparable. The starting point of any automated testing is the design and specification of test cases, which are used to check certain conformance requirements. Ideally, test cases are written in an abstract manner to be independent from hard- and software constraints. This purpose is well supported by TTCN. The conformance testing methodology in [14] defines an automated test process that includes the generation of executable tests from TTCN test cases, test execution, and test verdict assignment. This is mainly achieved by defining an operational semantics for each of the TTCN constructs. The semantics of TTCN is the basis for the development of TTCN analyzers, compilers and execution environments.

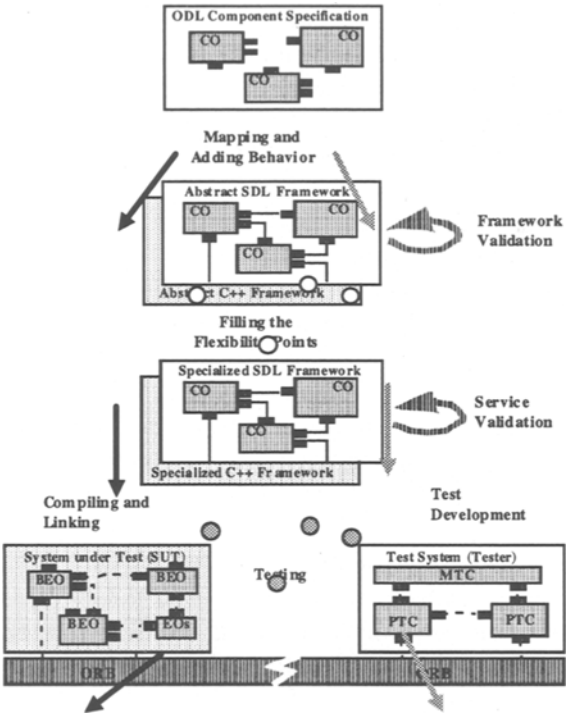

Figure 1: Testing in the TOSCA Service Creation Process

The information flows between the different notations used for specification, implementation and test case development are shown in Figure 2. The ODL (IDL) specifications of service components are the starting point for generating automatically SDL skeletons. Once the SDL model has been completed (e.g. by adding behaviour) and validated, it can be used to generate implementation code as well as to derive TTCN test cases. Another way for test case development is to obtain the data types directly from the ODL (IDL) specification, while the test case behaviour is developed manually. Furthermore, the IDL parts of the ODL specifications are used by the CORBA interface repository to support the interaction between the testing and the tested system.

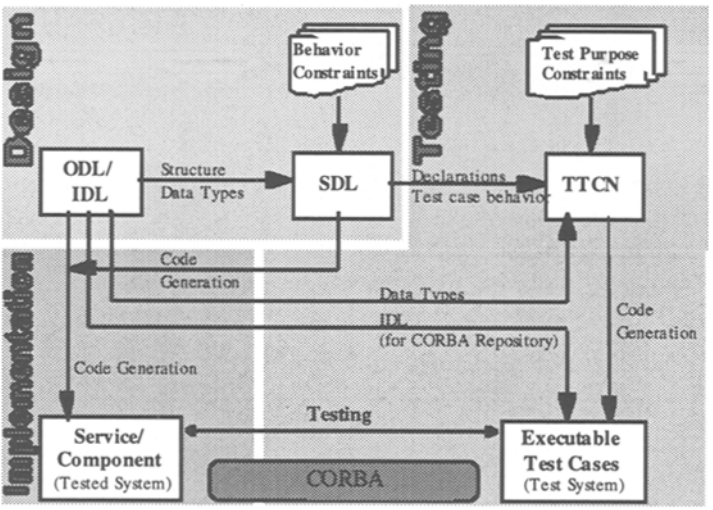

Figure 2: Information Flows in the Service Creation Process

3 Related Work

Testing of distributed object systems is an upcoming issue and has been investigated only recently (an overview is given in [19]). CORBA based test environments are discussed in [17] and [16]. To the authors best knowledge, neither results on the use of TTCN for testing distributed systems (only [16] recommends the use of TTCN without discussing technical details) nor on a testing methodology for TINA services (the OMG Test Special Interest Group was set up in 1996 to address this issue) are available.

The grade of testability of a distributed system is determined by the following aspects:

- The anticipated real-use scenarios of the system have to be covered by test cases. In practice, there are significant limitations due to the complexity of distributed systems, so that only a restricted set of test cases can be applied.

- Every significant event exchanged between the tested and the test system has to be observable. In addition, also the ordering and timing of those events have to be observable. Observability is the precondition to determine whether the tested system behaves correctly.
- The test system must be able to control the execution of the tested system, so that test executions are reproducible and so that test results are deterministic and repeatable.

According to [18], the concurrent nature of distributed systems is the source of making their testing harder than testing sequential systems:

- The probe effect, which reflects the effect of changed behaviour of a system when attempting to observe it, may occur.
- Racing conditions in concurrent activities may lead to non-reproducible behaviour.
- A synchronized global clock has to be realized for observability of test events.

These aspects are to a great extend determined by the system architecture of the tested and test system and are discussed for TINA service components in a subsequent section. Since the grade of concurrency in the test and tested systems defines the complexity of testing and determines applicable test architectures, a taxonomy for testing that depends on the grade of distribution is given in Table 1.

Table 1: The Testing Taxonomy (with Respect to the Grade of Distribution)

Type	System under Test	Test System	Selected Approaches
CC	centralized	centralized	ISO Conformance Testing: Peer-to-Peer [15]
CD	centralized	distributed	ISO Conformance Testing: Multiparty Testing [15]
DC	distributed	centralized	Test Execution of Telecommunications Services [16]
DD	distributed	distributed	General Design Rules for Distributed Tester [19]

TINA services are distributed and may be tested by means of a centralized or distributed test system. A centralized test system uses a sequential test case behaviour. Its ability to detect the interactions between the system under test (SUT) and test system at a central side implies a global knowledge of the test execution. The test events are causally ordered, what eases the test verdict assignment and makes test results repeatable. However, the complexity of a centralized tester is magnitudes larger than a distributed tester.

Due to complexity reasons, the work concentrates on a distributed test system. Such a test system can be structured similar to the system under test itself: PCOs per interfaces of service components or computational objects are controlled and monitored by parallel test components. Care has to be taken on the synchronization between the individual parallel test components: the parallel test components assign test verdicts on the basis of their local knowledge about the test execution. A main test component (MTC) is used to coordinate them and to accumulate the individual local test verdicts into the global one.

4 Test Objectives for TINA Service Components

Conformance testing considers the system under test to be a black-box, i.e. the internal structure and the internal behaviour is invisible. The test system interacts with the system under test only via the interfaces and assigns test verdicts after comparing observed reactions from the system under test with the expected ones.

The TINA business model [4] defines business roles and business relationships: a consumer uses services that are provided by a TINA system, a broker supplies information that enables a stakeholder to find other stakeholders and services, a retailer serves stakeholders by offering them access to services, a third party provider supports retailers or other third party providers with services, and, last but not least, a connectivity provider manages the communication network. The relationships between business roles are defined by means of reference points. The TINA service architecture defines the broker, retailer, third party service provider, and retailer-to-retailer reference points. It should be noted that currently only the retailer reference point is standardized. It supports the consumer's needs for accessing services from a retailer and offers functionality for the access part, e.g. discovery of services or initiation of usage, and usage part, e.g. control and management of sessions. Reference points are inter-domain reference points, since every business role is performed by a separate business administrative domain.

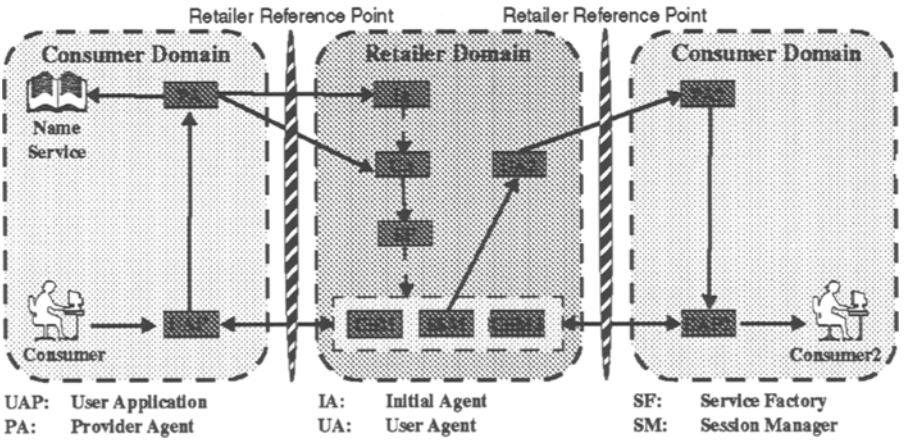

Figure 3: Example of Consumer-Retailer Roles Using Service Components

Consequently, reference points have to be tested to validate the correctness of interworking of business domains. Testing of reference points refers to testing the interfaces of those service components that interact at the respective reference points.

Another objective for testing would be to test the individual computational objects that are used for realizing service components. Again, a black-box approach can be used to test the computational objects via their interfaces and by abstracting from their internals. From the TINA service architecture perspective, testing of

computational objects is based on the structuring information of service components. It uses therefore a grey-box testing approach. The service components are not considered to be a black-box, but rather their internal structure on the configuration of computational objects with their interfaces has to be known. Due to political reasons, it might be practically useful for in-house tests rather than for third-party tests at an external test lab. However, testing the individual computational objects can use the same testing technique like testing service components - it is the question which implementation parts are accessible and observable by the test system.

5 The TINA ODL to TTCN Mapping

A TINA service component (SC) is defined as a single computational object (CO) in ODL (Object Definition Language [7]). All interfaces of the SC are defined in OMG IDL (Interface Definition Language [8]). It should be noted that this is only a CO representation of the SC, which is used to unambiguously define the interface of the component. However, that does not mean that the distribution of SC is restricted to a single DPE (Distributed Processing Environment) node. In fact, there may be several CO mappings for an SC. TINA ODL is a superset and extension of the OMG IDL which is defined by the Object Management Group (OMG) in the CORBA (Common Object Request Broker Architecture) context[1]. The specification of the interface of a service providing object in OMG IDL defines what operations are available by the object and how they should be invoked, independently of technology and programming language. In such a manner, equivalent information is offered to all service requesters (clients) from whom implementation details of service providers (servers) are hidden.

For the computational interfaces of TINA services components, specifications in IDL give adequate basic information for the development of TTCN test cases. An SC is considered to be a black-box, so that groups of computational objects cannot be distinguished from single computational objects. In particular, each interface of the SC is tested separately (see also below).

Mapping rules are defined for the translation of information from IDL specifications into TTCN compliant representations. They are described in the following:

- An IDL module provides a name scope and a mechanism to group interfaces. Although the second edition of TTCN supports a module concept for, besides other things, name scoping, it is currently not supported by any TTCN tool environment. Since the presented work aims at finding a practical approach for testing TINA service components, we decided to stick with the previous, tool supported version of TTCN. Therefore, an IDL modules and all other name

[1] ODL extends IDL with the concept of streams, with structuring features for groups of computational objects and with the ability to define multiple interfaces for computational objects. Testing of streams is a separate issue for future research.

scopes are flattened. Scope names are mapped to prefixes, which are added to the identifiers of the types contained in the scope.

- For every IDL interface type a separate test group is introduced to structure the TTCN test suite in accordance with the interface structure of the tested service component. For each service component with multiple interfaces a test group is introduced, which contains sub test groups, one for each provided interface.
- Interface inheritance is not supported by TTCN. Therefore, all user-defined types, attributes and operations which are inherited from the parent interface(s) and are not redefined, have to be duplicated in the TTCN specification for the derived type.
- An IDL operation describes a method that can be invoked from outside. Since TTCN uses asynchronous communication for the exchange of test events, IDLs synchronous mode of operations has to be emulated: A synchronous IDL operation is mapped to two ASP (abstract service primitives) types:
  - an ASP type for requests on the operation. The identifier of the ASP type is composed of the prefix "pCALL__" followed by the corresponding scope names and as last the operation name, e.g. `pCALL__TINA RetRetailer__i_RetailerInitial__requestNamedAccess`. The "in" and "inout" parameters fill the body of the ASP type definition as a SEQUENCE, keeping the order of the parameters in the IDL specification.
  - a second ASP type for replies of the operation. The identifier of the ASP type is composed of the prefix "pREPLY__" followed by the corresponding scope names and the operation name, e.g. `pREPLY__TINA RetRP__i_RetailerInitial__requestNamedAccess`. The "inout" and "out" parameters fill the body of the ASP type definition as a SEQUENCE, keeping the order of the parameters in the IDL specification. In addition, an optional content field for the return value is also reserved.

  An asynchronous IDL operation with the attribute "oneway" is mapped only to an ASP type with the prefix "pCALL__".
- An IDL exception describes exceptional cases during operation invocation or execution. It is mapped to an ASP type. The identifier of the ASP type begins with the prefix "pRAISE__", which is followed by the appropriate scope names.
- An IDL attribute definition is logically equivalent to describing a pair of accessor functions: a „get" function to retrieve the value of the attribute and a „set" function to set the value of the attribute. Therefore, IDL attributes have similar mappings as IDL operations. A read-write attribute is mapped as the following:
  - The „set" function is mapped as an „one-way" operation, to a request ASP type with an „in" parameter of the same type as the attribute.
  - The „get" function is mapped to a request ASP type without parameter fields and a reply ASP type with a content field for returned attribute value.

  For „read-only" attributes only the „get" function has to be defined.

- Most of the IDL basic types are mapped to ASN.1 types of the TTCN test suite. The IDL types "float" and "double" cannot be mapped, because TTCN does not support floating types. IDL "typedef" are translated to TTCN ASN.1 type definitions.
- IDL constants are mapped to TTCN constant declarations.

The mapping rules support only the translation to TTCN type declarations. The constraints part and especially the dynamic behaviour part of a TTCN test suite must be added in subsequent steps of test suite development. It should be based on the service component specification and can be developed either manually or semi-automatically by the use of test case generation methods. For example, [12] describes an approach of deriving SDL skeletons from ODL/IDL specifications, enhancing it with object behaviour and using the completed SDL specification as a basis for test case derivation. Test case derivation tools such as SaMstAG [16] can be used here.

6 An Example - Test Cases for the Service Access Session

This section presents selected aspects of test cases for the initial access interface of the TINA Retailer Reference Point. The TTCN test cases are developed by using the TINA ODL to TTCN mapping rules presented in Section 0. A test case that verifies the very first interaction between a consumer and a retailer, is shown as an example (see Figure). The dynamic behaviour of the test case has been developed manually on the basis of the textual description of an example scenario of the TINA Ret Reference Point Specifications [5].

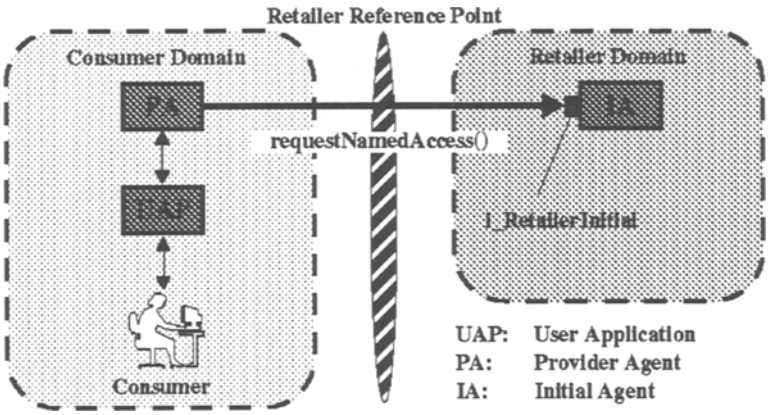

Figure 4: Configuration for the Access Session Example

The implementation under test (IUT) is in this case an implementation of the interface `i_RetailerInitial` provided by a retailer. The TTCN test system emulates a consumer. The behaviour of the test presented in the figure below can be read as follows:

- The reference of the interface `i_RetailerInitial` to be tested is gained and connected with the test system via a PCO named `PCO1__i_RetailerInitial` (line 1).
- Via this PCO, a request on the operation `requestNamedAccess()` with valid identification information is sent (line 2).
- When the expected reply is received, a preliminary `PASS` verdict is marked, the reference to an `i_RetailerNamedAccess` interface as well as the access session secret ID and the access session ID are extracted from the message (line 3).
- If the test suite operation `activePCO()` for connecting the appropriate PCO with the interface `i_RetailerNamedAccess` returns successfully, the access session is established (line 4). A postamble closes the access session in this case (line 5). Otherwise, an `INCONCLUSIVE` verdict is given and the test case is terminated.
- The test case ends also with a `FAIL` verdict, if replies other than the expected one, e.g. exceptions, are received (line 7). It shall be mentioned that a simplified presentation of the test case is shown here. For example, a timer should be added to constrain the time a client awaits reply.

Test Case Dynamic Behavior

Test Case Name: EstablishAccessSession1
Group: TINARetRetailerInitial
Purpose: To verify that in the case that CORBA security services are used, on receipt of an invocation on the operation requestNamedAccess() of the interface TINARetRetailerInitial::i_RetailerInitial, a reference to a TINARetRetailerAccess::i_RetailerNamedAccess interface is returned.
Selection Ref: SecurityServiceUsed

Nr	L	Behavior Description	Constraints Ref	V	C
1		+GetInitialRef (PCOName_PCO1_i_RetailerInitial, ObjName_ _i_RetailerInitial, ref_i_RetailerInitial_usr1)			
2		PCO1_i_RetailerInitial !pCALL_TINARetRetailerInitial_i_RetailerInitial_ _requestNamedAccess	pCALL_i_RetailerInitial_ _requestNamedAccess_s1		
3		PCO1_i_RetailerInitial ?pREPLY_TINARetRetailerInitial_i_Retailer Initial_requestNamedAccess (ref_i_RetailerNamedAccess_usr1:= pREPLY_TINARetRetailerInitial_i_Retailer Initial_requestNamedAccess.namedAccessIR, asSecretID_usr1:= pREPLY_TINARetRetailerInitial_i_Retailer Initial_requestNamedAccess.asSecretId, asID_usr1:=pREPLY_TINARetRetailerInitial_ _i_RetailerInitial_requestNamedAccess.asId)	pREPLY_i_RetailerInitial_ _requestNamedAccess_r1	(P)	
4		[activatePCO (PCOName_PCO1_i_RetailerNamedAccess, ref_i_RetailerNamedAccess_usr1) = TRUE]			

5		+EndAS (asSecretID_usr1, SpecifiedAccess Session_s1, EndASOption_usr1)			
6		[activatePCO (PCOName__PCO1__i_RetailerNamedAccess, ref_i_RetailerNamedAccess_usr1) = FALSE]		I	
7		?OTHERWISE		F	

Figure 5: A Test Case for the Service Access Session

7 The TTCN/CORBA Gateway

In this section we discuss the implementation of abstract TTCN test cases in a CORBA environment. CORBA is an object-oriented architecture for a distribution transparent communication supporting client-server applications. The central component of CORBA - the ORB (Object Request Broker) - is responsible for transparent relaying object requests, with the help of static or dynamic stub and skeleton interfaces. Object services such as Naming Service, Interface Repository and Implementation Repository offer persistent information at run time. The standard CORBA interfaces are specified using OMG IDL. Mappings for programming languages C, C++, Java, Smalltalk are already defined in[ ] to support CORBA based multi-lingual implementations. CORBA is the basic technology of TINA. As mentioned in section 5, the definition language TINA ODL has OMG IDL as the integrated part for specifying interfaces of computational objects. Most of the known implementation of TINA-DPE and TINA services are implemented using CORBA.

In order to test a CORBA based implementation, the test system must be integrated into the CORBA environment. With TTCN and the mapping rules introduced in section 5, the specification of related behaviour of such a test system on a high abstraction level is provided. The key issue of the realization of the test system is the bridging between a message-oriented non-CORBA system and the CORBA ORB. The basic issues of bridging within CORBA, e.g. building inter-ORB bridges and interceptors [8], and between CORBA and non-CORBA systems, e.g. TMN-CORBA interworking [1], have been discussed in a number of documents. The goal of the work presented in this paper is to provide a practical approach for the execution of TTCN-based test cases on a CORBA platform.

A typical TTCN code generator provides the translation of abstract test cases to language specific code, and an open interface (in form of function calls) for the adaptation of the test system to the particularities of the SUT. The adaptation includes the implementation of the TTCN snapshot mechanism, the implementation of the mechanism for dispatching ASPs/PDUs, which depends on the manner in which the communication to/from the SUT is carried out, and other functionality like configuration of the test system and management of timers. In addition, the SUT dependent parts of the encoding and decoding functionality for messages have to be provided.

In our case, the SUT is specified using OMG IDL and is implemented using CORBA as the underlying communication medium. Since the message processing mechanisms are CORBA specific but not dependent on particular SUT types, the adaptation part and the major part of the encoding/decoding functions are generic for all test cases and can be generated applying the same mapping rules. However, the generated code normally has tool specific function calls. In order to build a modular test system with reusable components, a tool-independent component called TTCN/CORBA Gateway is introduced. This component is designed to have the following properties:

- An interface is provided to support TTCN specific functions, such as sending and receiving of messages. Additional operations for the support of test systems with distributed test components shall be offered.
- It is capable to communicate with the SUT via CORBA.
- It is independent of particular SUT types and has therefore high flexibility.

With the TTCN/CORBA Gateway, the tool-specific adaptation of the test system can be reduced to a minimum. The complete executable test system can be divided into a generic part and a (minimal) specific part. The generic part is the TTCN/CORBA Gateway. The specific part consist of the test cases as defined in TTCN and are executed by use of the generic part. It uses the Gateway as a high-level bridge to test services provided by a particular SUT. In accordance to the TTCN terminology we use subsequently the notion executable test suite (ETS) to refer to the specific part of the test system.

Considering the realization of the Gateway, principally an application has three means to communicate with an CORBA ORB:

1. Using the object specific static stub and skeleton interfaces.
2. Using the dynamic invocation and skeleton interfaces, which are common for all objects.
3. Building an inter-ORB bridge using ORB internal or public APIs (see CORBA 2.2).

The third case involves the handling of low-level transport mechanism directly by the application. This may be essential for performance sensitive operations. However, due to its complexity the third case is currently not used by the Gateway.

The Gateway uses both the first and the second means in its implementation. Details are discussed in the following. Figure 6 illustrates the relationship between four major systems involved in the testing: an underlying interoperable ORB with the supported interfaces and interface repository, a system under test on the right side, the TTCN/CORBA Gateway on the left side, and an executable test suite which is capable to interact with the SUT over the Gateway and the ORB.

Functionally, the Gateway consists of three parts called GatewayMain, GatewayClient and GatewayServer. An object implementation normally contains a client part and a server part. The test system generally plays the role of the

counterpart of the SUT when testing it. In order to test a service of the SUT, the behaviour of a corresponding client is emulated by a tester (a logical component of a test system). To support transformation and handling of requests passed by the tester, the GatewayClient is constructed. In some cases, the counterpart of a client of the SUT need also be emulated by a tester to trigger the SUT into a desired state. Requests from the SUT relayed by the ORB are prepared by the GatewayServer for the corresponding tester. GatewayClient and GatewayServer are also responsible to proceed the replies and exceptions to the appropriate requesters. At run time, multiple instances of GatewayClient and GatewayServer may be created. Each of them is associated with an instance of either a provided or a requested OMG IDL interface of the SUT, which is an object reference in the CORBA context. Additionally, they are all locally managed by the GatewayMain, which provides the interface of the Gateway to the ETS. A set of operations (in the sense of services) are offered by the GatewayMain. For example, to send a message that includes information for a request, a reply or an exception, the GWSend() operation on GatewayMain is called by the ETS. The GWReceive() operation is called by the ETS to inquire whether some message from the ORB is already prepared by GatewayMain.

Technically, the Gateway itself is a full CORBA compliant application. The interfaces of GatewayMain, GatewayClient and GatewayServer are specified in OMG IDL. The internal communication between these objects is carried out by static stubs and skeletons. To achieve the flexibility of the Gateway, requests on SUT server objects via GatewayClient are dispatched to the ORB using the Dynamic Invocation Interface (DII). Analogously, the Dynamic Skeleton Interface (DSI) is used by the GatewayServer implementation. To obtain OMG IDL definitions at run time to determine e.g. the signature of the operation a GatewayClient instance is to request, or a GatewayServer instance is to offer, the CORBA Interface Repository is applied.

Finally, it should be emphasized that the construction of the Gateway supports distributed test components. The behaviour of a test system with parallel components can be specified using Concurrent TTCN. A typical Concurrent TTCN test configuration has a main test component (MTC) and one or more parallel test components (PTCs). MTC and PTCs communicate with each other over coordination points (CPs) with coordination messages (CMs). At the beginning of a test, we may associate each of the components with an instance of GatewayMain. Since GatewayMain objects are CORBA objects, the fundamental distribution transparent communication between the components is already supplied by the ORB. GatewayMain needs to have appropriate functionality to support CPs and CMs. We are working on the implementation details for this feature.

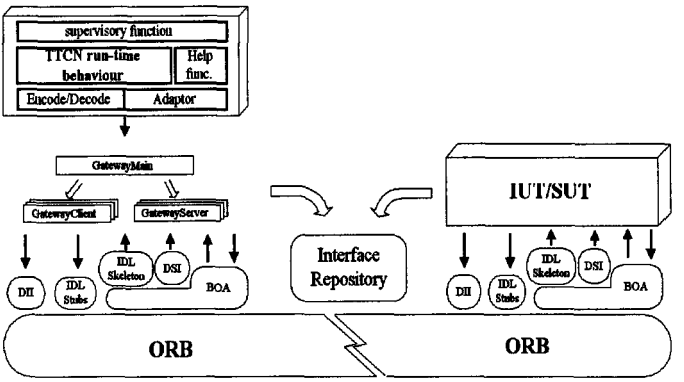

Figure 6: Implementation of TTCN Test Cases

8 Conclusions

This paper discusses testing of service components of TINA services, which is considered as a specific application of the ISO conformance testing methodology. Related work is considered, which identifies the open issue of using TTCN for describing abstract test cases for TINA services as well as to execute them in a CORBA based environment.

Subsequent to that, conformance test objectives for TINA services are identified. We propose to test each interface of a service component separately. The test development is supported by mapping rules of IDL to TTCN, which allow one to generate test skeletons from the IDL specification of an interface. A test case for the service access session gives a practical example for the mapping rules and explains basic interactions between the system under test and the tested system. Subsequent to that, the central idea of the TTCN/CORBA Gateway is presented. It supports the implementation of executable tests from TTCN test cases. With the use of the TTCN/CORBA Gateway, these tests can be executed on any CORBA compliant ORB (with minor efforts for ORB specific adaptation).

In future work, we will elaborate more on coordination features for parallel test components and investigate real-time aspects of the TTCN/CORBA Gateway.

9 References

[1] The Open Group, Inter-domain Management: Specification Translation, preliminary specification, Feb. 1997.
[2] Object-Oriented Concept: Omnibroker manual, version 2.0.2, Dec. 1997.
[3] Thomas J. Mowbray, Ron Zahavi: The Essential CORBA, John Willey & Sons, Inc., 1995.
[4] TINA-C Baseline Document: Service Architecture, Version 5.0. June 1997.
[5] TINA-C Ret Reference Point Specification, RFR/S, Version 0.7, Sept. 1997.

[6] TINA-C Stream Deliverable: Service Component Specification - Computational Model and Dynamics. Version 1.0b, Sept. 1997.
[7] TINA-C Baseline Document: TINA Object Definition Language Manual, Version 2.3. July 1996.
[8] OMG: The Common Object Request Broker Architecture and Specification, Version 2.2. Feb. 1998.
[9] ISO/IEC 10746-2, 10746-3, ITU-T Draft Recommendation X.902, X.903: Basic Reference Model of Open Distributed Processing - Part 2: Foundation, Part 3: Architecture, Feb. 1995.
[10] A. Schill: DCE - das OSF Distributed Computing Environment, Einführung und Grundlagen. Springer Verlag, 1996.
[11] Telelogic: The SOMT Method in Tau 3.2 Methodology Guidelines, Telelogic, Malmö, Oct., 1997
[12] M. Born, A. Hoffmann, M. Winkler, N. Fischbeck, J.Fischer: Towards a Behavioral Description of ODL. - TINA Conference '97, Santiago de Chile, 1997.
[13] J. Grabowski: SDL and MSC Based Test Case Generation - An Overall View of the SAMSTAG Method. - Technical Report IAM-94-005, University of Bern, May 1994.
[14] ISO/IEC 9646: Information technology - Open Systems Interconnection - Conformance testing methodology and framework. 1994.
[15] ISO/IEC 9646: Part 3: The Tree and Tabular Combined Notation (TTCN), Edition 2, Nov. 1997.
[16] L. P. Lima Jr., A. R. Cavali: Test Execution of telecommunications services. - in Proc. of IFIP FMOODS '97, Canterbury UK, 1997.
[17] S. Rao, Ch. BeHanna, M. Sun, J. Forys: CORBA Service Test Environment. - NEC Systems Laboratory, Inc., Apr. 1997.
[18] W. Schütz: On the Testability of Distributed Real-Time Systems. - Report of the ESPRIT Basic Research Project 3092 „Predictably Dependable Computer Systems".
[19] A. Ulrich: Test Case Generation and Test Realization in Distributed Systems. - PhD Thesis (in preparation, in german only), Otto-von-Guericke-University Magdeburg, Germany, Sept. 1997.
[20] TOSCA Deliverable 6: Service Creation: the TOSCA Paradigm and Framework Approach. - AC237/BT/DS/P/019/B1, 1997.