

# DEFINITION AND UTILISATION OF OMG IDL TO TTCN-3 MAPPINGS

Michael Ebner

*Institute for Telematics, University of Lübeck*

*Ratzeburger Allee 160, D-23538 Lübeck, Germany*

*Phone: +49 451 500-3721 Fax: +49 451 500-3722*

*ebner@itm.mu-luebeck.de*

Aihong Yin

*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*

*yin@fokus.gmd.de*

Mang Li

*Department of Computer Science, Ludwig-Maximilians-University Munich*

*Oettingenstr. 67, D-80538 München, Germany*

*Phone: +49 89 2178-2164 Fax: +49 89 2178-2262*

*mang.li@informatik.uni-muenchen.de*

**Abstract** An established middleware technology for Internet-based distributed systems is CORBA, where interfaces are described with IDL. TTCN is a standardised test description language widely used in the telecommunications area. The current version of TTCN, version 3 (TTCN-3), is among others designed to test CORBA-based systems.

This paper presents a definition of the OMG IDL to TTCN-3 mappings, which facilitate the testing of CORBA-based systems. The application of the mappings is shown by an example test for the CORBA Portable Object Adaptor (POA).

**Keywords:** TTCN-3, CORBA, IDL

## 1. Introduction

Nowadays, conformance and functional testing is widely used in the area of telecommunications. Due to the increasing amount of services provided via the

Internet, testing of distributed systems based on the Internet technologies gets more and more important.

The *Common Object Request Broker Architecture* (CORBA) is an established open distributed object-computing infrastructure, standardised by the *Object Management Group* (OMG) (OMG, 2001). The *OMG Interface Definition Language* (IDL) is used to describe object interfaces for CORBA-based systems.

The *Tree and Tabular Combined Notation* (TTCN<sup>1</sup>), a standardised test description language, has been applied to the functional testing of communication protocols for years (ETSI, 2001). The latest widely used version of TTCN is version 2 (TTCN-2). It has been proven that TTCN-2 is generally applicable to the testing of CORBA-based systems (Schieferdecker et al., 1998; Li et al., 1999; Mednongov, 2000; Mednongov et al., 2000). In TTCN-2 and its predecessor, the asynchronous communication mechanism on the message-passing basis is the only supported mechanism. Operation invocations, which are primarily used by CORBA applications, are mapped to inter-related messages.

To provide more adequate support for such synchronous communication, new concepts have been developed for version 3 of TTCN (TTCN-3) (Leach, 2000). Among them new constructs for the description of procedure calls and semantics of procedure-based communication ports were added. Following the approach of specification-based testing, as applied in the studies on TTCN-2, some parts of the IDL to TTCN-2 mappings can be reused, others must be changed to reflect the new concepts. This paper presents the recent work of the authors on this aspect (Ebner, 2001a; Ebner, 2001b; Yin, 2001; Yin et al., 2001).

The remainder of this paper is structured as follows. Firstly, the basic concepts of CORBA, IDL and TTCN-3 are introduced. Secondly, the IDL to TTCN-3 mappings are summarized, with the focus on the new rules. Thirdly, the application of the mappings is illustrated by an example test for the CORBA *Portable Object Adaptor* (POA). Finally, some concluding remarks are given.

## 2. CORBA, IDL and TTCN-3

This section gives a short introduction on CORBA, IDL and TTCN to motivate the rest of the paper.

### 2.1 Common Object Request Broker Architecture

The *Common Object Request Broker Architecture* (CORBA) is a standard architecture for distributed object systems. The heart of CORBA is the *Object Request Broker* (ORB) which is the communication infrastructure for the distributed environment. The ORB provides a mechanism for transparently communicating client requests to target object implementations. It simplifies distributed

<sup>1</sup>TTCN (in the context of TTCN-3) is now defined to mean *Testing and Test Control Notation*.

programming by decoupling the client from the details of the method invocations, and hence, makes client requests appear to be local procedure calls. The ORB consists of the ORB core and some interfaces on top of it. The ORB core provides the basic representation of objects and means for the communication of requests. The technology used in the ORB core is hidden by the public interfaces layered on top of it. They are the IDL Stub and Skeletons which present the language mapping and support the static invocation of requests to objects, the *Dynamic Invocation Interface* (DII) and the *Dynamic Skeleton Interface* (DSI) which allow dynamic creation and invocation of requests to objects at run-time, and the *Interface Repository* (IR) that provides storage of object interface definitions which are accessible by applications at run-time (Figure 1).

The *Portable Object Adapter* (POA) is an important component of the CORBA *Application Program Interface* (API) recently specified by the OMG. The POA provides the facility for flexible management of server objects, e.g. creation of object references, activation of objects and dispatching of requests made on the objects, etc. Each POA is associated with a set of policies. The POA policies describe characteristics of the POA and the server objects in the POA. The POA components are described below.

- **Object Id** is a value used by a POA or a user-supplied implementation to identify an object.
- **Servant** is a programming language object or entity that implements requests on one or more objects.
- **Adapter Object Mapping** (AOM) is logically a key-value pair, with the *key* set to `Object Id` and the *value* set to the `address` of the servant. It depends on the `RETAIN` policy.
- **POA Manager** is an object that encapsulates the processing state of one or more POAs.
- **Servant Manager** is an object that supplies a POA with the ability to activate objects on demand when the POA receives a request directed at an inactive object.

## 2.2 Interface Definition Language

The *Interface Definition Language* (IDL) is a language to *describe* interfaces in an implementation language independent manner and can also be used by other systems than CORBA. It does not support the description of implementation characteristics like behaviour, instances or relationships.

**Data Types.** IDL supports the most basic data types from C++, but there are no *references* in IDL. Instead, the types `string`, `boolean` and `any` are available. Some

data types have a different specification like `char` which is a type of its own in CORBA. The *constructed* types `enum`, `struct`, `array` and `union` are similar to the ones in C++. The *template* type `sequence` is a variable length array of elements of one, but any, IDL type. The type `string` is like `sequence` but it only supports ASCII ISO-Latin characters. Type `any` is like type `Object` in *Java* a placeholder for any possible IDL type.

**Modules and Interfaces.** The main concept behind IDL consists of interfaces. Each interface may contain constants, types, attributes, exceptions and operations for one object. A module is a method to separate name spaces and may contain any IDL construct. Each IDL construct is automatically *public* according to the object orientated concept. (Multiple-) Inheritance is only permitted for interfaces, not for modules. The inheritance structure may contain loops.

**Attributes and Operations.** Each client knows the IDL interface specification of each *object* containing all information about the *object*. Attributes are like variable definitions but they behave like operations in CORBA. Each *read-write* attribute gets a *set-* and a *get-*function and each *read-only* attribute gets a *get-*function. The main part of an interface is based on operations. An operation declaration consists of an operation attribute that specifies the invocation semantics, the type of the operation result, the operation name, a parameter list, optional exceptions and optional context expressions.

Exceptions are especially used to handle errors caused by the network environment like connection failures. The context expression allows the client to transfer context specific information, like security context information for the security service.

### 2.3 Tree and Tabular Combined Notation

The *Tree and Tabular Combined Notation* (TTCN) is the third part of the *Conformance Testing Methodology and Framework* (CTMF) standard for the specification of test suites for conformance testing. In May 2001, the new version of TTCN, called TTCN-3, was finally standardised (ETSI, 2001).

TTCN is designed for functional, black-box testing and to describe *Abstract Test Suites* (ATS) which are independent of a concrete test platform. Therefore, special interfaces defined by TTCN between a *System Under Test* (SUT) and the ATS are required to make a test suite executable. First, there has to be an *Abstract Test System Interface* (ATSI) which defines the sight of the ATS upon the SUT. The access points between ATS and SUT are called *Point of Control and Observation* (PCO). Secondly, there is a *Real Test System Interface* required which maps the ATSI to the SUT (Figure 1).

Test configuration in TTCN is done by the *Main Test Component* (MTC) which controls all other test components called *Parallel Test Components* (PTCs). PTCs

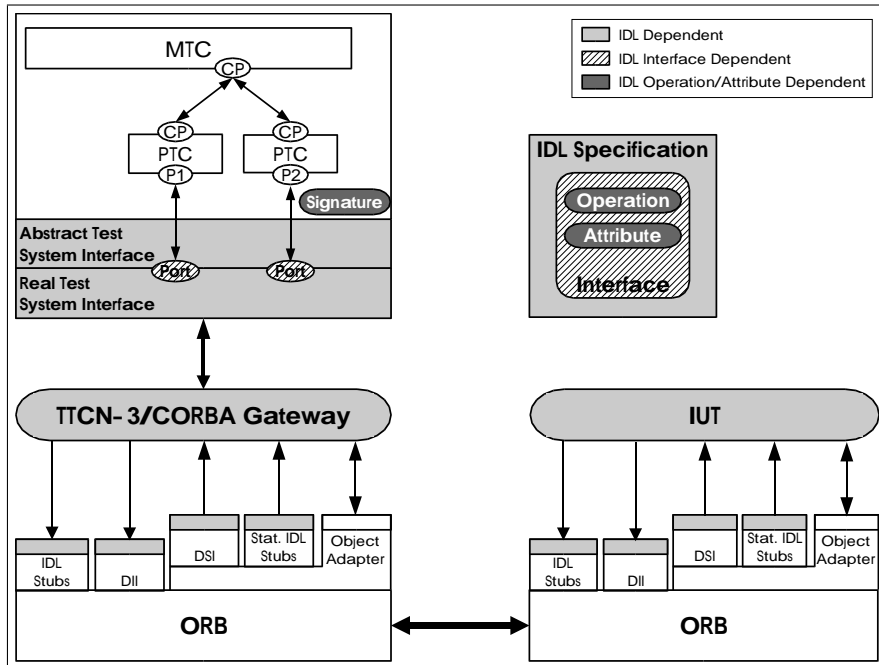


Figure 1. An architecture to use TTCN-3 for testing CORBA-based systems. The upper left corner describes the TTCN-3 part, the upper right corner the interface specifications by IDL, the middle part shows the gateway between TTCN-3 and CORBA and the Implementation Under Test (IUT). The bottom part represents the underlying CORBA system. All parts which depend on the IDL specification are marked (see legend).

can be dynamically created whereas the MTC is created automatically at each test case execution. Test components in TTCN-3 communicate with each other via ports (in TTCN-2 via *Communication Points* (CP)), which are modelled as infinite FIFO queues to store incoming calls. Communication between test components and the test system is also done via ports (in TTCN-2 via PCOs).

TTCN-3 improves concepts of TTCN-2 and introduces new concepts to be a test description language for reactive system tests over a variety of communication platforms such as CORBA -based platforms. An important feature of TTCN-3 is the enhanced communication concept which now supports procedure-based communication to provide synchronous communication, as well as the message-based communication which is asynchronous. In addition, a test execution control part, a module and grouping concept and new data types, are introduced to provide a better control and grouping mechanism.

If TTCN-3 is used for testing systems with interfaces specified by IDL this interface definition can be used as ATSI. Therefore, the mapping suggestion in the next section can be used to generate the *static* ATS parts automatically. This

would effect definitions like data types and signatures for procedures. Hence, interface modifications could be seamlessly introduced into the *static* part of TTCN-3 test suites which would improve consistence and allow simplified testing of CORBA-based systems.

### 3. Mapping of OMG IDL to TTCN-3

The definition of the IDL to TTCN-3 mapping rules is to allow direct use of IDL types and values of the CORBA-based systems in the specification of their tests. The major aspects presented in this section are based on some recent work done by the authors (Ebner, 2001a; Ebner, 2001b; Yin, 2001; Yin et al., 2001).

#### 3.1 Approach

Two different approaches can be followed: either using the *implicit* or the *explicit* mapping. The *implicit* mapping makes use of the *import* mechanism of TTCN-3, denoted by the keywords `language` and `import`. It facilitates the immediate use of data specified in other languages. Therefore, the definition of a specific data interface for each of these languages is required. Currently, ASN.1 data can be used besides the native TTCN-3 types. The data interface for IDL types and values is a topic of on-going research.

The work presented in this paper follows the approach of *explicit* mapping, i.e. IDL data are translated into appropriate TTCN-3 data. And only those TTCN-3 data are further used in the test specification.

#### 3.2 Data Types

Mapping IDL basic data types to TTCN-3 data types is straightforward, because IDL data types are similar to ASN.1 data types which are also used in TTCN-2 (Open Group, 2000; ITU-T, 1997). For example, the *boolean* and *enumeration* types are identical in IDL and TTCN-3. However, TTCN-3 provides more predefined types than TTCN-2, for which a better mapping can be constructed (Table 1). Nevertheless, the *fixed* type and the *constructed* types *union* and *any* require special treatment to get mapped properly. The IDL type *native* is not taken into consideration because of its local semantics.

For the specification of object interfaces IDL type `object` is used. The interface types for application objects inherit from `object`. Interface definitions contain structural information which has to be considered in the TTCN-3 configuration architecture and by the TTCN-3/CORBA gateway. Furthermore, object references associated with instances of interface implementations can be passed over operation invocations. Therefore, `object` is generally mapped to `address` in TTCN-3. In contrary to type `interface`, the IDL type `value` has local operations that are not used outside the object, and are therefore not relevant from the functional testing point of view. However, since the public attributes of `value` instances

Table 1. Mapping list for some data types

OMG IDL	TTCN-2/ASN.1	TTCN-3
float, double, long double		float
char	GraphicString, IA5String(SIZE(1))	char
wchar	GraphicString, BMPString(SIZE(1))	universal char
string	GraphicString, IA5String	charstring
wstring	GraphicString, BMPString	universal charstring
struct	SEQUENCE	record
sequence (bound) <sup>2</sup>	SEQUENCE SIZE(n) OF	record of (with length restriction)
sequence (unbound)	SEQUENCE OF	record of
Object	IA5String, ObjectInstance	address
exception	SEQUENCE	record

are used to communicate object states, we suggest to map the IDL `value` types to `record` types in TTCN-3. Please refer to the remainder of this section for further details.

**Type Extension.** To enhance readability and to provide a clear distinction, mapped IDL data types under TTCN-3 get the prefix `CORBA_`. Additionally, to get the same semantic meaning under TTCN-3 as given by IDL, the types get an `extension` attribute containing the source IDL type name. Therefore, a compiler or interpreter can detect the IDL dependency and can react accordingly like executing semantic checks. This includes encoding, too. This feature provides a flexible way of extending the TTCN-3 type system. For instance, the IDL float types `float` and `double` have a range limitation whereas the TTCN-3 `float` type has no such restriction. Thus, a correct mapping would be difficult but the type system extension solves this problem. The same can be done for the integer types. Although TTCN supports range limitation for integers, we suggest to use the `extension` attribute only as the example below illustrates.

```
type integer CORBA_Short with { extension "CORBA v2.4, IDL type: short" };
type float CORBA_Double with { extension "CORBA v2.4, IDL type: double" };
```

<sup>2</sup>Bound sequences could be matched with arrays in TTCN-3, however, the use of type `record of` with length restriction is preferred to handle bound and unbound sequences equally.

**Any Type.** In IDL it is possible to express each possible IDL type with the `any` type. There is no corresponding type currently in TTCN-3 available. Therefore, another construct is required. One possible way could be to use a `union` type to store all possible data types. The `union` type comprises all required types, whereas these types have to be known in advance. In order to avoid this restriction a `union` type for all possible types in TTCN-3 used by the IDL mapping rules could be defined. However, it is only possible to use basic data types and well defined constructed types wherefore no generic constructed types for `set`, `record`, `union`, etc. are supported. For instance, it is not possible to provide entries for all possible kinds of type `set` by using a generic `set` type. Hence, the user could define his own *restricted* `any` type in TTCN-3 by using a `union` type containing only all possible types of the concrete application and not all thinkable types. This new `any` type has to be mapped to a full `any` type by the test system. However, this requires a careful handling of `any` types because some type definitions could be missing.

The mapping for TTCN-2 is bound to ASN.1 and therefore, it encounters problems in distinguishing types which are mapped onto the same ASN.1 type (Mednogoov et al., 2000). There is no support of the `any` type given but the use of the design pattern *decorator* is suggested if `any` type is used. Another mapping provides only basic types for the `any` type and leaves structured types open for further study (Li et al., 1999). The use of type extension as described before is not appropriate to solve the `any` type problem, neither. This is because of the TTCN-3 type handling which makes it impossible to handle types which are not known beforehand.

Since data types used by the test system are usually already known at compile time, as it is now the case in TTCN-3, we suggest to use a `union` type for data that are communicated to/over CORBA by the IDL `any` type. The `union` type must cover all data types, either basic or derived, that are translated from the IDL specification used by the test system.

**Union Type.** In IDL, *unions* are discriminated to determine the actual type. Therefore, a `record` type is used, which contains two members. The first one stores the discriminator information using an enumeration type. And the second member is of the TTCN-3 `union` type, according to the specified IDL `union` members.

**Fixed Type.** The `fixed` type represents a fixed-point decimal number. There is no corresponding type for `fixed` type in TTCN-3 available. Therefore, a new type has to be created or an existing type has to be used. If we use a `float` with an extension attribute containing the digit and scale the user cannot use this information in his program. Because of the similarities between TTCN-3 and C the solution of the C language mapping of IDL (OMG, 1999, section 1.14)

is used. It maps the type `fixed` to a type `struct` which stores the number in a `char` array and the digit and scale number in extra variables. In TTCN-3 it can be realised by a `record` containing a `charstring` to store the number, and two integers for the digit and scale. Hence, the user can access scale and digit, too.

### 3.3 Modules and Interfaces

IDL uses *modules* as main grouping and scoping units. For this, the *module* concept of TTCN-3 is used.

*Interfaces* describe objects with all their access methods by using *operations* or *attributes*. Additionally, interfaces can contain local type definitions like *exceptions* and *constants* which can be used by its operations and attributes. Because of lacking an object model in TTCN-3, the group construct is used to retain the scoping information.

Furthermore, for each interface, a procedure-based `port` type is defined for the test specification. It is associated with signatures translated from attributes and operations of the interface (see also section 3.4). Since an interface can be used in operation parameters to pass object references, an `address` type is also declared in the data part.

### 3.4 Operations, Attributes and Exceptions

*Operations*, *attributes* and *exceptions* can be mapped well to TTCN-3 because synchronous communication was introduced especially for this purpose.

**Exception Declaration.** In IDL, exceptions are used in conjunction with operations to handle exceptional conditions during an operation call. Thus, a special *struct*-like `exception` type is provided which has to be associated with each operation that can trigger this exception. TTCN-3 also supports the use of exceptions with procedure calls by binding it to `signature` definitions. However, it provides no special `exception` type. Hence, exceptions are defined as `struct` by using `record`. TTCN-2 has no support of exceptions. Thus, it is realised by using PCOs with asynchronous communication to get exception information from the test system.

**Attribute.** An attribute is like a *set*- and *get*-operation pair to access a value. If an attribute is marked as `readonly`, the *get*-operation is used only. Therefore, attribute mapping can be done by the operation mapping.

**Operation.** Apart from attributes, operations are the main part of interface definitions in IDL and are used, for instance, in the CORBA scheme as *procedures* which can be called by clients. Procedure calls in general are supported by

Table 2. Mapping rules for interface elements

OMG IDL		TTCN-2/ASN.1	TTCN-3
operation		ASP	signature
attribute		ASP pair	signature pair
	readonly	ASP	signature
raise expression		CHOICE	signature exception option
context expression			additional signature parameter
interface	name space		group
	parameter	IA5String	address
	communication	PCO	port

TTCN-3 by means of synchronous communication operations which are used in combination with ports.

IDL supports an optional `oneway` attribute for operations which implies best-effort invocation semantics without a guarantee of delivery but with a most-once invocation semantics. *Message* or *procedure*-based ports could be used for oneway procedures because both would be a valid mapping from IDL perspective. However, the use of *procedure*-based ports for oneway procedures is recommended because the IDL specification does not guarantee that oneway calls are non-blocking or asynchronous. Furthermore, CORBA implements oneway procedures by synchronous communication, too.

The parameter attributes `in`, `inout` and `out` describe the transmission direction of parameters and can be mapped directly to the communication parameter attributes in TTCN-3 because they have exactly the same semantics.

A `raise` expression specifies all exceptions which can be thrown by an operation. It can be mapped directly to TTCN-3 because it can be indicated by the procedure signature definition by specifying an exception. Nevertheless, each operation can trigger a standard exception.

A `context` expression provides access to local properties of the called operation. These properties consist of a name and a `string` value. The `context` expression can be matched by redefining the operation with the `context` parameters included in the operation parameters (OMG, 2001, sec. 4.6). This is done in a TTCN-2 mapping introducing an additional array parameter (Mednonogov et al., 2000; Mednonogov, 2000). The additional parameter should be of type `sequence` containing a type `struct` for each context parameter. The `struct` itself contains two variables of type `string` for the context name and value.

## 4. Example

An example for TTCN-3 synchronous communication is explained in this section. It is intended to present the TTCN-3 based ATS for CORBA applications by applying the introduced mapping rules.

### 4.1 Scenario

In this Section, we consider the functionality of the POA (Section 2.1). Controlling the association between objects and servants for request processing is a key aspect of server application scalability. Depending on the number of objects an application contains, it might want to use a separate servant for each one, use a single servant for all of them, dynamically supply a servant to associate the `Object Id`, or use a combination of these techniques to best manage its resource.

POA policies are objects used to define the characteristics of a POA in server applications. The selected aspects of POA policies are discussed below, which are often used for the design of request processing. A POA with `RETAIN` and `USE_SERVANT_MANAGER` policies (abbr. `POA_Svt_Act`) is created as a child POA of the root POA to present the behaviour of POA policies. The POA `POA_Svt_Act` has an *Adapter Object Mapping* (AOM) and a servant activator which is a type of the servant manager. The POA configuration is illustrated in Figure 2.

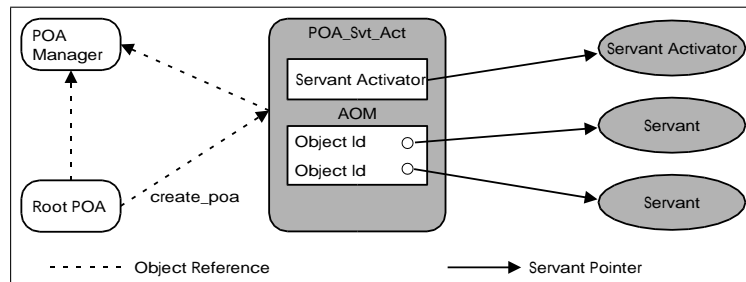


Figure 2. POA Architecture for `POA_Svt_Act`

The servant activator `ServantActivator` is responsible for locating or creating an appropriate servant that corresponds to the `Object Id`. It is used only when an object must be activated during request processing. Therefore, the POA `POA_Svt_Act` has the ability to activate objects on demand.

The behaviour of the POA `POA_Svt_Act` is described in Figure 3, which addresses the case that an object is not active before the request processing. In case the POA `POA_Svt_Act` does not find a servant in the AOM for a given `Object Id`, the servant activator registered with the POA `POA_Svt_Act` is available to determine the servant. It returns the servant that will be used to process the incoming request. The POA `POA_Svt_Act` enters the address of the servant into the AOM so that subsequent requests with the same `Object Id` will be delivered directly to

that servant without invoking the servant activator. Finally, the POA performs the request.

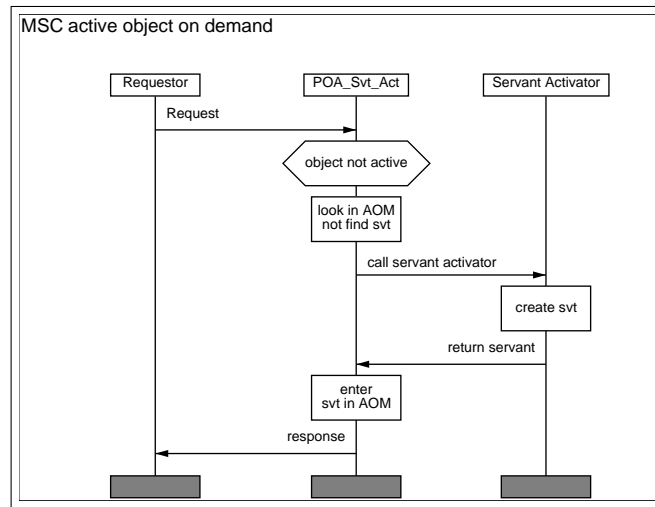


Figure 3. Behaviour of POA\_Svt\_Act

## 4.2 IDL Specification

The IDL specification below is used to test the selected aspects of the POA POA\_Svt\_Act, which are based on the test sequence diagram (Figure 3). The module PolicyTest contains the interfaces RequestPolicyTest and ServantProvider. The interface RequestPolicyTest defines the operations, which can be accessed directly by clients. The interface ServantProvider can only be accessed by clients using the object reference. For instance, this object reference is returned by operations that are defined in the interface RequestPolicyTest (e.g. create\_objectRef).

```

module PolicyTest {
    exception WrongPolicy {};
    enum objectState { active, deactivate, non_aom };

    interface ServantProvider { long increment(in long m); };

    interface RequestPolicyTest {
        objectState check_objectState( in Object obj );
        Object create_objectRef() raises ( WrongPolicy );
    };
};

```

### 4.3 TTCN-3 ATS Specification

The specification of the ATS in TTCN-3 can be divided into a *static* part which includes the type definitions and a *dynamic* part which defines the test behaviour.

**4.3.1 Static Part.** In this part, the IDL data types, operations and interfaces are mapped to TTCN-3 types based on the defined mapping rules.

The Module `PolicyTest` imports definitions from the module `CorbaModule`, which includes the mapping of IDL basic types to TTCN-3 as introduced in Section 3.2. In addition, it contains the derived data type from the IDL specification above.

```

module PolicyTest {
  import all from CorbaModule;

  type record WrongPolicy {} { extension "CORBA v2.4, IDL:exception" };
  type enumerated objectState {active, deactivate, non_aom}
    with { extension "CORBA v2.4, IDL::enumerated" };

  // Mapping of IDL interface to TTCN-3 group, port and address
  group CORBA_INTF_RequestPolicyTest {
    // signatures for operations in the IDL interface RequestPolicyTest
    signature check_objectState (in CORBA_Object svtProvider) return objectState;

    signature create_objectRef() return CORBA_Object
      exception( WrongPolicy );
  }
  type port CORBA_RequestPolicyTest procedure {
    out check_objectState, create_objectRef
  };
  type address CORBA_OBJ_RequestPolicyTest;

  // components for MTC and test system interface
  type component MTCType { port CORBA_RequestPolicyTest MTCpco; };
  type component TSIType { port CORBA_RequestPolicyTest SYSpco; };

  // invocation the operation increment
  external function invoke_increment( in CORBA_Object svtProvider,
    in CORBA_Long m) return CORBA_Boolean;
  :
}

```

The IDL module `PolicyTest` is translated into the TTCN-3 module `PolicyTest` based on the mapping rules. For example, the IDL interface `RequestPolicyTest` is mapped to TTCN-3 port type `CORBA_RequestPolicyTest`. It contains the signatures corresponding operations (e.g. `create_objectRef`) defined in the interface `RequestPolicyTest`. The direction `out` indicates that the operation is called by the test system.

Since the operation `increment` of the interface `ServantProvider` cannot be accessed by the client directly, the external function `invoke_increment` was generated in TTCN-3.

**4.3.2 Dynamic Part.** The dynamic part consists of *test cases* and a *control* part. It is generated manually.

**Test Cases.** The development of test cases can be done by using test sequence diagrams as shown in Figure 3. They are derived from the textual behavioural description in the IDL specification. The test case `createObjectRef` is used to test whether an object reference can be created with the selected POA.

```
testcase createObjectRef( inout CORBA_Object svtProvider ) {
  map( mtc:MTCpco, system:SYSpco );
  MTCpco.call( activate_object )
  {
    MTCpco.getreply ( create_objectRef ) -> value svtProvider
    {
      MTCpco.call( check_objectState:{ svtProvider } )
      {
        [] MTCpco.getreply ( check_objectState value active ) {
          verdict.set( fail ); stop;
        }
        [] MTCpco.getreply ( check_objectState value inactive ) {
          verdict.set( pass ); stop;
        }
      }
    }
  }
  :
}
```

First of all, in this test case, the test system performs the call of the remote operation `create_objectRef` defined in the interface `RequestPolicyTest` at the port `MTCpco` in order to create an object reference. Then, the operation `check_objectState` in the interface `RequestPolicyTest` is called by the test system in order to check the state of the created object.

**Control Part.** In this part, the execution order of test cases is described. The test case `createObjectRef` is used to find out whether an object reference can be created and the state of the object is inactive. If the test cases result is `pass`, the test client makes a request to the created object. This object contains the object reference for the interface `ServantProvider` and operation `increment`. If the request is being fulfilled, the test case `check_createdObj_state` is executed to test whether the created object becomes active.

```
control {
  var boolean is_true;
  var verdicttype theVerdict;
  var CORBA_Object svtProvider := null;

  theVerdict := execute( createObjectRef( svtProvider ), 80E-3 );
  if ( theVerdict == pass ) {
    is_true := invoke_increment( svtProvider, 1 );

    if ( is_true == true ) {
      execute( check_createdObj_state( svtProvider ) );
    }
  }
}
```

## 5. Conclusions

Our work presented in this paper is primarily focused on the definition of a set of OMG IDL to TTCN-3 mapping rules, in order to support adequate testing of CORBA-based systems based on their specifications. The first example tests show that the new concepts of TTCN-3, in particular the support of synchronous communication, apply well to these systems.

The defined mapping rules focus on the translation of type and structural information contained in IDL specifications. They can be integrated into the TTCN-3 data concept, e.g. using a specific data interface for IDL. Until this is approved by the TTCN-3 standardisation, the presented mapping rules can be implemented by translators to achieve semi-automated development of test data specifications. To support automated generation of the dynamic part in an ATS, additional formalised behavioural descriptions, e.g. test sequence diagrams, can be considered.

The limitation of the mapping for the IDL `any` type is intensively discussed in the paper. The proposed solution is already applicable for manual or semi-automated translations of IDL data types. It provides input to the consideration of a native TTCN-3 `any` type or the definition of the IDL data interface.

Future work will also address the realisation of the TTCN-3/CORBA gateway. Because of the integrated support of synchronous communication in TTCN-3, its implementation will be simpler than the one for the TTCN-2/CORBA gateway.

## References

- Ebner, M. (2001a). A Mapping of OMG IDL to TTCN-3. SIIM Technical Report SIIM-TR-A-01-11, Institute for Telematics, Medical University of Lübeck, Germany. Schriftenreihe der Institute für Informatik/Mathematik.
- Ebner, M. (2001b). Mapping CORBA IDL to TTCN-3 based on IDL to TTCN-2 mappings. In *Proceedings of the 11<sup>th</sup> GI/ITG Technical Meeting on Formal Description Techniques for Distributed Systems, Bruchsal, Germany, 21.-22. June 2001*. International University in Germany. <http://www.i-u.de/fbt2001/>.
- ETSI (2001). Methods for Testing and Specification (MTS) — The Tree and Tabular Combined Notation version 3 — Part 1: TTCN-3 Core Language. European Standard ETSI ES 201 873-1, European Telecommunications Standards Institute, Sophia-Antipolis, France.
- ITU-T (1997). Recommendation: Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. International Standard X.680, ITU-T.
- Leach, E. (2000). Enhanced Techniques for CORBA Validation CORVAL2 – Validating Multi-Vendor CORBA Conformance and Interoperability in Heterogeneous Environments. D29 – white paper, The Open Group. European Commission Project Number IST-1999-11131.
- Li, M., Schieferdecker, I., and Rennoch, A. (1999). Testing the TINA Retailer Reference Points. In *4<sup>th</sup> Int. Symposium on Autonomous Decentralized Systems (ISADS'99), Tokyo, Japan, Mar. 1999*.
- Mednonogov, A. (2000). Calypso Gateway specification, version 0.07. Technical report, Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, Finland.

- Mednonogov, A., Kari, H., Martikainen, O., and Malinen, J. (2000). Conformance Testing of CORBA Services using Tree and Tabular Combined Notation. In Ural, H., Probert, R., and Bochmann, G., editors, *Proceedings of the IFIP TC6/WG6.1 13<sup>th</sup> International Conference on Testing of Communicating Systems (TestCom 2000), August 29 – September 1, 2000, Ottawa, Canada*, pages 193–208. IFIP – The International Federation for Information Processing, Kluwer Academic Publishers.
- OMG (1999). C Language Mapping Specification. OMG Formal Document FORMAL/99-07-35, Object Management Group.
- OMG (2001). The Common Object Request Broker — Architecture and Specification. OMG Formal Document FORMAL/2001-02-01, Object Management Group. Version 2.4.2.
- Open Group (2000). Inter-Domain Management: Specification & Interaction Translation. Technical Standard C802, Open Group.
- Schieferdecker, I., Li, M., and Hoffmann, A. (1998). Conformance Testing of TINA Service Components — The TTCN/CORBA Gateway. In Trigila, S., Mullery, A., Campolargo, M., Vanderstraeten, H., and Mampaey, M., editors, *Proceedings of the 5<sup>th</sup> International Conference on Intelligence and Services in Networks, IS&N'98, Antwerp, Belgium, May 25–28, 1998*, volume 1430 of *Lecture Notes in Computer Science*, pages 393–408. Springer.
- Yin, A. (2001). Testing Operation-Based Interfaces — Exemplified for CORBA with ADL and TTCN-3. Diplomarbeit, Telecommunication Network Group, Faculty of Electrical Engineering and Computer Science, Technical University Berlin, Germany.
- Yin, A., Schieferdecker, I., and Li, M. (2001). Mapping of IDL to TTCN-3. Technical report, Fraunhofer Institute for Open Communication Systems (FOKUS), Germany.