TINA Conformance Testing Framework

In response to
the TINA-CAT RFP on
TINA Conformance Testing Framework

Final Version

May 2000

GMD FOKUS

1 Document Information

Final Version of the TINA Conformance Testing Framework RfP

Table of Contents

2 General Description

2.1 Submitters

GMD FOKUS

Competence Center TIP

Kaiserin-Augusta-Allee 31

D-10589 Berlin, Germany

Contact:

Dr. Ina Schieferdecker	**Mang Li**
Tel: +49 30 3463 7241	Tel: +49 30 3463 7101
Fax: +49 30 3463 8241	Fax: +49 30 3463 8101

2.2 Background of the Proposal

TINA combines modern methodologies and techniques, such RM-ODP [11] and CORBA [15] to support the development of large-scale telecommunication systems. TINA provides an open architecture for a multi-vendor environment. The TINA architecture addresses a wide range of issues and provides a complex set of concepts and principles. It has been partitioned into several models, subsystems, components, etc. in order to handle the complexity. An essential partitioning concept is that of reference points (RPs).

Reference points consist of a set of interfaces together with potential interactions at these interfaces. Reference point specifications define conformance requirements for a relationship between or within administrative domains of distributed systems. The TINA reference point concept follows the RM-ODP conformance assessment principles [11]. An example for an inter-domain reference point is the Retailer Reference Point [23].

Key issues for multi-vendor systems is interoperability. Reference points as a collection of conformance requirements are the basis to increase the likelihood for interoperability. Conformance testing is an effective and efficient means to validate the overall functionality of a multi-vendor system. It is a well-established alternative to the otherwise needed many-to-many test setups for individual components and/ or sub-systems to ensure their interoperability.

The Conformance Testing Methodology and Framework (CTMF) [8] is a well accepted technology in the area of protocol testing. CTMF defines test architectures and the test notation TTCN (Tree and Tabular Combined Notation) for the evaluation of capability and behavioral conformance of protocol implementations. CTMF allows the modelling and specification of both centralized and distributed test systems. It has been used for ISDN, ATM and Internet protocols. A recent work shows also the usability of CTMF for object-oriented systems [5].

Reference points provide a simple straight-forward means to express the TINA architecture in terms of objective requirements for conformance. However, current defined TINA reference points tend to be too large. They are inadequately structured and do not allow incremental specification, implementation, and testing.

Therefore, the design for testability of reference points is a requirement to enable and further facilitate the testing process for distributed systems. This motivates the TINA Conformance Testing Framework Request for Proposal (RfP) [25]. This RfP introduces the *facet* concept that is to support more effective and efficient conformance testing of TINA products.

As an initial submission to the RfP, this document elaborates the definition and specification of *reference point facets* (RP-facet)[1], as well as RP-facet based testing.

Reference points can be composed from RP-facets and/or segmented into RP-facets. Each of these RP-facets (or subtopics) has it's own concepts, partitioning, information model, and other details. Conformance can be tested separately for each of these RP-facets. Thus, a system may be tested at multiple levels with respect to various RP-facets representing different aspects of a reference point. This kind of testing is consistent with the current usage of TINA specifications, and allows a vendor to implement limited roles in the business of service provisioning.

1. The term facet is used in the OMG CORBA Component Model (CCM). In order to avoid misunderstandings, RP-facet is used instead.

3 Detailed Description

3.1 RP-Facet Specification Template

3.1.1 RP-Facet Definition

RP-facets define refinements of TINA reference points. A RP-facet is to enable interaction among components with separable concerns. It is a meaningful and self-standing portion of functionality. A RP-facet is a minimal set of conformance criteria, a TINA testing can be associated with. RP-facets are the basis for determining test purposes and generating test cases for TINA reference points.

Each reference point should be composed of one or more RP-facets. Typically, there will be a "core" facet that provides some minimum set of functionality. Additional interfaces and interactions can be specified to provide additional functionality. A RP-facet depends on the presence of the “core” facet and may depend on the presence of other RP-facets.

The RP-facet concepts is to facilitate conformance testing, which is in particular based on the observation of system behavior. Thus, a purpose-oriented functional description in terms of use scenarios is proposed. The functionality of a RP-facet is specified by the signature and behavior of operations[2]. Operations provide services to object’s environment, whereas interfaces represent access points of services.

Typically, an inter-domain TINA reference point separates two business roles with distinguished functionality. A RP-facet is associated with one of the architectural parts separated by the reference point, referred to as *RP-facet role*.

The RP-facet role is to denote the functionality of interest in relation to the corresponding reference point. The dynamic aspect of operations is described by *use scenarios*. The purpose-oriented use scenarios describe potential interactions between the RP-facet role and it’s environment.

Before we define the notion of RP-facet, we need to define miscellaneous notions such as dependent operations and self-containment.

Definition 1: *An interface[3] I has a set of operations SO_I. SO_I is divided into the set of <u>mandatory</u> and <u>optional</u> operations SO_I^{mand} and SO_I^{opt}, resp., referring to the set of operations, which*

2. Operational interfaces are considered currently. The results are directly applicable to event interfaces. Stream interfaces will be considered in a further work.
3. An interface denotes here an interface instance of an interface type, i.e. potentially there are a number of interfaces of the same interface type at a reference point.

need or resp. can be offered by this interface. It holds that $SO_I^{mand} \cap SO_I^{opt} = \emptyset$ and $SO_I^{mand} \cup SO_I^{opt} = SO_I$.

Definition 2: *A reference point R has a set of interfaces SI_R. SI_R is divided into the set of <u>mandatory</u> and <u>optional</u> interfaces SI_R^{mand} and SI_R^{opt}, resp., referring to the set of interfaces, which need or resp. can be offered by this reference point:*

- *$I \in SI_R^{mand}$ iff $SO_I^{mand} \neq \emptyset$*
- *$I \in SI_R^{opt}$ iff $SO_I^{mand} = \emptyset$*

It holds that $SI_R^{mand} \cap SI_R^{opt} = \emptyset$ and $SI_R^{mand} \cup SI_R^{opt} = SI_R$.

Assumption 1: *Subsequently we assume that the set of all operations SO_R of reference point R, i.e $SO_R = \bigcup_{I \in SI_R} SO_I$, is not empty.*

Definition 3: *Let o be an operation at an interface I of reference point R, i.e. $o \in SO_I$. Let $o_1..o_n$ be further operations at R, i.e. $o_i \in SO_R$, i=1..n.*
o is <u>dependent</u> on $o_1..o_n$ if the invocation of o requires previous invocations of $o_1..o_n$.[4]
o is <u>independent</u> if for all n there is no sequence of operations $o_1..o_n$, on which o is dependent.

Definition 4: *The dependent operations are either specified explicitly or derived from the use scenarios of the reference point.*

Definition 5: *The <u>dependence relation $dep_{I,R}$</u> $\subseteq SO_I \times \wp(SO_R)$ of operations at interface I, where $\wp(SO_R)$ denotes the powerset of all operations of reference point R is defined such that $\forall o \in SO_I \; \forall so = \{o_1..o_n\} \in \wp(SO_R)$: $(o, so) \in dep_{I,R}$ iff*

- *o is dependent on $o_1..o_n$ and*
- *$\forall o_k \in SO_R$: if o is dependent on o_k then $o_k \in so$.*

Lemma 1:
- *$\forall o \in SO_I : \exists!(o, \{o_1..o_n\}) \in dep_{I,R}$, i.e. $(o, \{o_1..o_n\})$ in $dep_{I,R}$ is unique.*
- *o is an independent operation iff $(o, \emptyset) \in dep_{I,R}$.*

A RP-facet is self-contained in terms of functionality. Self-containment is defined with respect to the dependence relation. It is the core property of a RP-facet. Please note that the set of operations of an interface may be used only partially in a RP-facet:

4. The dependence relation can be further refined to cover further aspects of dependencies. For example, if operation `o2` is only executable when operation `o1` returns `x`, then `o2` can be defined to be result-dependent on `o1`. Or, if interface `iB` is only reachable through an operation `o1` of interface `iA`, then `o2` can be defined to be reachable-dependent on `o1`.

Definition 6: *A <u>RP-facet</u> F_R is a set of operations of a reference point with the following properties:*

- *it is a non-empty set and*
- $\forall I \in SI_R\ \forall o \in SO_I\ \forall (o, \{o_1..o_n\}) \in dep_{I,R}$ *: if* $o \in F_R$ *then* $o_i \in F_R$*, i=1..n.*
  (the self-containment property)

The set of all RP-facets is denoted by SF_R*.*

Assumption 2: *Subsequently, we assume that* SO_R *is self-contained.*

Within the same reference point, dependent operations are captured by the same RP-facet:

Lemma 2: $\forall I,J \in SI_R\ \forall o1 \in SO_I\ \forall o2 \in SO_J$*: if* $o1 \in F_R$ *and o1 is dependent on o2, than* $o2 \in F_R$*.*

RP-facets can be ordered. This order will be used to identify necessary steps in conformance testing:

Definition 7: *The <u>order relation</u>* $\leq$ *on RP-facets uses the subset relation:*

$\forall F1, F2 \in SF_R$*:* $F1 \leq F2$ *iff* $F1 \subseteq F2$*.*

The core is used to denote the mandatory, self-contained subset of a reference point. It is the set of all operations that need to be offered at an reference point in order to have it self-contained with respect to the dependence relations and complete with respect to the mandatory operations. If the core is empty then the complete reference point is an optional one.

Definition 8: *The <u>core</u>* C_R *of a reference point R is the set of all mandatory operations of all mandatory interfaces of R with all their dependent operations, i.e.*

- $\forall I \in SI_R^{mand}\ \forall o \in SO_I^{mand}$*:* $o \in C_R$ *and*
- $\forall o \in C_R, \forall so \in SO_R^{n}: (o, \mathrm{so}) \in dep_{I,R} : so \subseteq C_R$*.*

Assumption 3: *Subsequently, we assume that* C_R *is non-empty.*

Lemma 3:

- C_R *is unique.*
- C_R *is a RP-facet.*
- *For each RP, there exist a partitioning into RP-facets* $F_i \in SF_R$*, i=1..n, such that* $SO_R = \cup_{i=1..n} F_i$ *and* $F_i \cap F_j = \emptyset$ *for* $i \neq j$*.*
- SO_R *is a RP-facet.*
- SO_R *is the maximal element of* $\leq$*, i.e.* $\forall F \in SF_R$*:* $F \leq SO_R$*.*

To support incremental specification, RP-facets are built cohesively, with the core as the origin. This leads to the definition of core-based RP-facets.

Definition 9: *A <u>core-based RP-facet</u>* $F_{R,C}$ *is a RP-facet that contains all operations of the core, i.e.* $C_R \subseteq F_{R,C}$*.The set of all core-based RP-facets is denoted by* $SF_{R,C}$*.*

A core-based RP-facet covers at least the core and possibly additional optional operations.

Lemma 4:

- C_R *is a core-based RP-facet, i.e.* $C_R \in SF_{R,C}$*.*
- SO_R *is a core-based RP-facet, i.e.* $SO_R \in SF_{R,C}$*.*

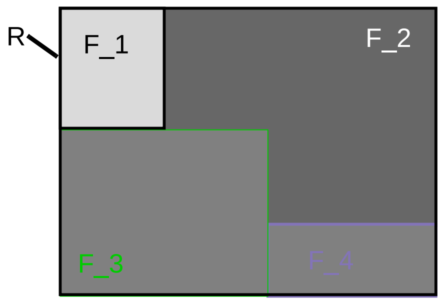
A reference point has a core and may have zero or more additional cohesive core-based RP-facets.

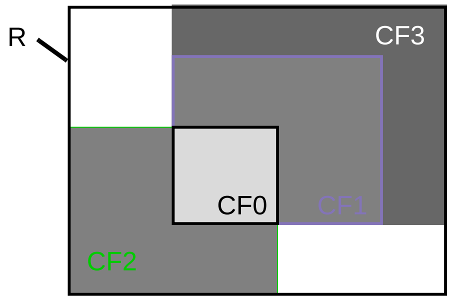
Lemma 5:

- C_R *is the minimal element of the order relation* $\leq$ *on* $SF_{R,C}$*,*
  i.e. $\forall F \in SF_{R,C}$: $C_R \leq F$*.*
- SO_R *is the maximal element of the order relation* $\leq$ *on* $SF_{R,C}$*,*
  i.e. $\forall F \in SF_{R,C}$: $F \leq SO_R$*.*
- *If* $C_R = SO_R$ *then is* $SF_{R,C}$ *a singleton.*

RP-facets can partition a reference point into non-overlapping portions. Such portions are called in [17] segments. A segment captures a subset of the reference point's functionality. Segments of a reference point may be interrelated.

The relation of partitioning RP-facets and core-based RP-facets are depicted in Figure 1.





Partitioning of R into RP-Facets F_1 .. F_4 with SO_R= F_1 ∪ F_2 ∪ F_3 ∪ F_4 and F_1 ∩ F_2 =∅, etc.

Hierarchies of core-based RP-facets CF0 .. CF3 of R with CF0 ≤ CF1 ≤ CF3 ≤ SO_R and CF0 ≤ CF2 ≤ SO_R.

Figure 1 Reference Points and its RP-Facets

The conformance test method (see Section 3.2) will be based on the concept of core-based RP-facets and their hierarchies, as they naturally reflect the mandatory and optional requirements for a reference point and their relation.

3.1.2 RP-Facet Specification

Making the RP-facet concept practical is essential for real, industrial relevant systems. This is possible by providing a development method for RP-facets in combination with appropriate specification techniques. Even more, the unambiguous specification of RP-facet including its static and dynamic models is crucial for testability. As any formalization reduces misinterpretation of the system under discourse, a formal specification supports in particular automated test generation and the possibility to validate tests for their soundness against the specification.

The reuse of specification parts of the reference point under test and therefore the reuse of specification techniques for distributed system is desired as it makes test development more efficient and allows a better integration of system development with test development.

Our approach for specifying RP-facets is based on the Object Definition Language (ODL [13]) for signatures of RP-facets in combination with Message Sequence Charts (MSC [12])[5].

ODL has been selected not only due to its mandatory use within TINA specifications. Furthermore. the structural information in ODL specification is used as a basis for the test configuration. Additions to ODL are needed to cover specific aspects of RP-facets according to the concepts introduced in the previous section. The specification template for RP-facets comprises:

- indication to the related reference point and the RP-facet role,
- statical specification of the RP-facet in ODL, and
- behavioral specification of the RP-facet in terms of use scenarios, including representations of dependence relations in MSC.

MSC presents the communication behaviour in a very intuitive and transparent manner, particularly in the graphical representation. The MSC-language is easy to learn, use and interpret. In connection with other languages it can be used to support methodologies for system specification, design, simulation, testing, and documentation. Due to its wide acceptance and practicability, MSC is a good candidate to be used for the behavior specification of objects defined in ODL.

Furthermore, we use the standard test notation TTCN (Tree and Tabular Combined Notation) to formulate test cases for RP-facets. TTCN has been selected as it is the only standardized test notation within telecommunication, it is accepted by both ISO and ITU, official test suites are written in TTCN and there is a growing number of standardized, internationally recognized, and publicly available test

5. We concentrate currently on the on the functional aspect in the behavioral specification of reference points. Extensions to support description of operational aspects, e.g. QoS, usage, will be elaborated in future work.

suites defined in TTCN. In addition, there is a short step between specification and test execution. Finally, test suites in TTCN can be easily maintained, flexibly modified and extended

Short overviews on the notations used for the RP-facet template are given in the appendices of this document. Further details can be found in tutorials on ASN.1 [1], MSC [14] and TTCN [28].

The RP-facet specification and test case generation cycle is presented in Figure 2. ASN.1 is the commonly used data representation form by MSC and TTCN. Thus, mappings for data types and constants from ODL to ASN.1 need to be defined.

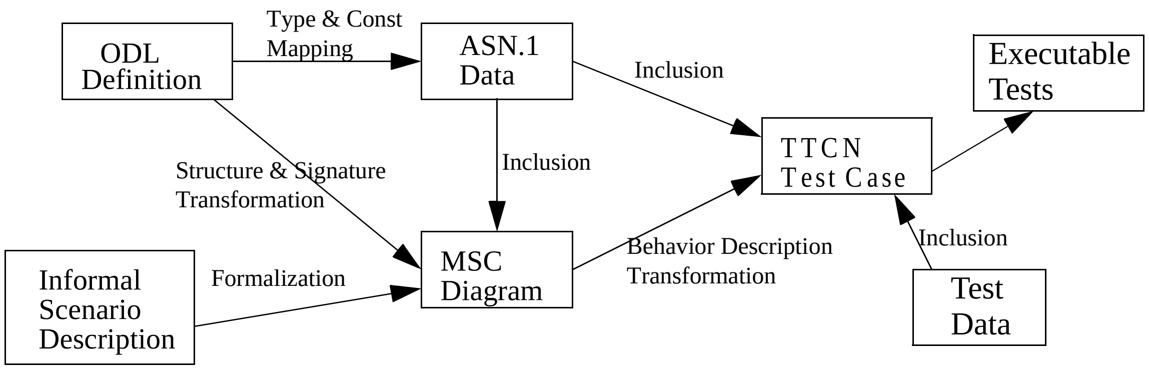


Figure 2 RP-facet specification and test generation

3.1.2.1 Structural Specification Template

The template for the RP-facet structural specification is an extension of the TINA reference point specification template [22], which uses TINA-ODL [24], referred to as ODL in the following.

ODL is a superset of the OMG IDL (abbr. as IDL). In fact, most of the current TINA reference points are specified using IDL only. A reference point specification is a collection of interface signatures. A naming convention is used to indicate the business role that provides an interface, for example, an interface named using the prefix *TINARetRetailer* is to be supported by a TINA retailer stakeholder. In addition, IDL does not have notions to specify which interfaces are provided by a business role and which interfaces are required by it. The ODL's object template provides a notion for this multiple interfaces. Therefore, the following usage of ODL is proposed:

- Each business role separated by a inter-domain reference point is represented by a *CO* (Computational Object) in ODL.
- Interfaces provided by a business role are indicated using the *supports* notion.

- Interfaces used by a business role are indicated using the *requires* notion.

Other interface relevant signature definitions do not change.

3.1.2.2 ODL to ASN.1 Data Mapping

The ODL to ASN.1 mappings for data types and constants are in-line with the rules defined in [5][6]. Rules for basic type translation are shown in Table 1. Structure types are mapped according to Table 2.

ODL Types	ASN.1 Type
long, unsigned long, long long, unsigned long long, short, unsigned short	INTEGER
char, wchar, string, wstring	GraphicString
octet	OCTET STRING(SIZE(1))
boolean	BOOLEAN
void	NULL
float, double, long double	Real

Table 1 Mapping rules for basic types

ODL Type	ASN.1 Type
struct	SEQUENCE
sequence	SEQUENCE OF
enum	ENUMERATED
array	SEQUENCE SIZE(n) OF
any	CHOICE
union	SEQUENCE

Table 2 Mapping rules for structured types

ODL *exception* declarations are *struct*-like. Hence, they are mapped to ASN.1 *SEQUENCE* types.

The mapping for the *Object* type is aligned to the OMG *interoperable object reference* (IOR) concept. An IOR is the global representation of the corresponding object and is composed of ASCII characters. For systems that are compliant with this concept, ASN.1 *IA5String* type is used.

3.1.2.3 Behavioral Specification Template

Message Sequence Charts (MSC) is a graphical and formal trace language defined by ITU-T [12]. MSC describes interactions between message-passing instances. MSC-2000 [12] is a new version of the standard that has been approved only recently. It has improved structural, data and time concepts. Method calls are introduced to support the description of control flows.

To use MSC for use scenarios of RP-facets, some structure and signature transformations are required.

6. This work is based on CORBA 2.2 specification. Mappings for IDL types included in the most recent and up-coming CORBA specifications, e.g. the *value* type, will be considered in future work.

Rule 1 *MSC diagrams for a RP-facet are organized by a MSC document. The identifier of the MSC document is equivalent to the name of the RP-facet.*

A MSC document defines an instance kind for a RP-facet. It contains instances, messages, timer and MSC diagram declarations. In addition, a data language can be declared.

Rule 2 *The RP-facet role, the environment of the RP-facet role, every supported/required interfaces are mapped to separate instances.*

Instances for the RP-facet role and supported interfaces form the scope of the RP-facet, while other instances represent the scope of the environment. Interface instances play the role of service supplier. Instances of the RP-facet role and its environment are of the service consuming role.

Rule 3 *The order relation between RP-facets, e.g. F1 ≤ F2, is represented by inheriting the MSC document for F1 into the MSC document for F2.*

Inheriting a MSC document into another results in inheriting all declarations and MSCs from the inherited into the inheriting MSC document. This reflects the idea that for *F1 ≤ F2, F2* covers *F1* completely as it is.

Rule 4 *The dependence relation is represented by MSC expressions or high-level MSC (HMSC), where the MSC sequential operator is used to order the individual operation invocations as a sequence of simple MSCs reflecting separate operation invocations and the MSC alternative operator for the subsequent behaviour in accordance to the potential outcomes of operation invocations.*

Rule 5 *MSC specifications for RP-facets consists of two diagram types:*

- *High-level MSCs (HMSC) give an overview on the main structure and dependencies at the RP-facet. Here, references to further MSCs (usually simple MSC, see below), which are typically executed sequentially and combined with guards, are used. Enhanced use scenarios of RP-facets contain also parallel interactions at different interfaces. The operands of parallel expressions are represented by separate MSC instances.*
- *Simple MSCs contain a detailed definition of allowed message exchange and timer events between involved MSC instances. Further they allow the usage of constructors for behavior control (e.g. alternatives, loops etc.) and guarded executions.*

MSC expressions, which can be graphically represented by HMSC, are the basic concept to represent the dependence relation between operations. If *o1* needs to be invoked before *o2,* it will be represented by *M1 seq M2* with *M1* reflecting the invocation of *o1* and *M2* the invocation of *o2*. In the case that several outcomes of *o1* and/or *o2* are possible, within *M1* and/or *M2* the alternative operator *alt* in combination with conditions will be used in addition. Please note that more complex behavior definitions for RP-facets will use also parallel, loop and optional expressions.

Rule 6 *An ODL operation declaration is transformed to MSC message declarations. Mandatory is a message corresponding to request on the operation. If the operation is not a "oneway" operation, a message in accordance with reply on the operation is also defined. If appropriate, each potential exceptional outcome of the operation is translated into a separate message.*

MSC asynchronous messages are selected rather than method calls to allow the representation of alternative operation invocation outcomes, in particular under exceptional conditions.

The rule for attribute transformation is defined analogously:

Rule 7 *An ODL attribute declaration is transformed to MSC message declarations. Mandatory is a message corresponding to the "get" operation on the attribute. If the attribute is not a "readonly" attributed, a message in accordance with the "set" operation on the attribute is also defined.*

The data concepts of MSC-2000 allows the flexible use of a data language of user's choice. No MSC specific data language is defined. MSC-2000 provides syntactical and semantical functions as interfaces to the use of external data languages within MSC.

3.2 Test Method

The RP-facet concepts, in particular the self-containment property of a RP-facet, allow system evolution by incremental specification and implementation. In addition, RP-facets provide also testable specifications:

- The identification of the *RP-facet role* and its communication parties yields to the definition of the scope of the System Under Test (SUT), as well as the environment of the SUT which will be emulated by components of the Test System (TS).
- The *structural specification* of the RP-facet to be tested, in form of ODL and ASN.1 definitions, can be shared by the TS.
- The formalization of *behavioral description* of RP-facet use scenarios in MSC supports an automated generation of tests.

- The *self-containment* property of RP-facets supports the identification of well-defined states of the SUT to achieve reproducible test results.
- The operation dependencies of a RP-facet define requirement on the sequence of test execution.

In order to support an efficient test development, we propose to use abstract test specifications. Our approach is based on the standard test notation TTCN [8].

3.2.1 Test Specification

TTCN (Tree and Tabular Combined Notation) was designed for conformance testing of OSI protocol implementations [8]. The test architecture is based on an asynchronous communication between SUT and TS. PCO (Point of Control and Observation) is an abstract location, where stimuli are sent to the SUT and reactions of the SUT are observed, either in form of Protocol Data Units (PDUs) or Abstract Service Primitives (ASPs). In decentralized test architectures, where typically several Parallel Test Components (PTCs) in addition to the Main Test Component (MTC) communicate with the SUT, more than one PCOs can be assigned to a PTC.

The analogy to the asynchronous message passing mechanism of MSC facilitates the transformation of MSC constructs to TTCN constructs. At first, it leads to the representation of MSC messages as TTCN ASPs[7]:

Rule A *MSC messages representing ODL operations or attributes are translated into TTCN ASPs.*

According to Rule 6 and Rule 7, the ASPs are denoted by *request-ASP*, *reply-ASP* and *exception-ASP*.

Further, PCOs, test components and test configurations need to be identified for the test system. Due to the distinction of *supported* and *required* interfaces, two classes of PCOs can be derived from MSC instances:

Rule B *A MSC instance representing a provided interface of the RP-facet role is interpreted by a client-PCO over which request-ASPs are sent to the SUT and reply-ASPs or exception-ASPs from the SUT are observed.*

Rule C *A MSC instance representing a required interface of the RP-facet role is interpreted by a server-PCO over which request-ASPs from the SUT are received and reply-ASPs or exception-ASPs to the SUT are sent.*

7. The selection of ASPs instead of PDUs is based on the analogies between the object model and the OSI reference model. Please refer to [5] for details.

The assignment of one PCO to one PTC is not stringent, but recommended. The semantic of a PTC is constrained by the class of PCOs it has. A PTC is in a client role when it communicates via a client-PCO with the SUT, and vice versa. Hence:

Rule D *Only PCOs of the same class, i.e. either client-PCOs or server-PCOs, can be assigned to a PTC. The assignment of more than one PCOs to a PTC is allowed, as long as the processing of test events, e.g. parallel sending of ASPs, is not restricted.*

The MSC inline expressions allow behavioral composition of event structures within a MSC. The operators refer to alternative (*alt*), parallel composition (*par*), iteration (*loop*), exception (*exc*) and optional (*opt*) parts. The *alt* operator, used in the example in Figure 4, defines alternative executions of MSC sections. In TTCN, the distinction between sequentialized and alternative behavior is identified by the indentation level of TTCN statements (subsequent TTCN events have a higher indentation as preceding events). Therefore:

Rule E *MSC inline expressions are expressed in TTCN by a combination of appropriate indentation levels, TTCN conditions and GOTO-statements.*

The TTCN timer concept addressing start, time-out and cancellation of timers is sufficiently to cover MSC timer events.

The derivation of HMSCs to TTCN test descriptions appears straight-forward:

Rule F *MSC references are mapped in TTCN to test step calls. MSC conditions are directly interpreted by TTCN qualifiers.*

TTCN test steps are a macro-like kind of subroutines. They are also used in case of RP-facet specifications representing extensions of previously specified smaller RP-facets. For example, it is typical that the test specification derived from a small RP-facet specification (e.g. from the minimal core-based RP-facet) will become the preamble (i.e. the very first test behavior at the beginning of a test description) of another “bigger” RP-facet test specification.

3.2.2 Test Campaign Derivation

In general, software testing is time and cost intensive, i.e. critical for large systems. Therefore, CTMF [7] gives advise for practical test purpose identification and for the grouping of test cases. We define a test suite structure according to core-based RP-facet hierarchies of the reference points under test. The sequence of test execution for the reference points under test is derived from the dependence relation between its operations.

The basic idea is to start with testing the core of a reference point and then to test incrementally by a repeating selection and testing of small extensions of the set of already tested operations. Each extension should comprise a complete core-based RP-facet.

At first, we define the ordered sequence of RP facets to be tested:

- the minimal core-based RP facets is tested first
- subsequently, other core-based RP-facets are tested according to their hierarchy.

Secondly, we define the sequence of testing operations within a RP facet:

- Independent operations are those that can be tested without any preconditions (i.e. without preambles in the test case body).
- Dependent operations can be tested only of the operations they are depending on have been tested already successfully.

An algorithm for the test method is as follows. For simplicity, we assume that the system under test S realizes reference point R by means of core-based RP-facets $CF_1..CF_n$ with $C_R=CF_1 \leq ... \leq CF_n=SO_R$.

Let T be the set of already tested operations at R. T is divided into T_P and T_F. T_P refers to the set of operations that passed all tests. T_F comprises those operations for which at least one test failed. Further, let I be the set of operations that are not testable as they depend on operations, which failed their tests. Let N be the core-based RP facet under test in the current testing iteration.

Start: $T=\varnothing$, $I=\varnothing$, i=1, $N=CF_i$.

Iteration i:

```
Step I: Select $o \in N$ with $(o, \varnothing) \in dep_{I,R}$ :
    /* independent operations */
    Execute the tests for o .
    If o passes all tests, then $T_P = T_P \cup \{o\}$ else $T_F = T_F \cup \{o\}$.
    In any case, $N=N \setminus \{o\}$

    Repeat until no further operations o with $(o, \varnothing) \in dep_{I,R}$ exist .
    Proceed with Step II.

Step II: Select $o \in N$ with $(o, \{o_1..o_m\}) \in dep_{I,R}$ and $\forall j, j=1..m: o_j \in T$
    /* dependent operations whose preconditional operations have been already tested*/
    If $\exists j=1..m: o_j \in T_F$ , then $I = I \cup \{o\}$.
    Else, execute the tests for o .
    If o passes all tests, then $T_P = T_P \cup \{o\}$ else $T_F = T_F \cup \{o\}$.
    In any case, $N=N \setminus \{o\}$ .
```

```
        Repeat until no further operations o with $((o, \{o_1..o_m\}) \in dep_{I,R}$ and $\forall j, j=1..m: o_j \in T)$ exist .
        Proceed with Step III.

Step III: Select $o \in N$ with $(o, \{o_1..o_m\}) \in dep_{I,R}$ and $\exists j=1..m: o_j \in I$
        /*dependent operations for which not all preconditional operations are tested successfully*/
        Then $I = I \cup \{o\}$ and $N = N \setminus \{o\}$ .

        Repeat until no further operations o with $((o, \{o_1..o_m\}) \in dep_{I,R}$ and $\exists j=1..m: o_j \in I)$ exist .
        Proceed with Step IV.

Step IV: Select $o \in N$ with $(o, \{o_1..o_m\}) \in dep_{I,R}$ and $\exists j=1..m: o_j \in N$
        /*dependent operations with cyclic dependencies*/
        Execute the tests for $o_j$.
        If $o_j$ passes all tests, then $T_P = T_P \cup \{o_j\}$ else $T_F = T_F \cup \{o_j\}$.
        In any case, $N = N \setminus \{o_j\}$ .
        Proceed with Step III.

        Repeat until no further operations o with $(o, \{o_1..o_m\}) \in dep_{I,R}$ and $\exists j=1..m: o_j \in N$ exist .
        Proceed with Step V.

Step V: If N empty and not yet termination, take $i=i+1$, $N = CF_i \setminus (T \cup I)$ and proceed with Step II.
```

Termination: *If $T \cup I = SO_R$ terminate.*

Interface operation tests will comprise static operation header tests as well as dynamic testing of operations semantics. First, the static header tests result from combinations of valid/invalid parameters and test values according to the interface signature and constraints [19]. The other test group which focuses on testing of valid/invalid sequences of operations at RP-facets can be derived using traditional test derivation algorithms well known from e.g. labelled transition systems (LTS [3]) or extended finite state machine (EFSM [18]) based test generation methods implemented in several academic and commercial test derivation tools.

3.3 Test Documentation

Test documentation should be given on two levels:

- on the test procedures to assess the conformance of systems for RP-facets
- on the results of test execution for a concrete system.

The test procedures realized by a TTCN test suites are best described by the TTCN ATS itself. The test documentation can be given in

- TTCN/mp to be readable with a TTCN editor (e.g. ITEX in [21]) or a TTCN browser (e.g. [26]),

- Postscript, PDF, etc. generated from TTCN/mp to be readable with a viewing tool, or
- HTML, XML, etc. generated from TTCN/mp to be readable with a Web browser.

Please note that the test documentation on test procedures is considered to be essential for a wide acceptance of the TINA conformance branding, as for potential customers of the TINA conformance branding the test procedures are transparent and serve as a base for understanding of and confidence in the testing process.

The results of test execution for a concrete system should be presented in two types of test reports: a *System Conformance Test Report* (SCTR) and a *RP-Facet Conformance Test Report* (FCTR). Those test reports should be based on the principles defined in [9]. In addition, the test laboratory may also produce detailed diagnostic trace information to accompany the test reports.

The SCTR provides a summary of the results of the conformance testing of the system under test. A SCTR template is given in Annex E. Furthermore, a SCTR shall be accompanied with FCTRs for each tested RP-facet. A FCTR shall follow the template given in Annex F.

3.4 Testing in Practice

This section introduces some tools for handling specifications and test suites for RP-facets in the selected techniques ODL, ASN.1, MSC and TTCN. Those tools make the proposed conformance testing process feasible for use in a production environment.

A graphical ODL specification tool is Y.SCE [30]. It is a CASE tool developed by GMD FOKUS. This tool provides notations based on the reference model of ODP, for the requirement capturing in the enterprise viewpoint, the specifciation of an information model and the definition of the functional decomposition in the computational viewpoint. It supports the generation of IDL, ODL and C++, as well as the generation of SDL-92 skeletons which can be imported to Telelogic SDT tool [21].

Telelogic also provides tools for MSC, ASN.1 and TTCN, together with the SDL tools in an integrated toolkit, named TAU [21]. It provides editing of MSC diagrams and TTCN test suites, as well as the generation of C-code skeletons from the test suite specification and a run-time environment. Verilog that was just aquired by Telelogic provides as well within ObjectGeode tools for MSC and ASN.1 [15].

The TTCN Toolbox is a product of Danet [27]. It supports the specification, compilation, execution and analysis of abstract and executable test suites written in TTCN.

Furthermore, test equipment vendors, such as HP, Siemens, Tektronix, Motorola, Nokia, etc., provide also editors and validators for TTCN test suites and compilers that produce test equipment specific code from test suite specifications.

The execution of TTCN-based test cases for CORBA-based systems, i.e. also for those TINA platforms that are based on CORBA ORBs, is supported by TCgate and TTman [29], two tools developed recently at GMD FOKUS. TCgate represents a generalized gateway between TTCN-based test systems and CORBA-based systems to be tested. TTman takes the role of a test manager for the setup, configuration and control of tests as well as for test reporting.

3.5 Demonstrate Usability

This section presents an example on how the proposed concepts and specification techniques are applied to the TINA Retailer reference point (Ret-RP)[8] [23]. The retailer is the focus of the consideration.

Using the proposal introduced in Section 3.1.2, the specification is extended with an indication of business roles as well as an indication of supported and required interfaces. As shown below, the business role retailer is represented by the CO *Retailer*. It requires six interfaces from the CO *Consumer*, which may be defined in a separate ODL document. *Retailer* supports five interfaces, among them the *i_RetailerInitial* interface that provides the operation *requestNamedAccess*.

```
#include "TINARet_Consumer.odl"
module TINARet {
 CO Retailer {
   requires
     Consumer::i_ConsumerInitial;
     Consumer::i_ConsumerAccess;
     Consumer::i_ConsumerInvite;
     Consumer::i_ConsumerTerminal;
     Consumer::i_ConsumerAccessSessionInfo;
     Consumer::i_ConsumerSessionInfo;
   supports
     i_RetailerInitial;
     i_RetailerAuthenticate;
     i_RetailerNamedAccess;
     i_RetailerAnonAccess;
     i_DiscoverServicesIterator;

   interface i_RetailerInitial {
       void requestNamedAccess (
         in TINACommonTypes::t_UserId userId,
         in TINACommonTypes::t_UserProperties userProperties,
         out Object namedAccessIR,
         out TINAAccessCommonTypes::t_AccessSessionSecretId asSecretId,
         out TINAAccessCommonTypes::t_AccessSessionId asId
       ) raises ();
     ...};
   interface i_RetailerAccess {...};
```

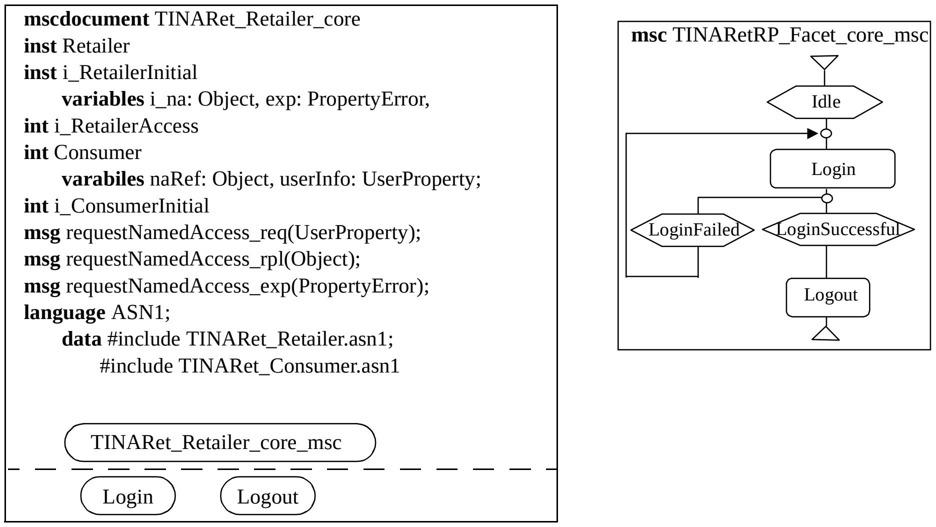
8. For readability reason, the example uses simplified naming and scenarios from the original Ret-RP specification.

```
      ...
    };
  };
```

From the textual description of Ret-RP business scenarios, two RP-facets[9] can be derived according to *Definition 7* (see Section 3.1.1):

- The core facet *TINARet_Retailer_core* involves login and logout of a consumer at the retailer domain. The retailer's interface *i_RetailerInitial* and the operation *requestNamedAccess* are used by login.
- An additional facet *TINARet_Retailer_add1* is based on the core facet. It is to start a service after a successful login, and to terminate the service before the logout.

TINARet_Retailer_core is organized by the MSC document and High-level MSC (HMSC) presented in Figure 3. According to Rule 2, five instances (prefixed by **inst**) are defined: *Retailer*, *i_RetailerInitial*, *i_RetailerAccess*, *Consumer* and *i_ConsumerInitial*. The operation *requestNamedAccess* is mapped to three messages (indicated by **msg**), respectively for the request, reply and exception related to the operation (see Rule 6). The inclusion of data types and constants translated to ASN.1 is enabled by the **language** and **data** constructs. The HMSC *TINARet_Retailer_core_msc* uses two utility MSCs *Login* and *Logout*, and conditions *Idle*, *LoginFailed* and *LoginSuccessful*. It describes the dependency of the logout activity on a successful login.



mscdocument TINARet_Retailer_core
inst Retailer
inst i_RetailerInitial
    variables i_na: Object, exp: PropertyError,
int i_RetailerAccess
int Consumer
    varabiles naRef: Object, userInfo: UserProperty;
int i_ConsumerInitial
msg requestNamedAccess_req(UserProperty);
msg requestNamedAccess_rpl(Object);
msg requestNamedAccess_exp(PropertyError);
language ASN1;
    data #include TINARet_Retailer.asn1;
        #include TINARet_Consumer.asn1

Figure 3 MSC example of TINARet_Retailer_core

Details of purpose-oriented use scenarios of RP-facets are described by MSC event traces. Figure 4 shows the message exchanges between a *Consumer* instance and a *i_RetailerInitial* instance in relation

9. We assume, login and logout functionality are mandatory, while start and termination of services are optional functionality.

to the login activity. The two alternative outcomes of a request on the operation *requestNamedAccess* are represented by use of the MSC inline expression *alt*. Data used in message parameters and/or conditions are defined in the data part of the MSC document.

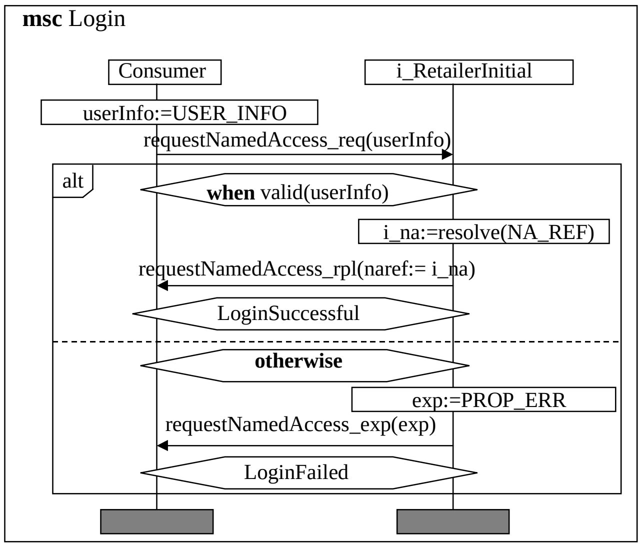


Figure 4 Login MSC diagram

The cohesive relation of *Ret_Retailer_add1* to *Ret_Retailer_core* is in particular reflected by the reuse of declarations and utility MSCs, as presented in Figure 5. To support start service related operation, declarations of the messages *startService_req*, *startService_rpl* and *startService_exp* are added to the core facet MSC document. Furthermore, two new utility MSCs are introduced: *StartService* and *EndService*. *Ret_Retailer_add1_msc* extends the core facet's HMSC by a description of the logical relation between *StartService* and *EndService* MSCs.

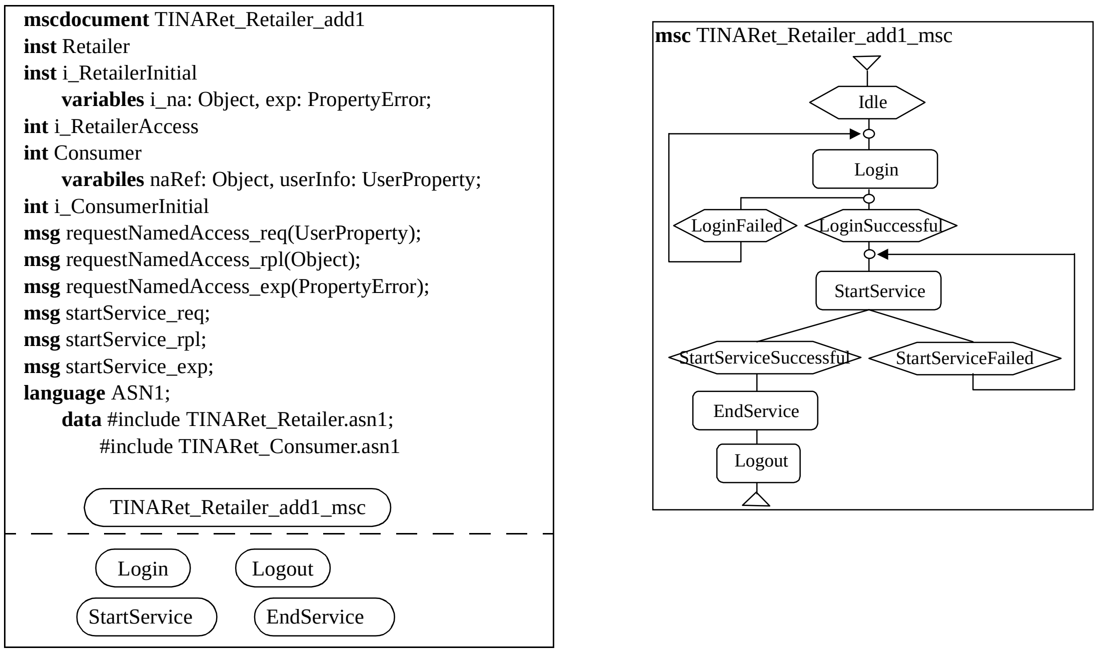

```
mscdocument TINARet_Retailer_add1
inst Retailer
inst i_RetailerInitial
      variables i_na: Object, exp: PropertyError;
int i_RetailerAccess
int Consumer
      varabiles naRef: Object, userInfo: UserProperty;
int i_ConsumerInitial
msg requestNamedAccess_req(UserProperty);
msg requestNamedAccess_rpl(Object);
msg requestNamedAccess_exp(PropertyError);
msg startService_req;
msg startService_rpl;
msg startService_exp;
language ASN1;
      data #include TINARet_Retailer.asn1;
            #include TINARet_Consumer.asn1
```

Figure 5 MSC example of TINARet_Retailer_add1

Table 3 shows the dynamic part of the TTCN specification of a test case in accordance with the Login MSC (Figure 4). The tabular form is simplified to ease the understanding.

In this example, the SUT is an implementation of the retailer domain core RP-facet. This test case is to evaluate the login activity at the retailer's *i_RetailerInitial* interface. The TS emulates the behavior of a client of *i_RetailerInitial*. It uses a client-PCO named *PCO1_i_RetailerInitial* defined using Rule B. The purpose of this test case is to verify: after a request on the operation *requestNamedAccess* with valid user information is sent to the SUT, a reply of *requestNamedAccess* is received by TS (see line 2 and 4). To indicate the ASP kind, the request-ASP is prefixed by *pCALL*, and the reply-ASP is prefixed by *pREPLY*.

The test step *GetInitialRef* (line 1) used as preamble is mainly intended to allow the resolution of object references that will be used in the test case. It is not derived directly from the MSC. It is a general purpose test step. In addition, a timer *Timer1* is used to ensure that a test event (including time-out events) will occur after a given time even in case that the SUT does not answer.

The postamble *Logout* (line 5) after the receive event recalls the MSC reference *Logout* in the HMSC of the core RP-facet.

Test Case Dynamic Behavior				
Nr	**Label**	**Behavior Description**	**Constraints Ref**	**Verdict**
1		+GetInitialRef		
2		PCO1_i_RetailerInitial ! **pCALL_**i_RetailerInitial**__requestNamedAccess**	pCALL_requestNamedAccess_s1	
3		START Timer1		
4		PCO1_i_RetailerInitial ? **pREPLY**_i_RetailerInitial**__requestNamedAccess** CANCEL Timer1	pCALL_requestNamedAccess_r1	(P)
5		+Logout		
6		?TIMEOUT Timer1		I

Table 3 TTCN test case example

The implementation and execution of TTCN-based test cases in a CORBA environment is discussed in [5].

3.6 Accreditation of Testing Vendors

4 Compliance to Evaluation Criteria

The response to the TINA Conformance Testing RfP presents the new concept of reference point facets (RP-facets) to express the architecture and behavior of distributed systems in a formal, detailed and extensible manner. RP-facets are based on the well-established concept of reference points as used in ODP and TINA. RP-facets describe statical and dynamic aspects of reference points as well as pre- and post-conditions for their use. The response gives a mathematical characterization of RP-facets, defines a specification template for the definition of RP-facets and derives a conformance test method for their validation. The basis for conformance testing are constituted by a specific kind of RP-facets - the core-based facets. The conformance testing process makes use of dependencies between RP-facets and the core RP-facet. An example taken from the TINA retailer reference point shows the application and practical use of RP-facets.

The response provides the information requested in RfP Sections 3.1, 3.2, 3.3, 3.4, and 3.5, but not for RfP Section 3.6.

5 Related Standards and Documents

[1] ASN.1 Tutorial: Burton S. Kaliski Jr.: A Layman's Guide to a Subset of ASN.1, BER, and DER. - An RSA Laboratories Technical Note, Revised November 1, 1993, ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc.

[2] R. V. Binder: Testing Object-Oriented Systems, Models, Patterns and Tools, Addison-Wesley, 1999.

[3] E. Brinksma, L. Heerink, and J. Tretmans: Developments in Testing Transition Systems.- Proc. of the Int.Workshop on Testing of Communicating Systems X, 1997.

[4] S. Ghosh, A.P. Mathur: Issues in Testing Distributed Component-Based Systems.- In Proc. of the First Intern. ICSE Workshop on Testing Distributed Component-Based Systems, Los Angeles, U.S.A, May 1999.

[5] M. Li, I. Schieferdecker, A. Rennoch: Testing the TINA Retailer Reference Point, Proceedings of ISADS'99, Tokyo, Japan, March 1999.

[6] M. Li, I. Schieferdecker, A. Rennoch: Formalization and Testing of Reference Point Facets, FMICS'2000, Berlin, Germany, April 2000.

[7] ISO/IEC 9646-2: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 2: Abstract test suite specification, 1991.

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

[9] ISO/IEC 9646-5: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 5: Requirements on Test Laboratories and Clients for the Conformance Assessment Process, 1991.

[10] ITU-T Rec. X.208 | ISO/IEC 8824: 1999 Specification of Abstract Syntax Notation One (ASN.1), Geneva, Swiss.

[11] ITU-T Rec. X.901 | ISO/IEC 10746-1: 1995, Open Distributed Processing - Reference Model Part 1, Geneva, Swiss.

[12] ITU-T Z.120: Message Sequence Charts (MSC'2000), Nov. 1999.

[13] ITU-T Z.130: Object Definition Language (ITU-ODL), March 1999.

[14] MSC Tutorial by E. Rudolph, P. Graubmann, J. Grabowski: Tutorial on Message Sequence Charts.- Proc. of the SDL Forum 1997.

[15] ObjectGeode (MSC Support Tool), http://www.csverilog.com/.

[16] OMG: Common Object Request Broker Architecture (CORBA), version 2.3, 1999.

[17] OMG: Telecommunications Service Access and Subscription, joint revised submission, telecom/00-02-02, Feb. 2000.

[18] A. Petrenko, N.Yevtushenko and G. v. Bochmann: Testing deterministic implementations from nondeterministic FSM specifications.-Proc.of the Int. Workshop on Testing of Communicating Systems IX, 1996.

[19] A. Rennoch, J. de Meer, I. Schieferdecker: Test Data Filtering, 9. GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme", München (D), June 1999.

[20] Steedman, D.: Abstract Syntax Notation One (ASN.1), Technology Appraisals Ltd., 1990.

[21] Tau Tool Set (MSC and TTCN Support), http://www.telelogic.com/

[22] TINA-C: TINA Reference Points, version 3.1, Jun. 1996.

[23] TINA-C: Ret Retailer Reference Point Specification, version 1.1, 1999.

[24] TINA-C: Object Definition Language (TINA-ODL), version 2.3, Jul. 1997.

[25] TINA-C: TINA-CAT WorkGroup Request for Proposals, TINA Conformance Testing Framework, version 1.0, Jul. 1999.

[26] TTCN Browser, http://www.davinici-communications.com.

[27] TTCN-Toolbox, http://www.danet.de/servlet/Danet/browse.en.html -> Products.

[28] TTCN tutorial by Telelogic AB, Web ProForum Tutorial, http://www.webproforum.com/acrobat/ttcn.pdf.

[29] TTman and TCgate (Test Execution Tools for TTCN), http://www.fokus.gmd.de/research/cc/tip/ -> Products&Services.

[30] Y.SCE (ODL Support Tool), http://www.fokus.gmd.de/research/cc/platin/ -> Products&Services.

6 Glossary

ASN.1	Abstract Syntax Notation One
ATS	Abstract Test Suite
ETS	Executable Test Suite
FCTR	RP-Facet Conformance Test Report
ICS	Implementation Conformance Statement
IUT	Implementation under Test
IXIT	Implementation Extra Information for Testing
MSC	Message Sequence Charts
ODL	Object Definition Language
PCO	Point of Control and Observation
Ret-RP	Retailer Reference Point
RP	Reference Point
SCTR	System Conformance Test Report
TTCN	Tree and Tabular Combined Notation
TTCN/mp	TTCN machine processable form
TTCN/gr	TTCN graphical form

Annex A: The Object Definition Language ODL

The Object Definition Language (ODL [24]) defined by the TINA Consortium is a superset of the Interface Definition Language (IDL) defined by the Object Management Group (OMG). It provides means to specify a system structure and relation between objects. This is specified on type level, the so called *templates*. Templates are used to define the types for interface instances, object instances, and for object groups.

ODL enables the computational specification of ubiquitous TINA architectural concepts: *interfaces*, *objects*, and *object groups*. Specifications of these individual concepts are combined to form the (computational) specification of TINA systems. When enterprise, information, engineering and other specifications are added, the initial steps in the TINA software development process are underway.

The ODL syntax is composed of five major parts: *data types and constants*, *stream interface templates*, *operational interface templates*, *object templates*, and *object group templates.* Data type and constant definitions can be made at any point in ODL specifications as long as names are defined before use.

Interfaces can be *operational* (i.e. invocation of operations by a synchronous, discrete communication) or *stream-based* (i.e. asynchronous continuous data flows). Operational interface templates are used to specify procedures in ODL. Stream interface templates are introduced to specify continuous-bit-rate data.

A basic concept in ODL is that of an *object instance*, which might have multiple *interface* instances of multiple interface templates. An object encapsulates its behavior and state, which realize the object fuctionality. The access to and the visibility of the object functionality is realized in a controlled manner via the objects' interfaces. ODL contains no language features for the formal behavior description, textual explanations are used instead.

Object instances are the basic unit of distribution in ODL and are represented by *object templates*. They incorporate stream and operational interface templates.

Objects might be grouped in order to describe the system structure in terms of *object group templates*. Object group templates enable aggregation of object templates to increase the conceptual level at which programs can be designed and increase the modularity of designs.

Annex B: Abstract Syntax Notation One ASN.1

The Abstract Syntax Notation One (ASN.1 [10]) has been defined by CCITT (nowadays ITU) for the definition of the syntax of information data in Open Systems Interconnection (OSI) Systems. The ASN.1 notations can be applied whenever it is necessary to define the abstract syntax of information without constraining in any way how the information is encoded for transmission. It is particularly, but not exclusively, applicable to application layer protocols.

ASN.1 is concerned with preserving the meaning of the information transported in telecommunication systems and, therefore, copes in addition to the data type definitions with the representation of transmitted data, its conversion, encryption and decryption, and compression of data values.

ASN.1 consists of two main parts:

- the description of structured data in a machine-independent way with the abstract syntax notation
- the representation of structured data "on the wire" with the abstract transfer syntax

ASN.1 defines a number of simple and structured data types with optional and mandatory parts and default values. It specifies a notation for referencing these types and for specifying values of these types. A module concept is used to structure data specifications and to reuse existing definitions.

Annex C: Message Sequence Charts MSC

Message Sequence Charts (MSC [12]) by ITU is a specification technique to describe executions of concurrent and distributed systems. It is used throughout the engineering process.

MSCs describe patterns of interaction between a number of independent components of a system. The basic model of interaction is that of *asynchronous* communication by means of *message* passing between the components, which are called *instances*. An MSC describes the order in which interactions and other events take place. MSC diagrams are used to graphically present the pattern of interaction.

MSC supports various constructs to represent the pattern of interaction. Core constructs are *instance*, *message*, *timer*, *condition*, and *inline expression*.

Instances of an MSC represent interacting components of a system. The message flow is presented by horizontal arrows between the interacting instances. The head of the message arrow denotes the message receiving and the opposite end the message sending. The message name is assigned to the arrow. In addition to the message name, message parameters in parentheses may be assigned to a message.

Along each instance axis a total ordering of the described communication events is assumed. Events of different instances are in general unordered. The only order for events of different instances is implied via the interaction with messages: a message must be sent before it is consumed.

A condition describes a state referring to a set of instances of the MSC. Conditions can be used to emphasize important states within an MSC. Conditions are represented by hexagons covering the instances involved.

Composition of event structures may be defined inside of an MSC by means of inline expressions. The operators of inline expressions refer e.g. to alternative (alt), iteration (loop), and optional (opt) regions. In the graphical form of an inline expression, a frame encloses the operands and the dashed lines denote operand separators. The operator keywords are placed in the left upper corner in the graphical representation of the inline expression.

Annex D: Tree and Tabular Combined Notation TTCN

The Tree and Tabular Combined Notation (TTCN [8]) by ITU is an informal test notation that supports specification of abstract test suites. The T for tabular refers to the use of tables (proformas) for the graphical representation of test suites. The T for tree refers to the hierarchical organization of a test suite.

TTCN has been defined as part of the Conformance Testing Methodology and Framework for testing the conformance of OSI protocols. It has been proven that TTCN is applicable in a wider scope such as for ODP, TINA and CORBA systems.

An abstract test suite (ATS) is composed of four parts: an overview part, a declarations part, a constraints part and a dynamic part. The *overview* part provides the test suite structure at a glance. It includes indexes for test groups, test cases, steps, and defaults. It covers information needed for the general presentation and understanding of the test suite
The *declarations* part contains declaration of simple and structured types, constants, variables, test suite operations, PCO (point of control and observation) types, PCOs, timers, PDU (protocol data unit) types and ASP (abstract service primitive) types.

In the *constraints* part, values and value ranges are specified for PDUs and ASPs that are used to describe test events. ASN.1 is used for both type specification and value specification.

The *dynamic* part deals with the specification of dynamic behavior in the form of abstract test cases. Each test case addresses a particular test purpose, i.e. a single conformance requirement. A test purpose is a prose description of a well defined objective of testing, focusing on a single conformance requirement as specified in the appropriate system specification. An abstract test case is a complete and independent specification of those test actions required to achieve a specific test purpose. Test cases can be organized in nested test groups. In addition, from subdivisions of a test case that can be reused by other test cases, a so-called test step library can be built. This library may also have a nested structure.

The behavior specification consists of test events (e.g. send, receive, timeout, otherwise) and other TTCN constructs (e.g. goto, repeat). The test events are organized in a tree form, with the levels represented by indentations.

In some cases, testing is only possible when the system is separated in concurrent components. Concurrent TTCN allows the specification of multi-party testing provided by concurrent executed test components. Multi-party testing is applied especially for systems realized by decentralized or distributed components.

Annex E: SCTR Template

System Conformance Test Report for

____

1. Identification Summary

1.1 System Conformance Test Report

STCR NUMBER:	______________________
STCR DATE:	______________________
TEST LABORATORY MANAGER:	
	______________________
SIGNATURE:	______________________

1.2 Test Laboratory

____

____

____

1.3 Test Client

____

____

____

1.4 System Under Test

NAME:	______________________
Version:	______________________
Supplier:	______________________

Dates of Testing: ____

1.5 Nature of Conformance Testing

Conformance testing deals with both the capabilities and behaviour of an implementation. It is used to check an implementation against the conformance requirements in the relevant International Standards (IS) or the ITU-T Recommendations.

Note:

Conformance testing does neither include assessment of the performance nor of the robustness or reliability of an implementation. It gives not judgements either on the physical realization or on the internal protocols of the implementation.

The purpose of conformance testing is to increase the probability that different ATM implementations/systems will interwork reliable. However, the complexity of protocols make exhaustive testing impractical on both technical and economic reasons. Therefore, there is neither a guarantee that successfully tested systems conform to every aspect of the underlying specification(s) nor a guarantee that it will interoperate with every other system. But the successful passing of the tests gives stronger confidence that the tested implementation/system will behave standard conformant and will show a high degree of multi-vendor-interoperability.

1.6 Limits and Reservations

____

1.7 Record of Agreement

IUT ____

ABSTRACT TEST METHOD ____

ATS STANDARD/

RECOMMENDATION: ____

1.8 Comments

____

____

____

____

____

____

2. *System Report Summary*

2.1 Protocol Layer Testing Summary for

IMPLEMENTATION IDENTIFIER	________________
IUT DEFINITION REFERENCE	________________
SYSTEM STANDARD/ RECOMMENDATION:	________________
ICS:	________________
IXIT:	________________
FTCR NUMBERS and FTCR DATES:	________________
ATS RECOMMENDATIONS:	________________
MEANS OF TESTING IDENTIFICATION:	________________
CONFORMANCE STATUS	________________
STATIC CONFORMANCE ERRORS?	________________
DYNAMIC CONFORMANCE ERRORS?	________________
TEST CASES RUN	________________
PASSED	________________
FAILED	________________
INCONCLUSIVE	________________

OBSERVATIONS (OPTIONAL)

____

____

Annex F:FCTR Template

Facet Conformance Test Report for

1. Identification Summary

1.1 Protocol Conformance Test Report

FTCR NUMBER:	
FTCR DATE:	
CORRESPONDING SCTR NUMBER:	
CORRESPONDING SCTR DATE:	
TEST LABORATORY MANAGER:	
SIGNATURE:	

1.2 IUT

NAME:	
VERSION:	
SYSTEM STANDARD/RECOMMENDATION:	

ICS: ______________________________

PREVIOUS FCTR(S): ______________________________

1.3 Testing Environment

IXIT: ______________________________

ATS STANDARD/RECOMMENDATION:

____

ABSTRACT TEST METHOD: ______________________________

MEANS OF TESTING IDENTIFICATION:

____

DATES OF TESTING: ______________________________

CONFORMANCE LOG REFERENCE:

____

RETENTION DATE FOR LOG REFERENCE:

____

1.4 Limits and Reservations

____

1.5 Comments

____

2. IUT Conformance Status

This IUT has been shown to be conforming/non-conforming the the specified protocol/recommendation.

3. Static Conformance Summary

The PICS for this IUT is consistent/not consistent with the static standard/recommendation.

4. Dynamic Conformance Summary

The test procedure did not/did reveal errors in the IUT.

5. Static Conformance Review Report

6. Test Campaign Report

No.	**ATS Reference**	**Selected**	**Run**	**Description**	**Verdict**	**Observations**
1.	TestGroup1/ TestCase1			Verify that the IUT... (Ref. section 1.1)		
2.	TestGroup1/ TestCase2			Verify that the IUT... (Ref. section 1.2)		

7. Observations