Formalization and Testing of Reference Point Facets

Ina Schieferdecker, Mang Li, Axel Rennoch

GMD FOKUS
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
phone: +49 30 3463-7000, fax: +49 30 3463-8000

{schieferdecker, m.li, rennoch}@fokus.gmd.de
www.fokus.gmd.de/tip

Abstract

The paper introduces a new concept to express the architecture and behavior of distributed systems in a formal, detailed and extensible manner in terms of reference point facets (RP-facets). RP-facets are based on the well-established concept of reference points as used in ODP and TINA. Facets describe statical and dynamic aspects of reference points as well as pre- and post-conditions for their use. The paper 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. An example taken from the TINA retailer reference point shows the application and practical use of RP-facets.

1 Introduction

TINA (Telecommunications Information Networking Architecture [12]) is an open system architecture for telecommunication systems in a multi-vendor environment. TINA combines modern methodologies and techniques, such RM-ODP [6] and CORBA [9] to support the development of large-scale systems. Core concepts and specifications of TINA have been established. TINA is in its maturity phase, where conformance evaluation plays an important role. In TINA, telecommunication stakeholders are characterized by their business roles, e.g. consumer, retailer or third-party service provider. Since every stakeholder represents an autonomous administrative domain, the implemented sub-systems used by stakeholders operate in a heterogeneous, unpredictable, and uncontrollable environment. Inter-domain reference points are introduced to ensure the interoperability of the various sub-systems [13].

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 [6]. An example for an inter-domain reference point is the Retailer Reference Point [14].

Key issues for multi-vendor systems are interoperability and interworking. Reference points as a collection of conformance requirements are the basis to increase the likelihood for interworking and 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 and interworking.

The Conformance Testing Methodology and Framework (CTMF) [5] 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, Internet protocols and many others. A recent work shows also the usability of CTMF for object-oriented systems [3].

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. In order to support conformance testing more effectively and efficiently, we propose a new partitioning concept for reference points: the concept of *reference point facets* (RP-facet)[1] [2] [16].

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.

In this paper, we present the concept of and specification techniques for RP-facets in Section 2 and 3, resp. Section 4 discusses RP-facet based test specification and a conformance test method based on RP-facets. An example showing the overall approach is presented in Section 5. Conclusions finish the paper.

2 TINA Reference Points

TINA inter-domain reference points[3] are located at the border of administrative domains in a telecommunication system and are used to define conformance and interoperability requirements for the business relationships between the telecommunication stakeholders, which are in certain business roles. The TINA business model (see Figure 1) defines five business roles: consumer, retailer, third-party service provider, broker and connectivity provider. It defines the following inter-domain reference points:

1. The term facet is used in the OMG CORBA Component Model (CCM). In order to avoid misunderstandings, RP-facet is used instead.
2. Please note that although we consider reference point facets under the realm of TINA the results of this work is in general applicable to distributed system, which use a notion of reference points as interface/set of conformance requirements to the outside.
3. Intra-domain reference points are not considered in this paper.

- Retailer inter-domain reference point (Ret)
- Broker inter-domain reference point (Bkr)
- Third-party inter-domain reference point (3Pty)
- Retailer-to-retailer inter-domain reference point (RtR)
- Connectivity service inter-domain reference point (ConS)
- Terminal connection inter-domain reference point (TCon)

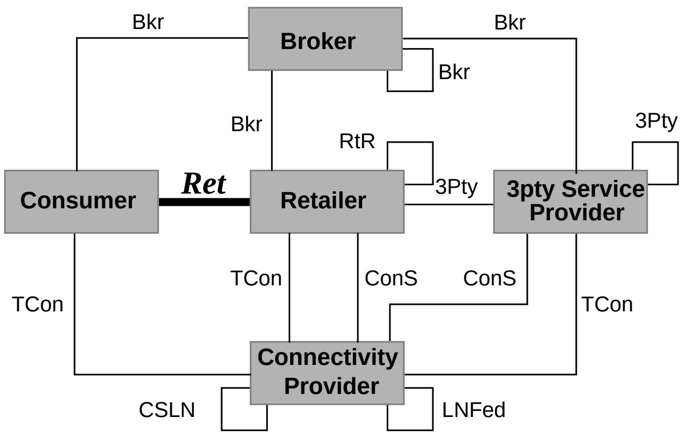


Figure 1 TINA business model

- Layer network federation inter-domain reference point (LNFed)
- Client-server layer network inter-domain reference point (CSLN)

The current TINA RP interfaces are operational. That is, the interactions over interfaces occur in form of operation invocations. Via an operation, a client requests the execution of some functionality by the server object that provides the operational interface. Typically, an operation invocation returns the results (after successful termination or exceptions) to the client. “Oneway” operations are special case of operations that do not require a response to the client.

In general, the computational viewpoint of RPs are characterized by object interfaces using computational languages. TINA RPs are specified using the ODL (Object Definition Language) [15]. An ODL specification defines objects with their interfaces and object groups, which constitute e.g. an RP. ODL provides syntax for the structural description of systems only. A formalization of behavioral specification is not prescribed.

We consider in this paper the Ret-RP between consumer and retailer as an example. In terms of telecommunication services, the retailer serves as the service provider and the consumer as the service user. The Ret-RP offers generic access to telecommunication services, operations for the discovery and start of operational, management, and administrative service offerings, operations for the control and management of service sessions such as announcement, termination, suspension, invitation, notification for the service users participating in a service session.

Ret-RP is separated into an access part and a usage part. The access part contains interfaces that are required to establish a contractual relationship between consumer and retailer, which is referred to as an access session. A service session can be built only upon an access session. The usage part of Ret-RP captures service session related interfaces. Ret-RP features are indicated either as mandatory or optional. This differentiation is significant for conformance testing.

3 The RP-Facet Concept

RP-facets define refinements of TINA reference points. An RP-facet is to enable interaction among components with separable concerns. It is a meaningful and self-standing portion of functionality. An 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. An RP-facet depends on the presence of the “core” facet and may depend on the presence of other RP-facets.

The RP-facet concept facilitates 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 an RP-facet is specified by the signature and behavior of operations[1]. Operations provide services to object’s environment, whereas interfaces represent access points for services.

Typically, an inter-domain TINA reference point separates two business roles with distinguished functionality. An 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. A reference point is defined by a set of interfaces, each of which offers functionality to the outside via operations.

Definition 1: *An interface[2] 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 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:*

1. Operational interfaces are considered currently. The results are directly applicable to event interfaces. Stream interfaces will be considered in a further work.
2. 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.

- *$I \in SI_R^{mand}$ 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 $o_1..o_n$.[1]
o is <u>independent</u> if for all n there is no sequence of operations $o_1..o_n$, on which o is dependent.

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

Definition 4: *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 $(o, \emptyset) \in dep_{I,R}$.*

An 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 an RP-facet. Please note that the set of operations of an interface may be used only partially in an RP-facet:

Definition 5: *An <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 $o2 \in F_R$.*

1. 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`.

- *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 an RP-facet.*

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

Definition 6: *The* <u>*order relation*</u> $\leq$ *on RP-facets uses the subset relation:*
$\forall F1, F2 \in SF_R$: $F1 \leq F2$ *iff* $F1 \subseteq F2$.

Lemma 3:
- SO_R *is the maximal element of* $\leq$, *i.e.* $\forall F \in SF_R$: $F \leq SO_R$.

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 7: *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 4:
- C_R *is unique.*
- C_R *is an RP-facet.*

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 8: *A* <u>*core-based RP-facet*</u> $F_{R,C}$ *is an 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* $SF_{R,C}$.

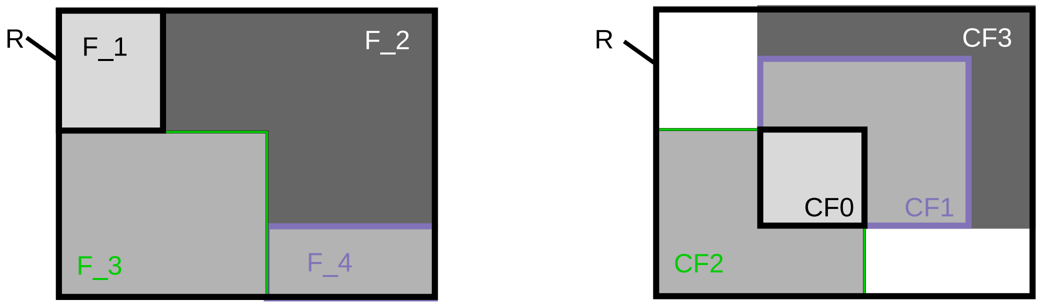
A core-based RP-facet covers at least the core and possibly additional optional operations.

Lemma 5:
- C_R *is a core-based RP-facet.*
- SO_R *is a core-based RP-facet.*

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

Lemma 6:
- 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.*

The relation of RP-facets and core-based RP-facets are depicted in Figure 2.



Partitioning of R into RP-Facets F_1 .. F_4 with $SO_R = F\_1 \cup F\_2 \cup F\_3 \cup F\_4$ and $F\_1 \cap F\_2 = \emptyset$, etc.

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

Figure 2 Reference Points and its RP-Factes

The conformance test method (see Section 5) will be based on the concept of core-based RP-facets and their hierarchies[1], as they naturally reflect the mandatory and optional requirements for a reference point and their relation.

4 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 an RP-facet including its static and dynamic models is crucial for testability. As any formalization reduces misinterpretation of the system under test, 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) [7] for signatures of RP-facets in combination with Message Sequence Charts (MSC) [8][2]. Additions are needed to cover specific aspects of RP-facets according to the concepts introduced in the previous section. The specification template for RP-facets compresses:

- indication to the related reference point and the RP-facet role,
- statical specification of the RP-facet in ODL, and

1. Please note that for every core-based facet F there is at least the following hierarchy $C_R \leq F \leq SO_R$.
2. 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.

- behavioral specification of the RP-facet in terms of use scenarios, including representations of dependence relations in MSC.

Further, we use the standard test notation TTCN (Tree and Tabular Combined Notation) to formulate test cases for RP-facets.

The RP-facet specification and test case generation cycle is presented in Figure 3. 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.

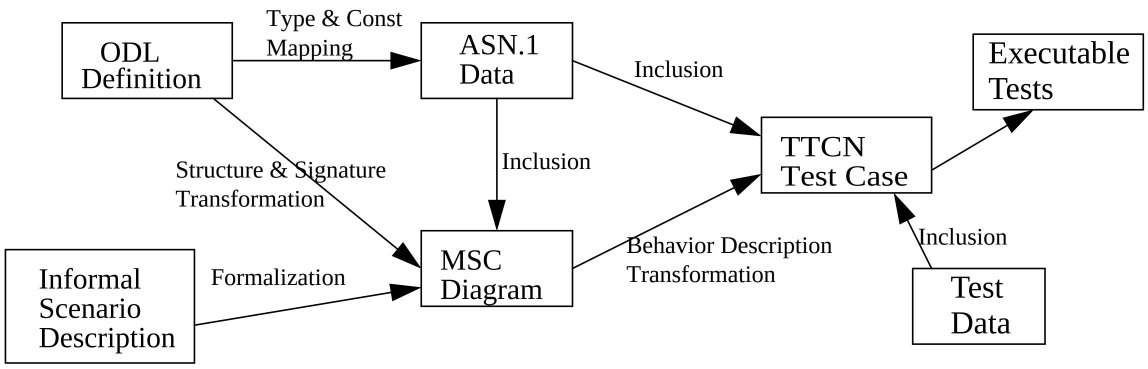


Figure 3 RP-facet specification and test generation

4.1 Structural Specification Template

The template for the RP-facet structural specification is an extension of the TINA reference point specification template [13], which uses TINA-ODL [15]. ITU-T ODL [7] adopts most of the concepts and definitions of TINA-ODL. Thus, the following discussion on ODL refers to ITU-T ODL.

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. An example of the TINA Retailer Reference Point (Ret-RP) [14] specification is shown below.

```
#include "TINACommonTypes.idl"

module TINAProviderInitial {
 interface i_ProviderInitial {
   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 ();
 };
};

module TINARetRetailerInitial {
 interface i_RetailerInitial: TINAProviderInitial::i_ProviderInitial {
 };
};
```

It specifies the interface *i_RetailerInitial* of the retailer domain, that inherits definitions of the interface *i_ProviderInitial*. Domains where interfaces reside, are indicated in the naming of modules and interfaces, e.g. *TINAProvider*, *TINARetRetailer*. *TINAProvider* is a generalized business role and can be specialized to a consumer, retailer, third-party service provider, broker and connectivity provider. In our example, *TINAProvider* is specialized to *TINARetRetailer*. The counter part of a provider is a user, which is at the Ret-RP a consumer.

Interactions at the Ret-RP between a retailer and a consumer involve not only interfaces provided by the retailer, but possibly also those interfaces supported by the consumer. This reflects the object model, in which an object supports interfaces, where services can be used by clients, and may require interfaces that are provided by other objects in its environment. Thus, there is a need for the identification of *supported* and *required* interfaces. The ODL's object template provides a notion for this purpose. Further, the template is also used to indicate the reference point and the role related to an RP-facet.

As shown in the following example, the module *TINARet* corresponds to the considered reference point. The facet role is represented by the object template identifier *Retailer* prefixed by the keyword *CO* (originated from Computational Object). Related interfaces are declared respectively behind the keywords *requires* and *supports*.

```
module TINARet {
 CO Retailer {
   requires
     Consumer::i_ConsumerInitial;
   supports
     i_RetailerInitial;
     i_RetailerAccess;
 };
};
```

Dependence relations of operations and interfaces (see Section) allow reuse of ODL definitions by inclusion. A systematic document structuring eases system evolution.

4.2 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 [3][1]. 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

1. 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.

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.

4.3 Behavioral Specification Template

Message Sequence Charts (MSC) is a graphical and formal trace language defined by ITU-T [8]. MSC describes interactions between message-passing instances. MSC-2000 [8] 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.

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

An MSC document defines an instance kind for an RP-facet. It contains instances, messages, timer and MSC diagram declarations. In addition, a data language to be used in the MSCs 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 a 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 used instead of method calls to make the representation of alternative operation invocation outcomes, in particular under exceptional conditions, more readable. In addition, the mapping to TTCN in the case of asynchronous messages is straightforward (see also Section 5).

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 “readonly”, 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 the 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. The definition of these functions for using ASN.1 in MSC at these interfaces is currently under work.

5 RP-Facet Based Testing

The RP-facet concept, in particular the self-containment property of an RP-facet, supports 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 leads 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 an RP-facet define requirements 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 [5].

5.1 Test Specification

TTCN (Tree and Tabular Combined Notation) was designed for conformance testing of OSI protocol implementations [5]. 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 abstract service primitives (ASPs)[1]:

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 an MSC instance:

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.*

The assignment of one PCO to one PTC is not stringent, but recommended. The semantics of a

1. The selection of ASPs instead of protocol data units (PDUs) is based on the analogies between the object model and the OSI reference model. Please refer to [3] for details.

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 presented in the paper (Figure 5), 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 sufficient to cover MSC timer events.

The derivation of TTCN test descriptions from HMSCs is as follows:

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.

5.2 Test Campaign Derivation

In general, software testing is time and cost intensive, i.e. critical for large systems. Therefore, CTMF [4] 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 an 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 or belong also to *I*. Let *N* be the core-based RP-facet under test in the current testing iteration.

```
**Start**: $T=\emptyset$, $I=\emptyset$, $i=1$, $N=CF_i$.
**Iteration i**:
        Step I: Select $o \in N$ with $(o, \emptyset) \in dep_{I,R}$ :
            /* 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, $N=N \backslash \{o\}$

            Repeat until no further operations o with $(o, \emptyset) \in dep_{I,R}$ 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 tested successfully*/
            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, $N=N \backslash \{o\}$ .

            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 $N=N \backslash \{o\}$ .

            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, $N=N \backslash \{o_j\}$ .
            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 [10]. The other test groups, which focus on testing of valid/invalid sequences of operations at RP-facets, can be derived using traditional test derivation algorithms well known from e.g. LTS or EFSM based test generation methods implemented in several academic and commercial test derivation tools.

6 An Example

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

The following ODL definition is a simplified representation of [14] (see also Section 4.1). It indicates the reference point *Ret* and the RP-facet role *Retailer.* It defines further: *Retailer* provides a *i_Initial* interface and a *i_Access* interface; *Retailer* uses the *i_Initial* interface of the *Consumer*; the operation *namedAccess* of the *Retailer*'s *i_Initial* interface requires an input parameter for passing some user information, and provides an output parameter for returning a reference of requested object, and in case that the passed user information is invalid an *PropertyError* exception is raised. The consumer domain services are defined in a separate document *Ret_Consumer.odl*.

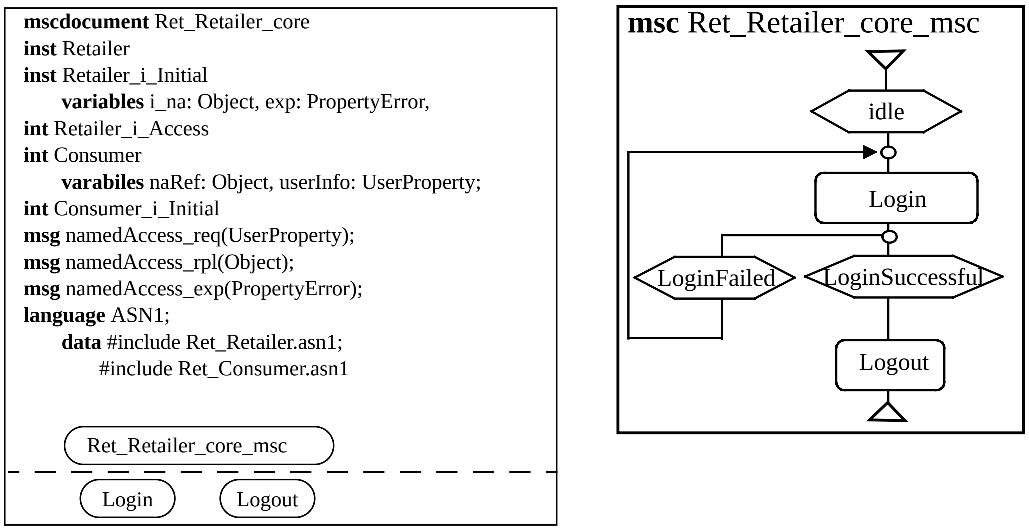
```
#include "Ret_Consumer.odl"
module Ret {
 CO Retailer {
   requires
     Consumer::i_Initial;
   supports
     Retailer::i_Initial;
     Retailer::i_Access;

   interface i_Initial {
     void namedAccess (
       in UserProperty userInfo,
       out Object i_na;
     ) raises (PropertyError);
     ...};
   interface i_Access {...};
 };};
```

From the textual description of Ret-RP business scenarios, two RP-facets can be derived:

- The core facet *Ret_Retailer_core* involves login and logout of a consumer at the retailer domain. The retailer's interface *i_Initial* and the operation *namedAccess* are used by login.
- An additional facet *Ret_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.

Ret_Retailer_core is organized by the MSC document and High-level MSC (HMSC) presented in Figure 4. According to Rule 2, five instances (prefixed by **inst**) are defined: *Retailer*, *Retailer_i_Initial*, *Retailer_i_Access*, *Consumer* and *Consumer_i_Initial*. The operation *namedAccess* 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 *Ret_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 Ret_Retailer_core
inst Retailer
inst Retailer_i_Initial
    variables i_na: Object, exp: PropertyError,
int Retailer_i_Access
int Consumer
    varabiles naRef: Object, userInfo: UserProperty;
int Consumer_i_Initial
msg namedAccess_req(UserProperty);
msg namedAccess_rpl(Object);
msg namedAccess_exp(PropertyError);
language ASN1;
    data #include Ret_Retailer.asn1;
        #include Ret_Consumer.asn1

    Ret_Retailer_core_msc
- - - - - - - - - - - - - - - - - - - - - - -
      Login      Logout
```

Figure 4 MSC example of Ret_Retailer_core

Details of purpose-oriented use scenarios of RP-facets are described by MSC event traces. Figure 5 shows the message exchanges between a *Consumer* instance and a *Retailer_i_Initial* instance in relation to the login activity. The two alternative outcomes of a request on the operation *namedAccess* 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.

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 6. 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.

Table 3 shows the dynamic part of the TTCN specification of a test case in accordance with the Login MSC (Figure 5). 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_Initial* interface. The TS emulates the behavior of a client of *i_Initial*. It uses a client-PCO named *PCO1_Retailer_i_Initial* defined using Rule B. The purpose of this test case is to verify: after a request on the operation *namedAccess* with valid

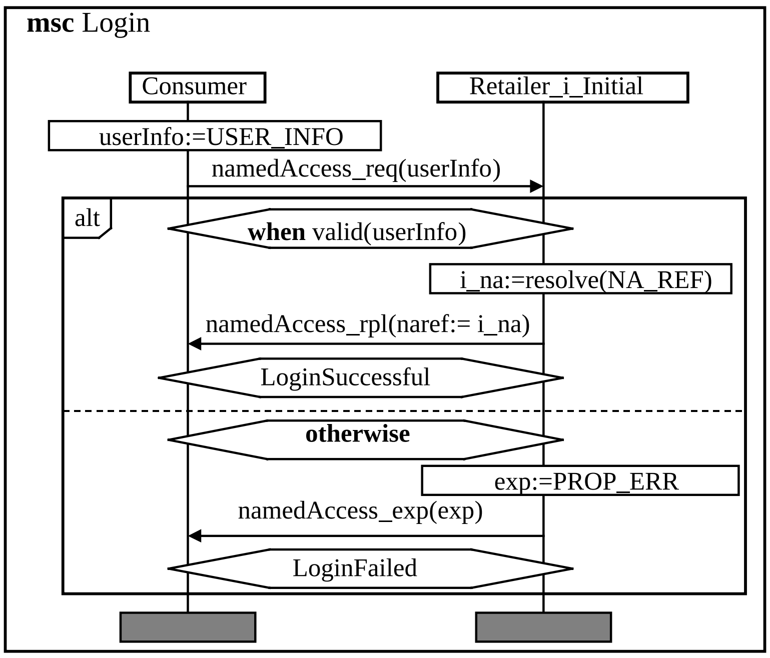
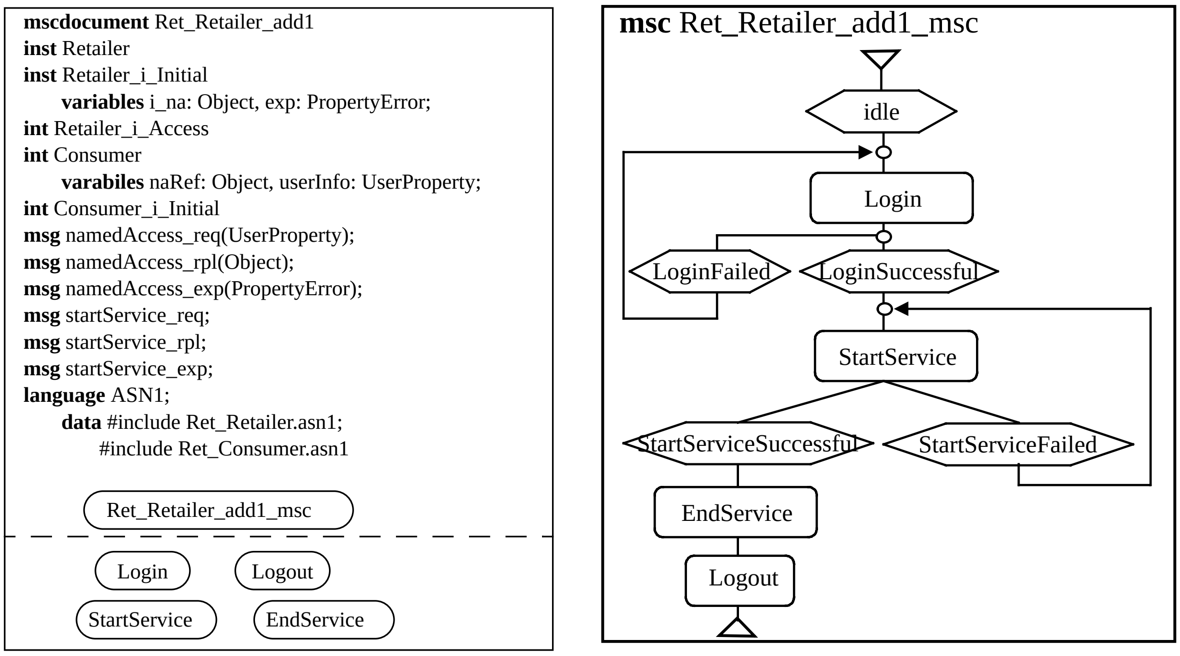


Figure 5 Login MSC diagram

user information is sent to the SUT, a reply of *namedAccess* 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.*



```
mscdocument Ret_Retailer_add1
inst Retailer
inst Retailer_i_Initial
     variables i_na: Object, exp: PropertyError;
int Retailer_i_Access
int Consumer
     varabiles naRef: Object, userInfo: UserProperty;
int Consumer_i_Initial
msg namedAccess_req(UserProperty);
msg namedAccess_rpl(Object);
msg namedAccess_exp(PropertyError);
msg startService_req;
msg startService_rpl;
msg startService_exp;
language ASN1;
     data #include Ret_Retailer.asn1;
          #include Ret_Consumer.asn1
```

Figure 6 MSC example of Ret_Retailer_add1

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				
No	**Label**	**Behavior Description**	**Constraints Ref**	**Verdict**
1		+GetInitialRef		
2		PCO1_Retailer_i_Initial ! **pCALL_**Retailer_i_Initial__**namedAccess**	pCALL_namedAccess_s1	
3		START Timer1		
4		PCO1_Retailer_i_Initial ? **pREPLY_**Retailer_i_Initial__**namedAccess** CANCEL Timer1	pCALL_namedAccess_r1	(P)
5		+Logout		
6		?TIMEOUT Timer1		I

Table 3 TTCN test case example

The implementation and execution of TTCN-based test case in the CORBA environment is discussed in [3].

7 Conclusions

The goal of the work presented in this paper is to provide concepts and means for testable specifications that facilitate conformance and interoperability testing for distributed systems. The approach is based on the RM-ODP and TINA reference point concept. It refines reference points into self-contained and extensible facets, referred to as RP-facets, in order to allow incremental specification, implementation and testing of distributed systems at their reference points.

Formalization is key to the concept. Besides a mathematical characterization of RP-facets, a template for structural and behavioral specifications of RP-facets as well as a conformance test method are elaborated. The specification template consists of ODL, ASN.1, and MSC to provide adequate information detail for the derivation of abstract test cases in TTCN. The combination of all these specification techniques is shown by an example.

In addition to a thorough usability study of the approach, future work will be on the support of further testing aspects, e.g. operational conformance under load situation, what requires extensions of the RP-facet specification template. Additional issues of test campaigns, which are partly discussed in the paper, such as efficient test strategy, will be also considered.

8 References

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

[2] 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.

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

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

[5] 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.

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

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

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

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

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

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

[12] TINA-C: Overall Concepts and Principles of TINA, version: 1.0, Feb. 1995.

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

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

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

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