5

INCREMENTAL TESTING AT SYSTEM REFERENCE POINTS

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, http://www.fokus.gmd.de/tip

Abstract RM-ODP (Reference Model of Open Distributed Processing) is a reference model for open information processing systems. Conformance is one of the key aspects of open systems. Conformance ensures the interworking in a multi-vendor information processing environment that may evolve over the time. An essential partitioning concept for open system is that of reference points (RPs). RPs consist of a set of interfaces together with potential interactions at these interfaces. RP specifications define conformance requirements, so that they can be used to determine the conformance of a system. However, the testing of RPs of real systems is restricted as their functionality is often very complex. Refinement of the RP structure is required to ease both their realization and also their testability. In particular, the incremental development of RPs that allows both the evolution of the system architecture and the step-by-step implementation by vendors should be supported. A new concept - that of an RP-facet - has been recently defined to structure RPs and to improve their testability. The focus of this paper is on structuring principles and dependence relations used by the RP-facet concept. They serve as a basis for incremental and efficient testing at RPs. An example taken from use scenarios on service access in open systems is elaborated in the paper.

Keywords: Conformance Testing, Reference Points, Dependence Relations, ODL, MSC, TTCN, ODP

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35516-0_20

H. Ural et al. (eds.), *Testing of Communicating Systems*
© IFIP International Federation for Information Processing 2000

1. INTRODUCTION

Conformance is one of the key aspects of open systems. Conformance ensures the interworking in a multi-vendor information processing environment that may evolve over the time. Conformance evaluation accompanies the whole development process of a system. It is in particular used in the late phase of technology maturity, where it is carried out in form of conformance testing. The consideration of conformance is an essential part of the system architecture and specification. In particular, the design for testability and the specification for testability have to be addressed.

A reference model for open information processing systems is the RM-ODP (Reference Model of Open Distributed Processing) REF[8]. Five viewpoints, the object model and the conformance assessment are basic concepts of RM-ODP. The enterprise, information, computational, engineering and technology viewpoints provide consideration of a system from different concerns. The object model is the basis for the flexibility, modularity and scalability of ODP systems. An object is an entity that contains information and offers services at its interfaces. An ODP system is composed of interacting objects, which may be organized in object groups. As objects or groups of objects may origin from different sources, the reference point concept is used as a basis for conformance assessment. Reference points (RPs) are potential conformance points at which the conformance of an ODP system is determined.

TINA (Telecommunications Information Networking Architecture) [15] is an open system architecture based on RM-ODP. TINA adopts and specializes many concepts of RM-ODP, including those introduced above. TINA is in its maturity phase, where conformance evaluation plays an important role. In TINA, telecommunication stakeholders are generalized by 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 and unpredictable/uncontrollable environment. Inter-domain RPs are introduced to ensure the interoperability of the various sub-systems.

However, the testability of TINA RPs is restricted. At first, the TINA RP specification template is lacking a behavioral description, so that current TINA RP specifications are inadequate for conformance testing. Secondly, the functionality of an inter-domain RP is often very complex. A refinement of the RP structure would ease their implementation and testing. In particular, an incremental development of RPs that allows both the evolution of the system architecture and the step-by-step implementation by vendors would be beneficial.

The TINA Conformance Testing Framework Request for Proposal (RFP) REF[17] initiated a facet concept to improve the testability of TINA RPs. This paper presents an extension of our contribution to the RFP [4]. The focus of this paper are various *dependence relations* of RP-facets, which serve as a basis for efficient testing of RP-facets. The paper is structured as follows: Section 2 presents an overview on the notion of RPs in ODP and TINA. The structure of an RP is analyzed in Section 3. Based on this analysis, the notion of RP-facets and an approach to the specification of RP-facets are given. Section 4 discusses the basic principles for conformance testing of RP-facets. The main ideas for the definition of an efficient test campaign for RP-facets is given in Section 5. Conclusions finish the paper.

2. REFERENCE POINTS

In RM-ODP, all interfaces are defined as RPs, while subsets of those RPs to which conformance requirements apply are chosen as conformance points. Four classes of RPs are defined: perceptual, programmatic, interworking and interchange. They correspond to object interfaces to human beings, to other objects or to storage media. Programmatic RPs refer to interfaces allowing logical access to functions. They correspond to intra-domain RPs in TINA. At an interworking RP communication between several systems can be established. It corresponds to an inter-domain RP in TINA. Perceptual RPs referring to interfaces between the system and the outside world, and interchange RPs constituted by interfaces bridging a system and an external physical storage medium, are not in the scope of TINA.

TINA RPs impose conformance requirements on the involved interfaces, so that there is no separation between RPs and conformance points. TINA RPs reside between generalized telecommunication stakeholders, i.e. business roles. The TINA business model (see Figure 1) defines five business roles: *consumer*, *retailer*, *third-party service provider*, *broker*, and *connectivity provider*, and the corresponding domains. Inter-domain RPs and intra-domain RPs are defined. Except that intra-domain RPs are located within administrative domains, they are guided by the same principles as inter-domain RPs.

In this paper, we consider inter-domain TINA RPs. Their functionalities are realized by various objects such as user agent, initial agent, provider agent, etc. Conformance requirements that are imposed on the interfaces of the involved objects are analyzed and a testing method is presented. We consider the Ret-RP between consumer and retailer as an example. In terms of telecommunication services, retailer is the service provider and consumer is the service user.

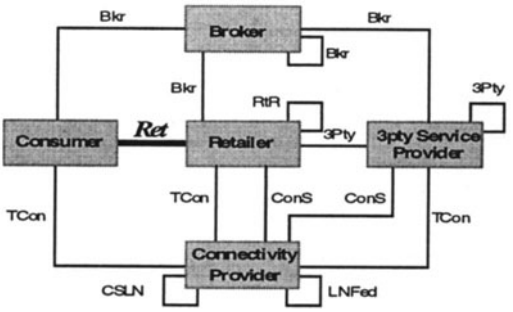

Figure 1. TINA business model

In general, the computational viewpoint of RPs is defined by object interfaces, which are specified in computational languages. TINA RPs are specified using ODL (Object Definition Language) [9]. ODL provides syntax for structural description of objects and object groups. A formalization of behavioral specification is not prescribed.

The current TINA RP interfaces are operational meaning that the interactions at interfaces occur in form of operation invocations. With an operation invocation, a client requests the execution of some functions by the server object, which are provided by the operational interface of the server object. Typically, an operation invocation returns the termination (results or exceptions) to the client. "Oneway" operations are special case of operations that do not require a response to the client.

The 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. It is referred to as the access session. The access session is service-independent. The usage part of Ret-RP captures service session related interfaces. In contrast to the access session, a service session is service specific and is built only upon an access session.

3. RP FUNCTIONALITIES AND RP-FACETS

The whole Ret-RP is characterized by a number of functionalities being either mandatory or optional. The functionalities are in general realized by various operations at different interfaces, so that functionalities impose a logical structure on a RP: the RP can be partitioned into those operations and interfaces that are used for a selected functionality and into the rest. The logical grouping at an RP has been reflected already in the Ret-RP specification with the concept of feature sets. It has been further refined with the concept of segment[1] in [12]REF.

RP-facets reflect another structuring principle for refining TINA RPs. It has been introduced in [13]. An RP-facet is a meaningful and *self-contained* portion of functionalities. Let us have a first look at the example: the Ret-RP has the functionalities *Login*, *StartService*, *Invitation* and *Logout*, which stand in a logical dependency. For example, the *Login* is the precondition for *StartService*, what is the precondition for *Invitation* and *Logout*. Self-contained subsets are {*Login*}, {*Login, Logout*}, {*Login, StartService*}, {*Login, StartService, Logout*}, {*Login, StartService, Invitation*}, and {*Login, StartService, Invitation, Logout*}. Furthermore, *Login*, *StartService*, and *Logout* are mandatory for Ret-RP as they constitute the basic activities. They are therefore part of the so called core-facet. This results in two facets: the core-facet with {*Login, StartService, Logout*} and an extended facet with {*Login, StartService, Invitation, Logout*}, either of which can be implemented by a system.

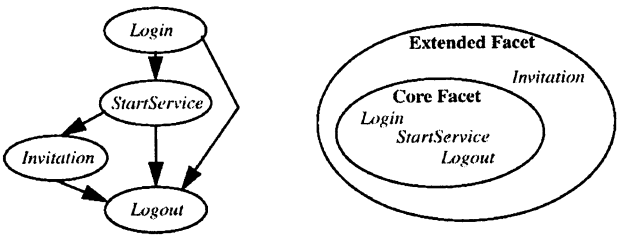

Figure 2. Selected functionalities and facets of TINA Ret-RP

Each RP is composed of one or more RP-facets. There is a core-facet that provides a minimum set of functionalities. Additional interfaces and interactions can be specified to provide extended functionalities. A RP-facet depends on the presence of the core-facet and may depend on the presence of other RP-facets. Dependent RP-facets form a chain of cohesive sets.

The RP-facet concept facilitates conformance testing by testing the dependent RP-facets according to their order in this chain (see Section 5). Functionalities of a RP-facet are specified by the signature and behavior of operations. In addition, a RP-facet is associated with one of the architectural parts separated by the RP, referred to as *RP-facet role* (e.g. retailer or consumer).

3.1 Structures and Dependencies at a Reference Point

The initial definition of RP-facets [13] includes two relations:

- a *dependence relation* between operations (e.g. an operation can be invoked only after the invocation of another operation), and

- an *order relation* on RP-facets (e.g. a RP-facet is a subset of another one).

There are two additional kinds of dependencies: A RP offers different functionalities that are realized by sets of operations. For each functionality, a number of use scenarios are defined to be the required or forbidden patterns of behavior. The use scenarios refer to preconditional functionalities for the functionality under consideration. A functionality is in this case *causally related* to its preconditional functionalities. For example, *StartService* can be used only after a successful *Login*.

Another kind of relation exists in the case that more than one RP instances are involved in a communication, e.g. in a conference call scenario. It is possible that the involved RP instances belong to the same RP type (e.g. both are of type Ret-RP) or that they are instances of different RP types (e.g. of type Ret and of type 3Pty-RP). For example, an invitation is realized by two Ret-RP instances: one instance at the domain of the inviting consumer and the other at the domain of the invited consumer. We denote this relation between RP instances as *usage relation*. It places requirements on the test execution environment. When testing the global behavior, the consistency of events that occur across RPs (instances) has to be considered.
The structuring principles for an RP are summarized in Figure 3.

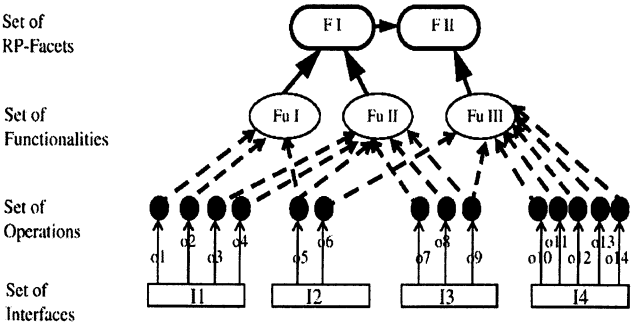

Figure 3. RP structure

Subsequently, we give a series of definitions to reflect the dependence relations at RPs:

Definition 1: *An interface[2] I has a set of operations SO_I. A reference point R has a set of interfaces SI_R.*

Definition 2: *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 dependent on $o_1..o_n$ iff a successful invocation of o requires previous successful invocations of $o_1.. o_n$[3]. o is independent iff there is no operation o_k, $k \in \{1, ...n\}$ on which o is dependent.*

Definition 3: A functionality Fu is a set of operations SO_{Fu} at potentially different interfaces of R. The set of operations covers all those operations that are invoked by the required use scenarios $(o_1,...,o_k)$, ... of this functionality. The set of functionalities of R is denoted by SFu_R.

A RP can be structured into functionalities in different manners. Functionalities reflect the use of a RP by its user, i.e. the service aspects are considered here. For the example in Figure 2, another possibility would be to consider one functionality for the *Login/out* behavior and the operations related to *StartService* and/or *Invitation* may comprise other functionalities.

Definition 4: Let fu_1, $fu_2 \in SFu_R$ each with its corresponding sets of use scenarios $\{u_1,...\}$, $u_i=(o^1_{i1},..,o^1_{in})$ and $\{r_1, ...\}$, $r_j=(o^2_{j1},..,o^2_{jm})$. fu_2 is causally related to fu_1 iff there is an i and j with o^2_{j1} (i.e. a start operation of a use scenario of fu_2) dependent on o^1_{in} (i.e. a successful end operation of a use scenario of fu_1). Otherwise, fu_1 is causally unrelated to fu_2.

We extend the definition of a RP-facet given in [13]. RP-facets are now composed of functionalities, so that RP-facets define a "logically" complete partitioning of RPs. While functionalities can be defined in a rather flexible way, a RP-facet contains all those functionalities between which dependencies on their operations exist. Therefore, a RP-facet is self-contained.

Definition 5: A RP-facet F_R is a set of functionalities of a reference point with the following properties:

- *it is non-empty, and*
- *$\forall fu \in SFu_R$ $\forall fu' \in SFu_R$ causally related to fu: if $fu \in F_R$ then $fu' \in F_R$ (the self-containment property).*

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

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

The relations at a RP are used to direct the conformance testing process for a RP (Section 5).

3.2 RP-Facet Specification

The unambiguous specification of RP-facet including its static and dynamic models is crucial for testability. 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

RPs under test 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 ODL [9] for signatures of RP-facets in combination with Message Sequence Charts (MSC) [10]. 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 comprises:

- an indication to the related RP and the RP-facet role,
- static 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.

The RP-facet specification and test case generation cycle are presented in Figure 4.

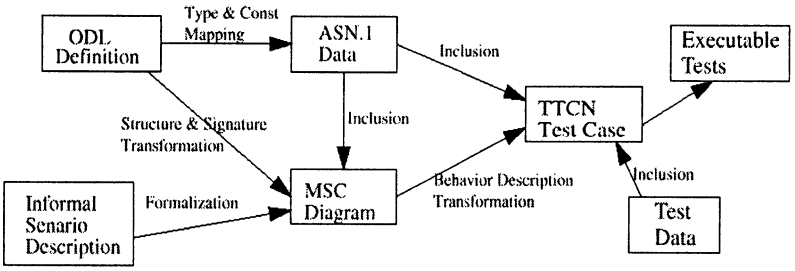

Figure 4. RP-facet specification and test generation

MSC has been selected as it presents the communication behavior in a very intuitive and transparent manner and can be used for the behavior specification of objects defined in ODL. The standard test notation TTCN (Tree and Tabular Combined Notation) [7] is used to formulate test cases for RP-facets since there is a short step between specification and test execution. Test suites in TTCN can be easily maintained, flexibly modified and extended. ASN.1 [14] is used as a common data representation form for MSC and TTCN. Mappings for data types and constants from ODL to ASN.1 need to be defined.

3.2.1 Example Scenarios

The specification approach is applied to a (simplified) example of the TINA Ret-RP [16], where the following two features are considered:

(fa) The mandatory part of the Ret-RP's access session. It includes login and logout of a consumer with a retailer, using the operation *namedAccess* of the retailer's *i_Initial* interface, as well as start and termination of services at the retailer's *i_Access* interface in case of successful login.

(fb) An optional feature of the Ret-RP's service session part, which supports the invitation of an additional consumer. The invitation is initiated by a consumer that participates already in the multiparty service. The signature of used interactions is given below (see also Section 3.2.2).

```
module TINARet {
 interface i_ProviderInviteReq {
   void inviteUserReq (in UserDetails u,out InvitationReply rep); };
 interface i_PartyInd {
   void inviteUserInd (in UserDetails u); };
 interface i_ProviderVotingReq {
   void voteReq(in ParticipantId myId,in Vote vote); };
 interface i_PartyInfo {
   oneway  void inviteUserInfo (in UserDetails u); }; };
```

The inviting consumer invokes *inviteUserReq* on the retailer's *i_ProviderInviteReq* interface with *UserDetails* of the invited consumer as input parameter. The retailer indicates the invitation by calling *inviteUserInd* of the *i_PartyInd* interface of all other consumers in the same service session. The invited consumer identifies himself by the *UserDetails* contained in *inviteUserInd*. He/she uses *voteReq* of the *i_ProviderVotingReq* interface to indicate his/her *ParticipantId* and *Vote* to the invitation. In case the invitation is accepted, all consumers except the inviting one are informed by *inviteUserInfo* on their *i_PartyInfo* interfaces.

Applying the definitions for RP-facets, we can derive a facet *Ret_Retailer_core* from *fa* representing a core RP-facet, and a facet *Ret_Retailer_add1* from *fa+fb*, which is an additional optional RP-facet.

3.2.2 Structural Specification

ODL provides means to specify on type level the system structure and relation between objects. The so called templates are used to define types for interface instances, object instances, and for object groups.

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. The access to and the visibility of the object functionality are 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 represented by *object templates*. Groups of objects are defined in terms of *object group templates*. They enable aggregation of object templates to increase the conceptual level at which programs can be designed. They increase also the modularity of designs.

Current TINA RP specifications use a subset of ODL, that is the OMG-IDL (Interface Definition Language) [11]. The example specification presented in Section 3.2.1 shows that it can be hardly identified which

interface is provided/used by which business role. The distinction of *supported* and *required* interfaces is a build-in concept of ODL. The improved structural specification of the example is shown below. The business roles retailer and consumer are represented by two COs (Computational Objects). Each of them contains a list of required interfaces and a list of supported interfaces.

```
module TINARet {
 CO Retailer {
   requires Consumer::i_PartyInd; Consumer::i_PartyInfo;
   supports i_Initial; i_Access; i_ProviderInviteReq;
            i_ProviderVotingReq;
   interface i_Initial {...};
   interface i_Access {...};
   interface i_ProviderInviteReq {...};
   interface i_ProviderVotingReq {...}; };
 CO Consumer {
   requires Retailer::i_Initial; Retailer::i_ProviderInviteReq;
            Retailer::i_Access; Retailer::i_ProviderVotingReq;
   supports i_PartyInd; i_PartyInfo;
   interface i_PartyInd {...};
   interface i_PartyInfo {...}; }; };
```

The transformation of ODL type and constant definitions into ASN.1 representations is rather straight-forward. Please refer to [13] for details.

3.2.3 Behavioral Specification

The behavior of a RP-facet is given as use scenarios and specified by MSCs. The rules how to apply MSC in this context are given in [13] and shortly explained here.

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*. A MSC describes the order in which interactions and other events take place.

The behavioral specification of a RP-facet is organized by a MSC document. The example MSC document (Figure 5) defines an instance kind for the core RP-facet *Ret_Retailer_core*. It contains the declaration of instances for the business roles separated by the RP, *RetailerCore* and *ConsumerCore*, and interfaces involved in the RP-facet, *i_Initial* and *i_Access*. In addition, the data language ASN.1 is declared.

ODL operations are transformed to asynchronous MSC messages to allow the representation of alternative invocation outcomes under exceptional conditions. A message corresponding to requests (by suffix *_req*) on an operation is defined. If the operation is not a "oneway" operation, a message related to replies (by suffix *_rpl*) on the operation is

defined in addition. Each potential exceptional outcome (by suffix *_exp*) of an operation is translated into a separate message. Figure 5 shows examples for the operations *namedAccess* and *startService*.

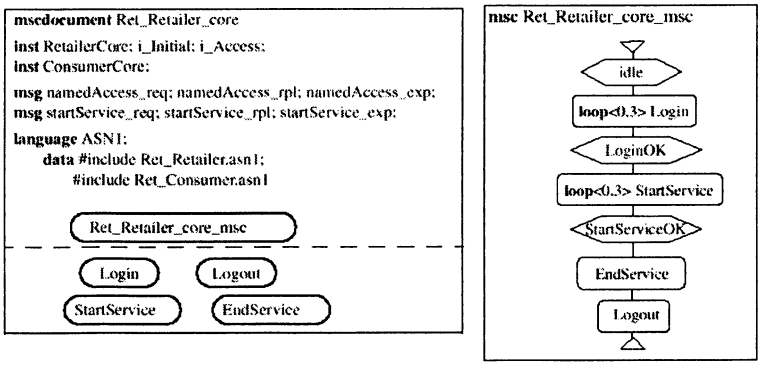

Figure 5. MSC document and HMSC for Ret_Retailer_core

The behavior of a core RP-facet is represented by a High-level MSC (HMSC) diagram, which refers to utility MSC diagrams. In this example, the diagram *Ret_Retailer_core_msc* is supported by utility diagrams *Login*, *Logout*, *StartService* and *EndService*.

The *dependence relation* and *causal relation* are represented by MSC or HMSC expressions, where the MSC sequential operator is used (see Figure 6). For example, 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, the alternative operator **alt** in combination with conditions will be used in addition. More complex behavior definitions for RP-facets will use also parallel (**par**), loop (**loop**) and optional (**opt**) expressions.

When specifying an additional core-based RP-facet, the *order relation* is reflected by the MSC's **inherits** construct, for example *mscdocument Ret_Retailer_add1 inherits Ret_Retailer_core*. This notion allows inclusion of instances and MSCs defined in the inherited instance kind. New definitions can be added in the inheriting instance kind, for example the diagram *Invitation* (Figure 6) for *Ret_Retailer_add1*.

4. A TEST METHOD FOR RP-FACETS

The test method for RP-facet is based on the Conformance Testing Methodology and Framework (CTMF) [6] for OSI systems and its test notation TTCN [7].

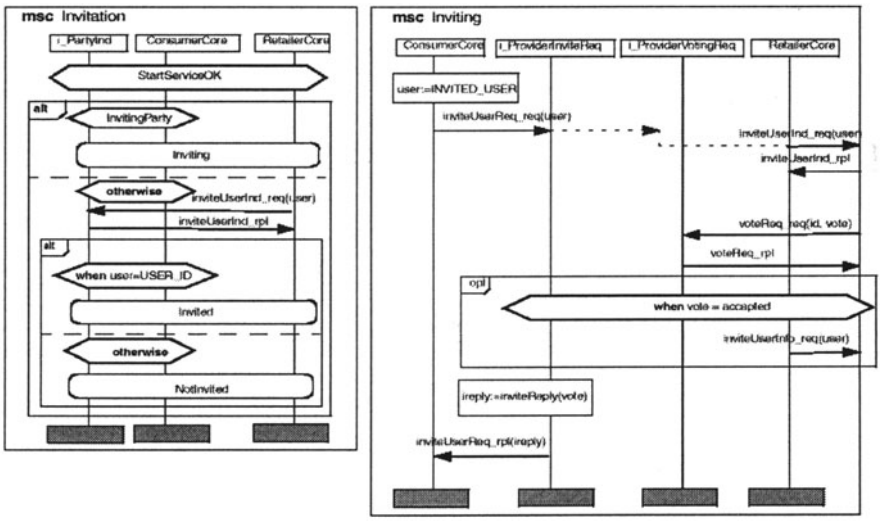

Figure 6. MSC diagram Invitation and Inviting

REFREFCTMF test architectures are built on an asynchronous communication between *systems under test* (SUTs) and *test systems* (TS). A *point of control and observation* (PCO) 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 data types* (ASPs). In decentralized test architectures, where typically several *parallel test components* (PTCs) communicate with the SUT, co-ordinated by a *main test component* (MTC), more than one PCOs can be assigned to a PTC.

In [5] and REF[13]we elaborated the analogy of the ODP's object model and the OSI reference model that leads to rules for mapping MSC constructs to TTCN constructs. For example, MSC messages for object operations and attributes are represented by TTCN ASPs. Furthermore, two kinds of PCO are defined due to the distinction of supported and required interfaces. A supported interface of a RP-facet role is interpreted by a *client-PCO* at which request-ASPs are sent to an SUT and reply-ASPs or exception-ASPs from the SUT are observed. Whereas, a required interface of a RP-facet role is interpreted by a *server-PCO* over which request-ASPs from a SUT are received and reply-ASPs or exception-ASPs to the SUT are sent.

A test component may be associated with several PCOs of both kinds. Following the approach of component-based test systems [2], we propose to use one PTC for each emulated entity (e.g. derived from a MSC instance of the corresponding MSC diagram) that interact with the RP-facet role instance to be tested. In the example, three PTCs are derived to represent the alternative behaviors of a consumer, namely *PTC_Inviting*, *PTC_Invited* and *PTC_NotInvited* (Figure 7).

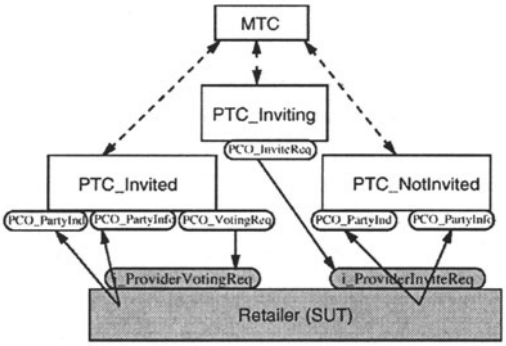

Figure 7. Test configuration

PTC_Inviting represents the consumer's behavior as an inviting user. It uses the client-PCO *PCO_InviteReq* to access the retailer's *i_ProviderInviteReq* interface. *PTC_Invited* represents the consumer's behavior as an invited user. It uses the client-PCO *PCO_VotingReq* to access the retailer's *i_ProviderVotingReq* interface, and accepts retailer's requests on interfaces *i_PartyInd* and *i_PartyInfo* of an invited consumer at the server-PCOs PCO_PartyInd and *PCO_PartyInfo*. Finally, *PTC_NotInvited* represents a consumer who is not invited but in the same session as the inviting and invited consumers. It accepts retailer's requests on interfaces *i_PartyInd* and *i_PartyInfo* at the server-PCOs *PCO_PartyInd* and *PCO_PartyInfo*.

The PTCs are co-ordinated by the MTC. The behavior of a PTC is specified in a test step that is called when the *CREATE* construct is used to instantiate a PTC. Typically, the test purpose relevant activities are reflected in PTCs test steps as e.g. for *PTC_Invited* shown in Table 1. This test step takes two parameters for PTC instantiation. The first one is an identifier representing a consumer. The second one can be used to trigger either an „invitation is accepted" or an „invitation is refused" case. The PTC's behavior corresponds to the MSCs shown in Figure 6.

5. A TEST CAMPAIGN FOR AN RP-FACET

A test campaign for a RP-facet makes use of the structure at the RP (see also Section 3), i.e. it considers the dependence relation on operations, the causal relation on functionalities and the order relation on RP-facets. This results in a three-level test hierarchy for a RP.

Table 1. Test step for PTC_Invited

Test Step Dynamic Behavior **Test Step Name:** PTC_Invited (id: UserId; vote: Vote) **Default:** Invited_Default			
Nr	**Behavior Description**	**Constraints Ref**	**Verdict**
1	+SetupAccess(id)		
2	+GetRef(PCO_VotingReq, i_ProviderVotingReq)		
3	+CreateServant(PCO_PartyInd, i_PartyInd)		
4	+CreateServant(PCO_PartyInfo, i_PartyInfo)		
5	PCO_PartyInd ? inviteUserInd_req (uid:=inviteUserInd_req.uid)	inviteUserInd_req_r1	
6	PCO_PartyInd ! inviteUserInd_rpl	inviteUserInd_rpl_s1	
7	[id = uid]		
8	PCO_VotingReq ! voteReq_req	voteReq_req_s1(vote)	
9	PCO_VotingReq ? voteReq_rpl	voteReq_req_r1	(P)
10	[vote = ACCEPTED]		
11	PCO_PartyInfo ? inviteUserInfo_req	InviteUserInfo(id)	(P)
12	+ReleaseSession		
13	[vote = REFUSED]		
14	START testTimer		
15	?TIMEOUT testTimer		(P)
16	+ReleaseSession		
17	[id <> uid]		(F)
18	+ReleaseSession		

RP test groups are defined:

- per operation and will consider the availability and signature for an operation at an interface,
- per functionality and will consider the required and forbidden use scenarios for the functionality, and
- per RP-facet. They will consider the causal order of using the different functionalities of a RP- facets and will take into account the order relation with respect to other RP-facets.

Some test cases for operations are re-used by test cases for RP functionalities, which again are re-used by test cases for RP-facets.

Let us assume two RP facets F1 and F2 with F1≤F2, that F1 has been tested already, and that F2 has to be tested in addition. First, we determine the functionalities of F2 that are not part of F1. The additional functionalities use operations, which are tested first. The dependence relation on operations is considered here: in a first step, the independent operations are tested. Afterwards, those operations are tested whose preconditional operations

have been tested successfully. An operation for which one of the preconditional operations failed a test, is assigned an inconclusive test verdict. The tests on dependent operations are repeated until all of them have been tested or assigned an inconclusive test verdict.

In the next phase of testing, the functionalities are tested with respect to the defined use scenarios. Indeed, only those functionalities need to be tested whose operations have passed their tests. In the last phase of testing, the RP-facet as such is considered. Test cases that reflect the sequential and parallel use of functionalities at the RP will be executed. The conformance assessment for the tested RP-facet results from considering the individual test results of operations, functionalities and their combination.

For the basic test configuration given in Figure 7, a test should check the invitation functionality if more than two customers are involved. This is reflected by the creation of several test components of PTC_Inviting, PTC_Invited, and PTC_NotInvited. The parameterization for these test components is handled by the MTC, so that the right mixture of invitations, acceptances and non-acceptances of invitations will be tested. For those types of test, we propose to make use of the upcoming TTCN3 version, which will support dynamic configurations and a flexible handling on the creation and termination of PTCs. As the syntax of TTCN3 is not fixed yet, we do not give an example here.

6. CONCLUSIONS

RPs are an established concept to structure open distributed systems and for the definition of conformance requirements. Current RP specifications e.g. in TINA tend to be too large and complex, what negatively impacts their implementability and testability.

This paper discusses structuring principles of RPs into functionalities and RP-facets. The main concept of an RP-facet is that it is a self-contained portion of an RP, so that stakeholders may implement selected RP-facets only, while preserving a level of completeness for the realized subset of the RP. A combination of existing specification methods is used to support the formal definition of RP-facets. Further, the paper analyses different relations implied by these structures. It considers the dependence relation on operations, the causal relation on functionalities and the order relation on RP-facets. These relations are used to direct an efficient testing of RP-facets.

Indeed, looking at RPs and their structure as a basis for determining test cases for distributed systems opens new ways of handling the conformance testing. In fact, a number of issues are still open. For example, we will investigate the impact of different notions of dependability of operations or

the combination of use scenario based test cases in test configurations with several test components.

REFERENCES

[1] R. V. Binder: Testing Object-Oriented Systems, Models, Patterns and Tools, Addison-Wesley, 1999.
[2] M. Born, I. Schieferdecker, M. Li: UML Framework for Automated Generation of Component-Based Test Systems, Proc. of SNPD '00, France, May 2000.
[3] 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.
[4] GMD FOKUS: Final Subm. to TINA Conformance Testing Framework RFP, Mar 2000.
[5] M. Li, I. Schieferdecker, A. Rennoch: Testing the TINA Retailer Reference Point, Proc. of ISADS'99, Tokyo, Japan, Mar. 1999.
[6] ISO/IEC 9646-2: OSI Conformance Testing Methodology and Framework - Part 2: Abstract test suite specification, 1991.
[7] ISO/IEC 9646-3: OSI Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN), edition 2, Dec. 1997.
[8] ITU-T Rec. X.901 | ISO/IEC 10746-1: Information Technology - Open Distributed Processing - Reference Model: Overview, Aug. 1997.
[9] ITU-T Z.130: Object Definition Language (ITU-ODL), Mar. 1999.
[10] ITU-T Z.120: Message Sequence Charts (MSC'2000), Nov. 1999.
[11] OMG: Common Object Request Broker Architecture (CORBA), version 2.3, 1999.
[12] OMG: Telecommunications Service Access and Subscription, Joint Revised Subm., telecom/00-02-02, Feb. 2000.
[13] Schieferdecker, M. Li, A. Rennoch: Formalization and Testing of Reference Point Facets, Proc. of FMICS'00, Berlin, Germany, Apr. 2000.
[14] Steedman, D.: Abstract Syntax Notation One (ASN.1), Techn. Appraisals Ltd., 1990.
[15] TINA-C: TINA Reference Points, version 3.1, Jun. 1996.
[16] TINA-C: Ret Retailer Reference Point Specification, version 1.1, 1999.
[17] TINA-C: TINA-CAT WorkGroup Request for Proposals, TINA Conformance Testing Framework, Version 1.0, Jul. 1999.

---

1. The concept of segments is for specification purposes only. It has been developed in independently and in parallel with the RP-facet concept, which is also dedicated at conformance testing.
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.
3. Please note that we abstract here from the parameters in the operation invocation. Therefore, the dependence relation could be refined to cover further aspects of dependencies. For example, if operation o_2 is only executable when operation o_1 returns `x`, then o_2 can be defined to be result-dependent on o_1. This is for further study.
4. For readability reason, the description is simplified at some places, e.g. the usage of timers whenever a RECEIVE event is expected is not shown.