

REALISATION OF OPEN SYSTEMS

H.J. Burkhardt

Rapporteurs: G.D. Parrington
S. Pflieger

Realisation of Open System

H. J. Burkhardt

Gesellschaft für Mathematik und Datenverarbeitung
Rheinstrasse 75, D 6100 Darmstadt, FRG

This paper reports about work done within the GMD project PROSIT during the past years. The objective of PROSIT is to develop a consistent set of methods and tools for the realisation of open systems. This work was undertaken by Dr. B. Baumgarten, H. J. Burkhardt, Dr. H. Eckert, E. Faul-Luers, A. Giessler, Dr. P. Ochsenschläger, W. Orth, Dr. R. Prinoth, U. Faltin, Ch. Paule. It has benefitted from the ideas of many people and has been stimulated and inspired, especially by ideas born in the OSI-standarization committees.

1. Introduction

The objective of Open Systems Interconnection (OSI) is to enable heterogenous systems - i. e. are systems differing in origin and technology - to cooperate.

More specific, OSI aims at the establishment of an interworking technology which provides the means for the transfer of data between heterogenous systems and the elementary building blocks for the construction of a wide variety of distributed applications.

Following the OSI-approach, heterogenous systems get the capability to cooperate by observing a behaviour as defined in international communication service and protocol standards. Systems having this capability are called real open systems.

To create real open systems, three major steps must be done:

- communication services and protocols have to be specified by international standardization bodies,
- to accomodate their systems with interesting cooperation capabilities manufacturers have to implement functional groupings of standardized

- communication protocols,
- Protocol implementations have to be tested by authorized test centers to assure that a real system provides for the claimed openness and that it can therefore cooperate with each other real system having passed the same tests.

Each of these steps as well as the transitions among them involve problems.

To standardize communication services and protocols means to model distributed systems and describe them formally and in a strictly implementation independent manner. The problem of standardization is that no approved methodology exists for that purpose. Therefore, resulting standards are not necessarily complete, error free, consistent and unambiguously understandable.

Because of this deficiencies, current communication standards form an unsatisfactory basis for implementations. But even perfect standards would face implementors with the problem how to interpret the standards with respect to the reality constituted by their real systems. In other words, the problem of implementation is that no approved method exists to derive implementations from implementation independent protocol specifications, but that each difference in interpretation of a protocol standard may and usually will lead to incompatible implementations.

Because of this problems related with standardization and implementation of communication standards and because of the absence of a single authority which can take over the responsibility for the functioning of the cooperation among heterogenous systems, tests are regarded as necessary. But tests must not improve restrictions on protocol implementations, neither with respect to the communicational behaviour allowed by the related protocol specification, nor with respect to the choices of implementations. Therefore, test should be derived formally from protocol specifications and based purely on the outside visible implementation structure following from the claimed openness. The problem of test is that no approved method exists which fulfills these requirements.

This "state of the art" and the belief that the OSI-objective is worth being supported has motivated work within GMD on a systematic approach, called PROSIT, for realizing open systems. Its aim is to provide for all the steps (see Fig. 1) - leading from service specifications to compatible systems - improved methods and tools.

This paper outlines the basic ideas of PROSIT for modelling communication services and protocols and for specifying them formally, for constructing protocols from services and verifying them against the bordering services and for deriving implementation specifications and test sequences from protocol specifications.

2. Refinement of the OSI-Reference Model towards models for communication services and protocols

The backbone of the PROSIT approach is formed by a refinement of the OSI-Reference Model to models for communication services and protocols. In these models, the essential ingredients of the Reference Model are put into the appropriate perspective, thus providing an overall structure for object oriented specifications of OSI-communication services and protocols. Both, services and protocols are specified with the same formal description tool, a dedicated form of Predicate/Transition Nets which we call Product Nets.

A specification of a communication service defines all interactions between the users of this service and its provider as sequences of service primitives at discrete interaction points. Our specification conforms to this procedure but is not restricted to sequences related to a single connection. It regards all connections, thus describing the interdependencies which are caused by the usage of common resources among logically independent connections. Semantic aspects of a service are expressed by modelling the change of "consciousness" of interacting entities.

The formal specification of a communication service is a prerequisite for the development of a supporting protocol specification. Therefore, the modelling of a service which forms the basis of its specification, is adjusted

to support the construction of protocol specifications from service specifications. As a consequence, one logical approach for the development of a protocol model would be to superimpose two service models and to interrelate provider-behaviour of service N with user-behaviour of service (N-1).

The approach of specifying all interactions as concurrent, which can concurrently occur provides the implementation independence of service and protocol specifications which is desired for standardization purposes.

Each implementation of such a specification, therefore, appears as a valid restriction of concurrency, i. e. as sequentialization reasoned by the mapping onto real storages and processors.

At present, only logical relationships between interactions are defined in our specifications, but no relationships with regard to physical time.

2.1 Derivation and refinement of the OSI-notion entity

As a basis, let us consider the scenario depicted in fig. 2.1 wherein a service provider appears as counterpart of a set of service user instances.

A general task of the service provider is the establishment of communicational relationships among service user instances, e. g. with respect to "connectionless" interaction or connections maintained over an extended period of time.

A communicational relationship among service user instances is always based on an interaction between (1) a service user instance acting as originator and the service provider and (2) the service provider and the corresponding service user instance. All interactions between a specific service user instance and the service provider take place at their specific interaction point by means of service primitives.

The service provider, seen from the user's point of view as a black box, is itself a distributed system. This system is composed of multiple independent service provider instances. To each service user instance, a service provider instance is dedicated, in a static and strict one-to-one

relationship. Service provider instances cooperate following a peer-to-peer protocol in order to provide the service requested.

Adopting a term of the OSI-Reference Model, we call an interaction point by which a service user instance can communicate with a service provider instance a "Distinct-Service-Access-Point".

To model connection-oriented communication, it is necessary to differentiate static preassignments, which are prerequisites to the dynamic establishment and release of connections from the connections themselves. This leads us to refine both the service user and provider instance assigned to a distinct SAP into a static instance and dynamic multiple instances respectively (see fig. 2.2).

The dynamic instances associated with one SAP are created and destroyed by a static instance which exists permanently for this purpose. Dynamic instances are created during the connection establishment phase and destroyed during the connection release phase.

Over each distinct service-access-point, exactly one service-user-static-instance communicates with one service-provider-static-instance. Distinct connection-endpoints at a service-access-point represent the resources which the static instance of the service provider allocates to its respective dynamic instances on both sides of the interface boundary during each connection establishment. All instances attached to a distinct-service-access-point reside in the same OSI-system.

In this way, it may be said, that a service-access-point associates at its global address c (locally in each system) two static instances having the same name: one service-provider-static-instance c and its corresponding service user-static-instance c . Similarly, an allocated distinct-connection-endpoint (c, k) in a system associates locally two dynamic instances of the same name (c, k) , i. e. a service-provider-dynamic-instance (c, k) with its corresponding service-user-dynamic-instance (c, k) .

Consequently, there exist in a system the following interaction points between service-provider-instances and service user instances:

- for each distinct service-access-point, an interaction point between static instances, which is represented architecturally, by the term service-access-point itself, and across which are exchanged the service primitives related to the establishment and release phase of a connection
- and
- for each connection, an interaction point between a service-user-dynamic-instance and a service provider-dynamic-instance, which is represented architecturally, by the term connection-endpoint, and across which are exchanged the service primitives related to the data transfer phase of a connection.

To model connectionless data transmission, there is of course no need to use dynamic entities since no connections are involved. Consequently, connectionless data transmission has to be interpreted as interaction among static instances. This appears to be quite natural in recognition that a connectionless data transmission request at one distinct service access point, possibly followed by a connectionless data transmission indication at another distinct service access point is of the same nature as a connect request at one distinct service access point, possibly followed by a connect indication at another distinct service access point forming the first half of a connection establishment.

Keeping in mind OSI-objective of systems compatibility, we define an equivalence relation on the set of service user and provider instances which leads to the definition of OSI-service-user-entities and OSI-service-provider-entities as equivalence classes of service user and provider instances. These equivalence classes consist of instances which show identical communicational behaviour.

In the course of forming equivalence classes we compose (see Fig. 2.3)

- a static-service-user-entity consisting of static service user instances,
 - a static-service-provider entity consisting of static-service-provider-instances,
- and
- a dynamic-service-user-entity consisting of dynamic-service user-instances
- and
- a dynamic-service-provider-entity consisting of dynamic-service-provider instances.

In consequence, the OSI-concept "service-access-point" comprises the collection of distinct-service-access-points and the OSI-concept "connection-endpoint" comprises the collection of distinct-connection-endpoint. In other words, both notions define relationships between equivalence classes of instances. These equivalence classes of instances are called entities.

2.2 The PROSIT communication service model

The PROSIT communication service model consists of a refinement of the equivalence relation introduced above. This refinement is based on the differentiation between active and passive instances and leads to the notions of active and passive entities.

An active entity (service user and provider) is understood to be the collection of all calling instances (service user and provider), whereas a passive entity (service user and provider) is understood to be the collection of all called instances (service user and provider).

This refinement is necessary for making visible the boundary between OSI-Systems in addition to the service boundary. Therefore, it is possible to interpret a protocol which binds active and passive entities across the system boundary as a correlation of active and passive communications behaviour. The resulting communication service model interrelates eight entities (see Fig. 2.4):

- at one service-access-point, a static-service-user-entity, active with a static-service-provider-entity, active and a dynamic-service-user-entity, active with a dynamic-service-provider-entity, active,
and
- at the corresponding service-access-point a static-service-user-entity, passive with a static service provider entity, passive and a dynamic service user entity, passive with a dynamic service provider entity, passive.

The establishment of a connection appears within that model as the creation of the four appropriate dynamic instances, as mentioned above, and, due to interaction among four corresponding static instances, one for each static entity mentioned above.

The release of a connection appears correspondingly as deletion of the four dynamic instances previously created.

Connectionless data transmission appears within that model as interaction among four corresponding static instances, one for each static entity mentioned above.

N-way data transmission can best be interpreted as connectionless data transmission where a multicast address is used as To-Address.

This has the effect that a connectionless data transmission request, issued by a static service user instance to its corresponding static service provider instance at one distinct service access point, leads to a connectionless data transmission indication at each distinct service access point, which is a member of the set of recipients named by the multicast address.

2.3 The PROSIT communication protocol model

The PROSIT communication protocol model is based on the interrelation of two communication service models at adjacent layer boundaries (see fig. 2.5).

It is therefore hierarchically structured into the following three functional levels:

- the N-service user level comprising the N-service user entities of the model for the N-service,
- the N-protocol level comprising the N-service provider entities of the model for the N-service, the (N-1)-service user entities of the model for the (N-1) service, the functionality of mapping N-service provider behaviour onto (N-1) service user behaviour and the functionality of bridging the gap between N-Service and (N-1) service ; the OSI term N-Layer corresponds to the N-protocol level.
- the (N-1) service provider level comprising the (N-1) service provider entities of the model for the (N-1)-service.

This protocol model implies that an N-Layer is to be understood as two inter-related protocol entities, namely an active N-protocol entity and a passive N-protocol entity.

Each protocol entity is composed of two N-service provider entities and two (N-1)-service user entities.

In this refined model of the protocol entity at the Layer N, it is apparent that:

- an N-connection can be released by destroying the related dynamic N-service user and provider instances without releasing the supporting (N-1) connection;
- an (N-1) connection can be released by destroying the related dynamic (N-1) service user and provider instances without releasing the supported N-connection.

It should be noted further that this refined model of the protocol entity of layer N allows to express any mapping of connection oriented or connectionless services at the boundary between layer (N + 1) and layer N onto connection-oriented or connectionless services at the boundary between layer N and layer (N - 1).

In (1) is illustrated the use of the PROSIT communication service and protocol model for structuring service and protocol specifications.

3. Product nets - a PROSIT-tool for the specification of communication services and protocols

The PROSIT communication service and protocol model establish logical relationships between roles. Each box in Fig. 2.4 and 2.5 represents a distinct role and the arcs between boxes stand for the logical relationships. Each role defines communicational behaviour and is the characteristic of a set of instances. All instances belonging to such a set behave in communication as defined by the role. The behaviour of each instance could be defined by a

finite automaton, the state transitions of which represent interactions with other automata.

Therefore, each box in the service and protocol models can be interpreted - from a specification point of view - as representing a set of finite automata. The complete models can be seen as collection of sets of finite automata communicating via channels.

These models are characterized by containing explicitly:

- aspects of distribution,
- concurrency of actions,
- global features of cooperating systems,
- moduls, channels, interfaces, interactions,
- folding and parameterization of interactions.

Product Nets allow to express all these aspects in a unique manner. Especially in its graphical representation, they are highly adequate for the design of distributed systems.

Roughly speaking, a product net is a Petri net combined with arc labels and transition inscriptions, which are as formally constructed as this is done in case of terms, predicates, atoms and formulas in the first-order logic. In first-order logic an interpretation of a formula is a well known concept. By introducing adequate restrictions with respect to the syntax of arc labels and transition inscriptions, the formal semantics of nets is derived from the interpretation of arc labels and transition inscriptions, whereby a domain of objects is assigned to each place. A formal specification of product-nets can be found in (2), an illustration of its use in (1).

To illustrate some fundamental concepts of product nets, the flow controlled transfer of data from a sending user of a transfer service to a receiving user is modeled (see Fig. 3.1). This transfer leads from the sending user over a service boundary to the sending provider. The sending provider communicates over a system boundary with the receiving service provider. At least, the receiving service provider delivers received data to the receiving service user.

The product net in Fig. 3.1 has the initial marking:

s1 : a set of messages s8 : W $\langle \rangle$ -tokens
s4, s5 : one $\langle \rangle$ -token s9, s10 : $\langle 0 \rangle$

The other places are empty.

The places s3 and s4 represent the service boundary between the sending service user and the sending service provider and s5 and s6 represent the service boundary between the receiving service provider and the receiving service user. The 'ready signals' on s4 and s5 guarantee that not more than one message appear at S_3 and S_6 . The places S_7 and S_8 represent the system boundary.

Receiving the messages in the order, they are sent, shows up as a consistent numbering in the sending and the receiving service provider. The flow control is enforced by 'ready signals' on s8.

4. The basic thoughts behind the construction of protocols from services and the verification of protocols against services

The formal specifications of communication services adjacent in the OSI-RM form the prerequisites for the development of a protocol specification, since a protocol rules the cooperation of layer entities which has to bridge the functional difference between adjacent services.

In PROSIT, the construction of a protocol specification from service specifications is supported on one hand by the modelling approach and on the other hand by the use of elementary building blocks and rules for their composition.

The modelling approach support the construction since the PROSIT communication protocol model, which forms the basis of a protocol specification, is constituted by superimposing two service model, which form the basis for the service specifications. A protocol specification of layer N, therefore, consists of

- provider behaviour of service N and of user behaviour of service (N-1) - both already specified as part of the specification of service N and service (N-1);
- and a part which must be added to interrelate provider and user behaviour and to bridge any functional gap between the two services.

The seek for elementary building blocks and rules for their composition is motivated by the recognition that in different context always the same communication patterns occur. Such pattern are in distributed systems e. g. typical for the consistent establishment, progression and release of global context, which is just another formulation for cooperation. The flow control example shown in Fig. 3.1 is another example.

The usage of building blocks for the design of distributed systems is shown in (3) on the example of a Product Net specification of the OSI-Transport Service and (4) on the example of a Product Net specification of a distributed application.

The use of the communication service and protocol models and of building blocks and composition rules facilitates the formal specification of services and protocols, by providing constructive aids, and increase their readability.

Beside the design, the validation of communication services and protocols and the verification of protocols against the bordering services benefit from this approach.

Validation means to check services and protocols whether they have desired properties (e. g. freedom of deadlocks and lifelocks, globally consistent binding and releasing of resources).

Verification of a protocol against the bordering services means to prove that a protocol provides the service above while using correctly the service below.

To verify a protocol, one has to compute from each service specification the complete set of sequences of services primitives. These sequences can be derived from the product net specifications of the services as set of sequencing of markings at places representing the service boundary.

The same procedure has to be applied on the places representing upper and lower service boundaries in the protocol specification. This results in a set of sequences of markings at places on the upper service boundary and a set of sequences of markings at places on the lower service boundary.

Due to the constructive relationship between protocol model and service model, one can regard a protocol as verified if the sets of sequences of markings derived from the service specifications coincide with the respective sets of sequences of markings, derived from the protocol specification. In (5) this verification method is demonstrated on the example of the alternating bit protocol. In (6) a checkpoint-restart-protocol is proved and in (7) the PROSIT net simulation system NESSY is described.

5. The basic thoughts for deriving implementation specifications and test sequences from global protocol specifications

A protocol specification, structured according the PROSIT communication protocol model and formally specified by means of Product Nets, defines cooperation in a distributed system. Roughly spoken, it identifies all communicating instances and describes how they interact. This view of description can be called the "designers view".

For an implementation, a specification is needed which defines - roughly spoken - a single instance and how it acts or reacts at its interaction points with the outer world. This view of description can be called the "implementors view".

For the test of a protocol implementation on conformance with the protocol specification, one needs a record of all events which might occur at boundaries accessible in a test. This view of description can be called the "observers view".

It is quite obvious that the designers view of description contains the implementators view as well as the observers view and that it must be therefore possible to derive an implementation specification and test sequences from a protocol specification.

The global implementation independent protocol specification identifies all protocol functions and establishes their logical relationships, only. It is therefore characterized by a high degree of concurrency. In contrast, an implementation specification deals with tasks and their interrelationships.

A task is understood as a functional unit,

- which can be formally described by a finite automaton
 - which is allocated as a whole to one local processor
- and
- which is the basic unit in an operating system being provided with system resources.

The derivation of implementation specifications from global protocol specification, therefore, comprises

- as first step the assignment of protocol functions to tasks,
- as a second step the serialization of concurrent protocols functions assigned to one task,
- as a third step the logical coupling of tasks.

The method for the derivation of implementation specification from product nets Specification of Protocols is described in (8).

For implementation specifications, CCITT's SDL is an accepted and widely used language, which is supported by software development tools of various organizations.

It seems feasible to bridge the gap between a global protocol specification in form of product nets and an SDL-implementation specification and to combine the benefits of product nets for formal description of global systems and their mathematical analysis with the SDL support for implementations.

Conformance testing is based on test sequences or more generally, test suites which should allow comparability and wide (international) acceptance of test results.

In this area the ISO ad-hoc group on conformance and conformance testing has defined a set of abstract test methods and has provided a framework for specifying conformance test suites.

The following three main categories of abstract test methods shown in Fig. 5.1 were identified with respect to the kind of observable and controllable interfaces within the implementation under test (IUT) or system under test (SUT). The applicability of different abstract test methods to a particular SUT depends on the accessibility of interfaces within the SUT.

The local test methods use control and observation of the (abstract) service primitives defined at the service boundaries directly above and below the implementation under test.

In this case IUTS are tested in isolation from the rest of the system which is simulated by the upper and lower tester. Test synchronization (test coordination procedures) between activities of the upper tester and activities of the lower tester are needed for these methods.

The distributed test methods use control and observation of the (N-1)-service primitives defined at the service boundary between the lower tester and the (N-1)-service provider and control and observation of the (N)-service primitives defined at the service boundary between the IUT and the upper tester. These methods also require the use of test coordination procedures between the upper tester and the lower tester.

The remote test methods use control and observation of the (N-1)-service primitives defined at the service boundary between the lower tester and the (N-1)-service provider.

These methods provide the weakest tests in terms of control and observation since no interface within the SUT is accessible.

The main difference between the local test methods and the two other categories of test methods is that a (N-1)-service provider is only simulated in the case of local testing whereas a real (N-1)-service provider is involved in the

case of distributed and remote testing. Therefore, the local test methods require further testing, since the lower tester can only be a more or less adequate approximation of a real (N-1)-service provider.

Finally it should be mentioned that the distributed and remote test methods are defined on the assumption that the (N-1)-service provider behaves correctly. This assumption must be validated by the execution of basic interconnection tests which are part of the complete conformance test suite.

Formal protocol specification, structured according to the PROSIT protocol model and described by means of Product Nets, offer an enriched structuring of entities, but are in full accordance with the ISO test methods.

Formal protocol specifications define correct actions and correct reactions of instances. The correct reactions comprise correct reactions on receipt of correct pdus and invalid pdus which are either syntactically invalid or out-of-sequence pdus. Correct actions, however, do only consist of actions which describe the generation and sending of valid pdus. Formal protocol specifications per se do not include definitions of incorrect actions. Therefore, those further specifications are required for the purpose of testing (error generator) and must be added to the global protocol specification in order to get a complete test specification.

The procedure to derive test sequences from formal protocol specifications includes the following main steps:

- superposition of the protocol specification with a test configuration and identification of those interfaces within the SUT which are observable or controllable during the test (shown in Fig. 5.2 for the local test method);
- use of the formal protocol specification (specification language plus tools) in order to compute
 - the complete state space at the considered interfaces;
 - the complete set of state transitions at the considered interfaces;
 - the set of allowable sequences of transitions at the considered interfaces;
- definition and selection of appropriate test strategies.

The reachability analysis produces the complete reachability graph representing the behaviour of an SUT at the considered interfaces. This graph contains all possible and allowable test sequences which can be observed or controlled at the interfaces. Therefore, this set of test sequences is considered as being complete from the theoretical viewpoint. Thus it provides a measure for the quality of a concrete set of test sequences which is selected for a particular test and which might be a subset of the full set. Test strategies are based on a superposition of the complete reachability graph). Different test strategies are possible and so far the selection mechanism of defining an appropriate test strategy is purely intuitive and pragmatic. Test strategies are used for the following purposes:

- systematic selection of individual test patterns;
- systematic structuring of test (separation into test phases, ordering of test, specification of test events, test steps, test groups and test suites);
- definition of quality measures for testing;

The derivation of test sequences from Global protocol specification is described in (9).

6. Conclusion

The realisation of open system means to realize distributed systems in a distributed manner.

Together with the implied systems heterogeneity, this leads to a new category of problems to which established methods oriented on the design and programming of single systems offer no adequate solutions.

This new category of problems is characterized by the need to differentiate clearly between various levels of abstractions and to provide for consistent transitions between them; of course, such a need was formulated before in the context of single or homogenous systems design, but was there not of such a crucial importance.

At the highest level of abstraction, communication and cooperation in organizational structures have to be described. This description has to be based on a model deduced in an abstraction process from the existing or projected reality. This model has to express the aspects, only, relevant for cooperation; i. e. it has to emphasize upon which function an instance performs in their cooperation with other instances and not how it is internally conditioned to fulfill this function.

At the lowest level of abstraction, this 'how' has to be described for a distinct real open system.

The gap between these levels of abstractions must be bridged. A common understanding, what adequate modelling means, must be established and a consistent sets of method and tools for the various levels of abstraction and the transitions between them must be developed.

What has to be done in the end, is to develop the construction of distributed systems, especially the construction of distributed applications towards an engineering science.

Currently, we are far from this end, but our experience in applying PROSIT methods within industrial cooperations have given us the confidence that this end can be reached.

Reference to papers presenting PROSIT results in more details:

- (1) Burkhardt, H. J.; Eckert, H.; Prinoth, R.
Modelling of OSI-Communication Services and Protocols
using Predicate/Transition Nets
Protocol Specification, Testing, and Verification
Y. Yemini, R. Strom, and S. Yemini (ed.)
Elsevier Science Publishers B. V. (North-Holland)
C IFIP, 1985
- (2) Eckert, H.; Prinoth, R.
Produktnetze - Definition eines PROSIT-Beschreibungsmittels
Arbeitspapiere der GMD Nr. 92
- (3) Baumgarten, B.; Ochsenschläger, P.; Prinoth, R.
Building Blocks for Distributed System Design
Proceedings of the IFIP WG 6.1
Fifth International Workshop on Protocol Specification,
Testing and Verification, 1985
North Holland, 1985
- (4) Baumgarten, B.; Burkhardt, H. J.; Ochsenschläger, P.; Prinoth, R.
The Signing of a Contract - a Tree Structured Application
Modelled with Petri Net Building Blocks
Arbeitspapiere der GMD, Nr. 161, 1985
- (5) Eckert, H.; Prinoth, R.
A Computation-Systems Based Method for Automated
Proving of Protocols Against Services
Protocol Specification, Testing, and Verification, III
H. Ruding and C. H. West (ed.)
Elsevier Science Publishers B. V. (North-Holland)
C. IFIP, 1983

- (6) Baumgarten, B.; Ochsenschläger, P.
Modelling and Verification of a Checkpoint-Restart-Protocol
2. GI/NTG/GMR-Fachtagung: Fehlertolerierende Rechensysteme
Bonn 1984
Informatik-Fachberichte 84, pp 353-363
Springer Verlag
- (7) Paule, C.
Das Netzsimulationsystem NESSY
Arbeitspapiere der GMD Nr. 156, 1985
- (8) Faul-Luers, E.; Prinoth, R.
Ableitung von Implementationsvorgaben aus
modularisierten Produktnetzen
Arbeitspapiere der GMD Nr. 123, 1984
- (9) Burkhardt, H. J.; Eckert, H.; Giessler, A.
Testing of Protocol Implementations
- A Systematic Approach to Derivation of Test Sequences
from Global Protocol Specifications -
Proceeding of the IFIP WG 6.1
Fifth International Workshop on Protocol Specification,
Testing and Verification, 1985
North Holland, 1985

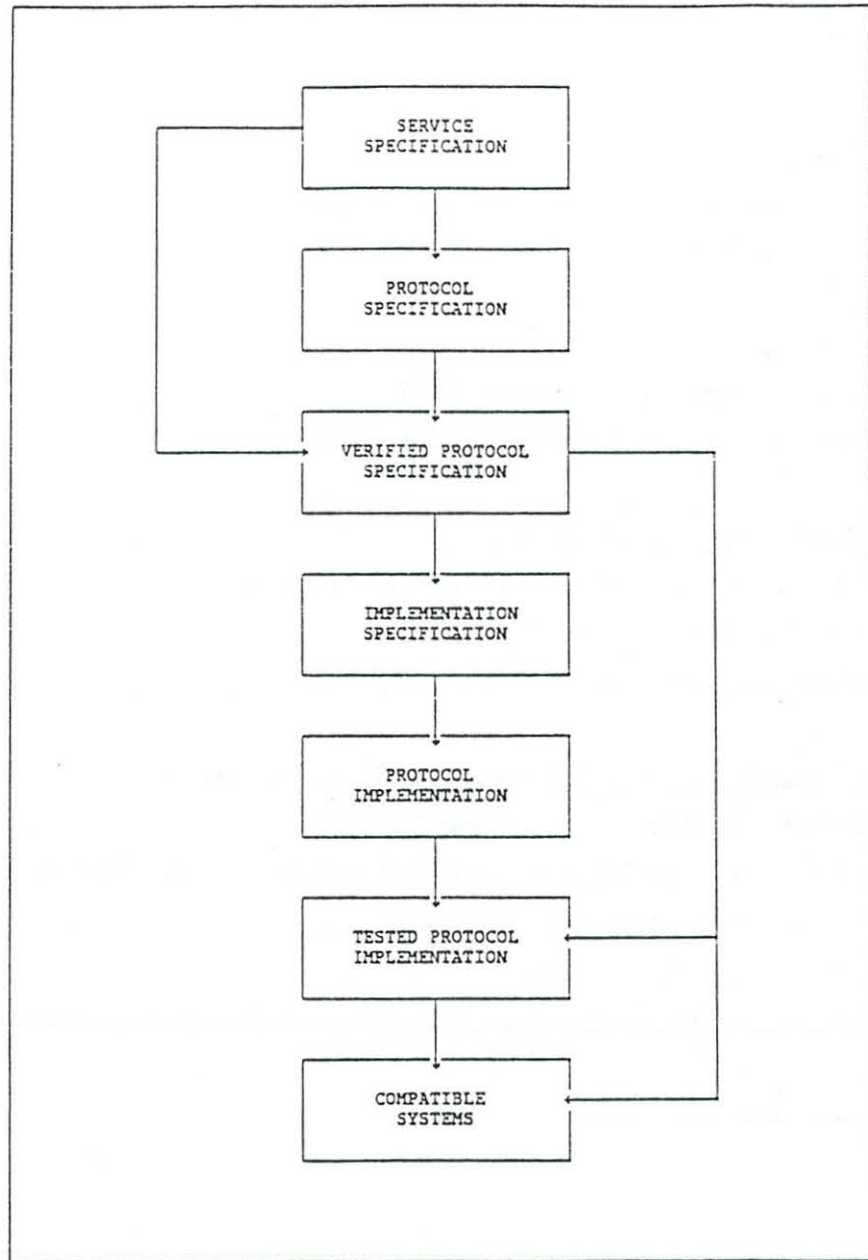


Fig. 1.1 : The intermediate steps necessary for the realization of compatible systems from standards

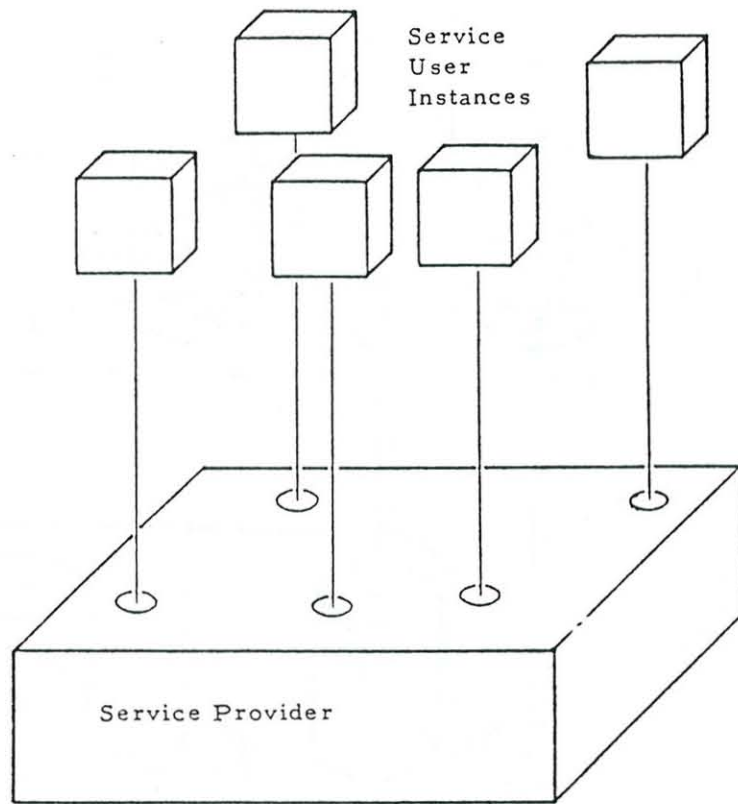


Fig. 2.1 : The general Scenario for a Communication Service in the Sense of the OSI-Reference Model

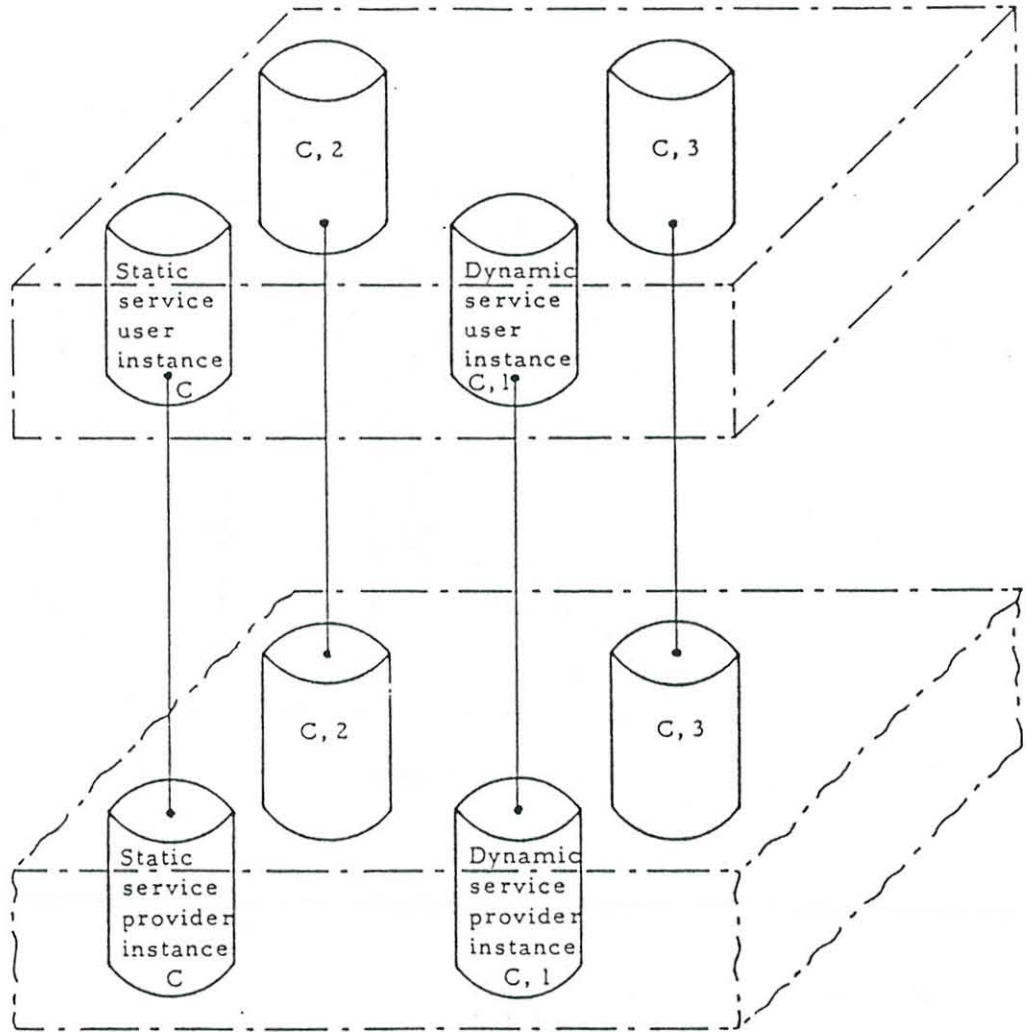


Fig. 2.2: The Refinement of both Service User and Provider Instances at a distinct SAP with Address C into Static Instances C and Dynamic Instances C,K; K = 1 to 3

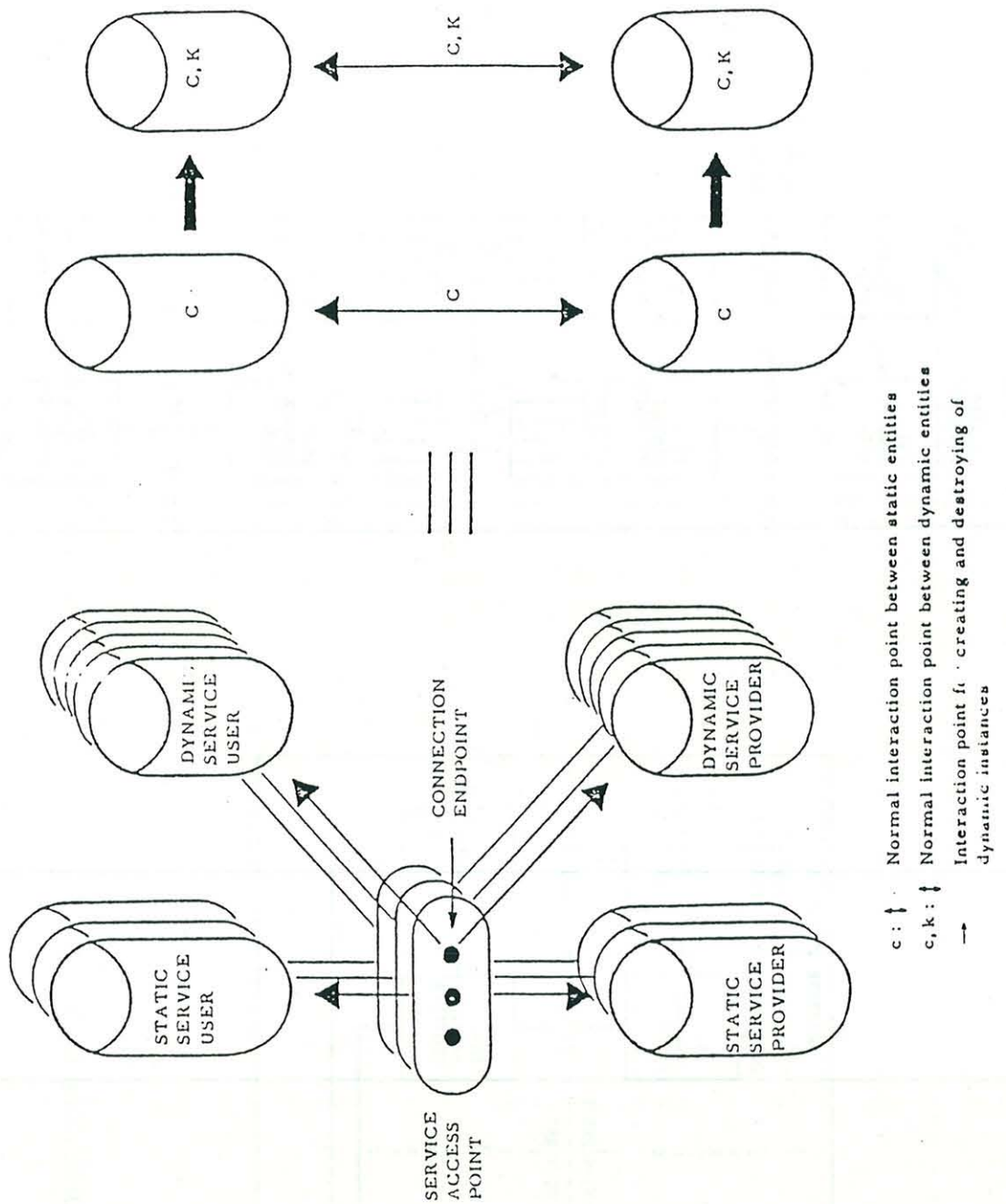


Fig. 2.3: The refined model of a Service Access Point introducing the notions of static and dynamic entities.

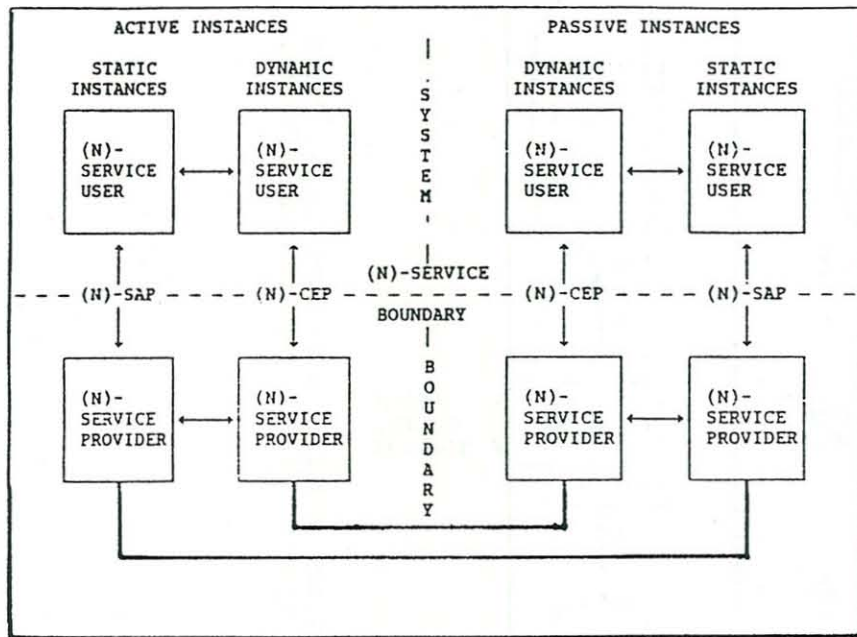


FIG. 2.4: THE PROSIT COMMUNICATION SERVICE MODEL

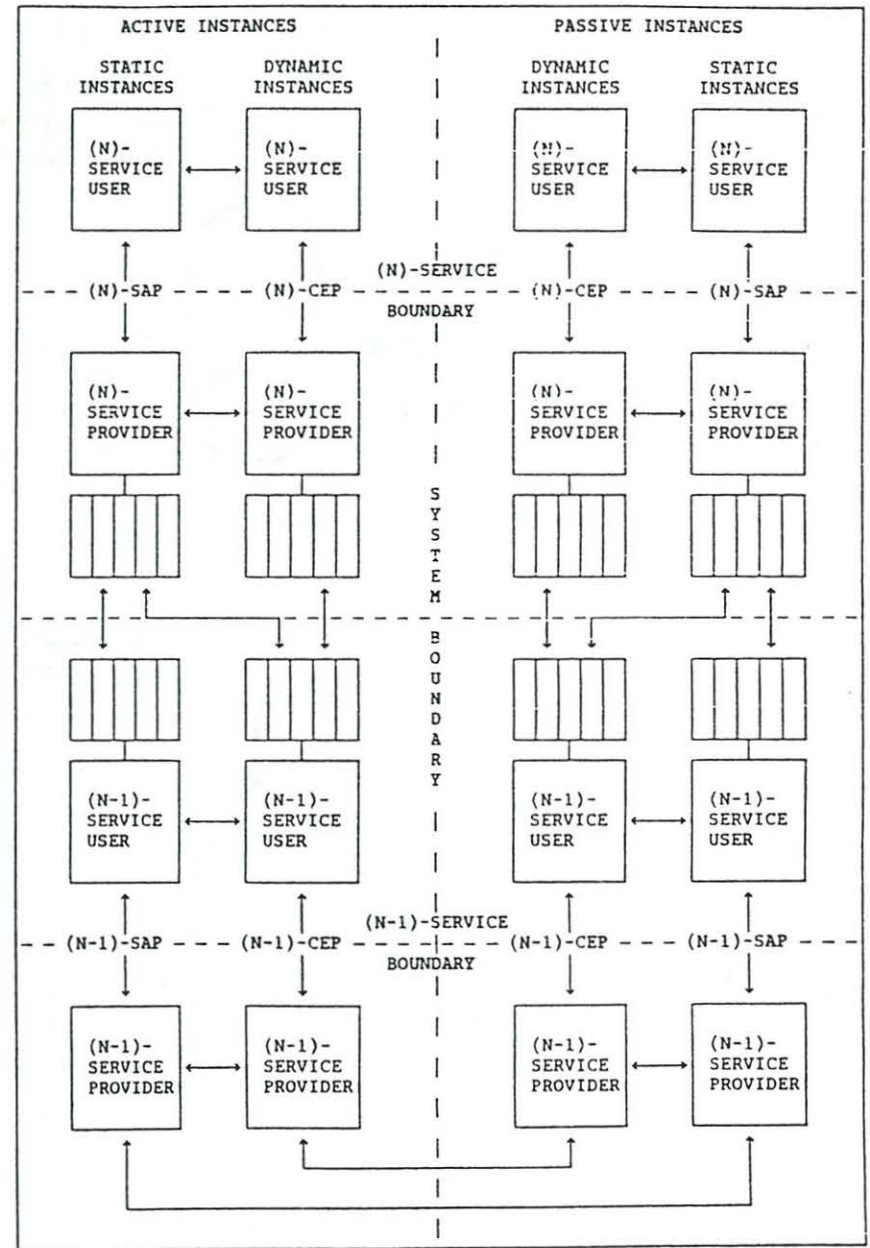
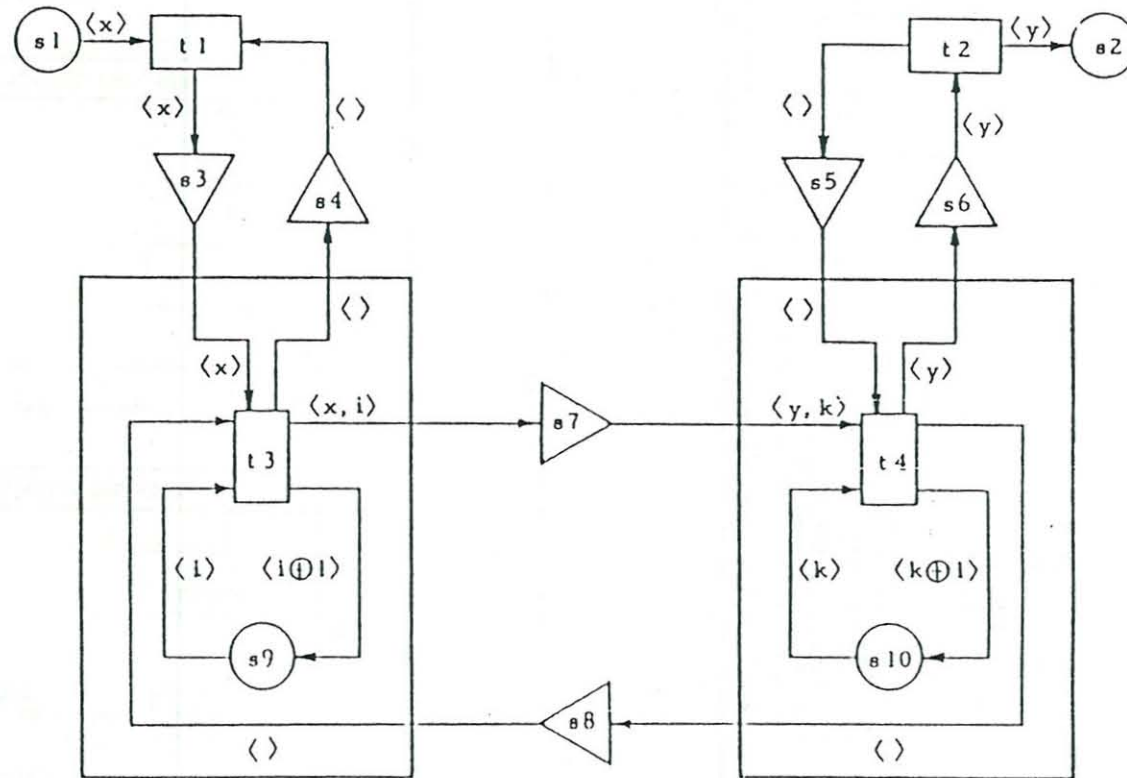


FIG. 2.5: THE PROSIT COMMUNICATION PROTOCOL MODEL



\oplus denotes addition modulo W . W is the capacity of the queue ('window size').

FIG. 3.1: PRODUCT NET SPECIFYING A FLOW CONTROLLED DATA TRANSFER SERVICE

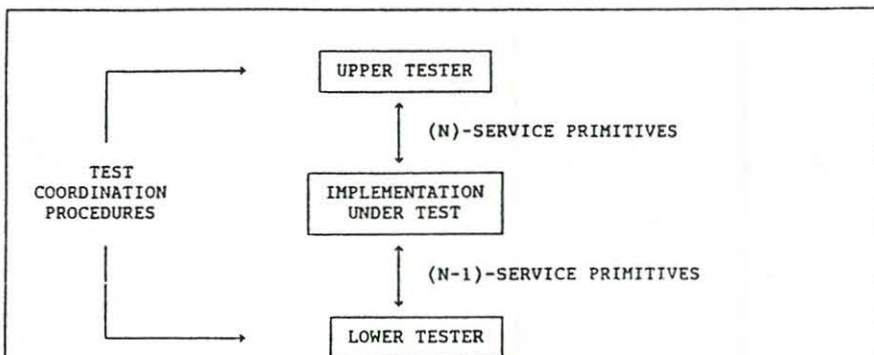


Figure 2.a: The local test methods

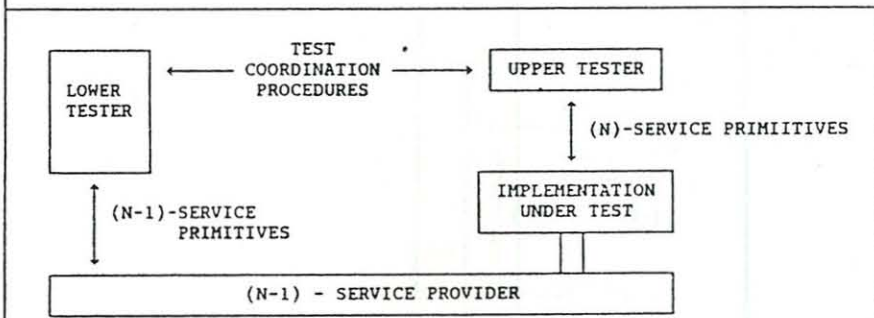


Figure 2.b: The distributed test methods

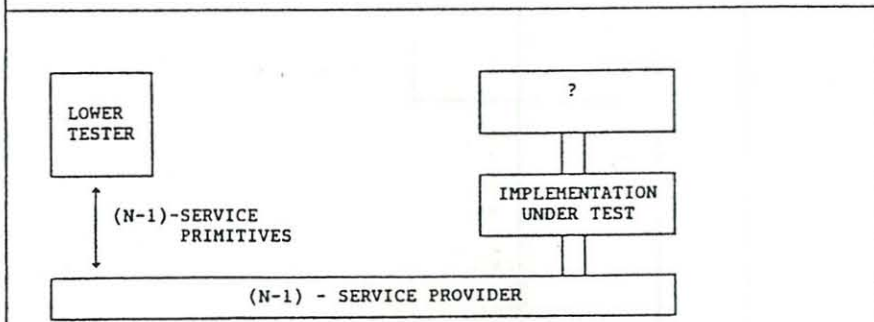


Figure 2.c: The remote test methods

FIG. 5.1: THE OSI TEST METHODS

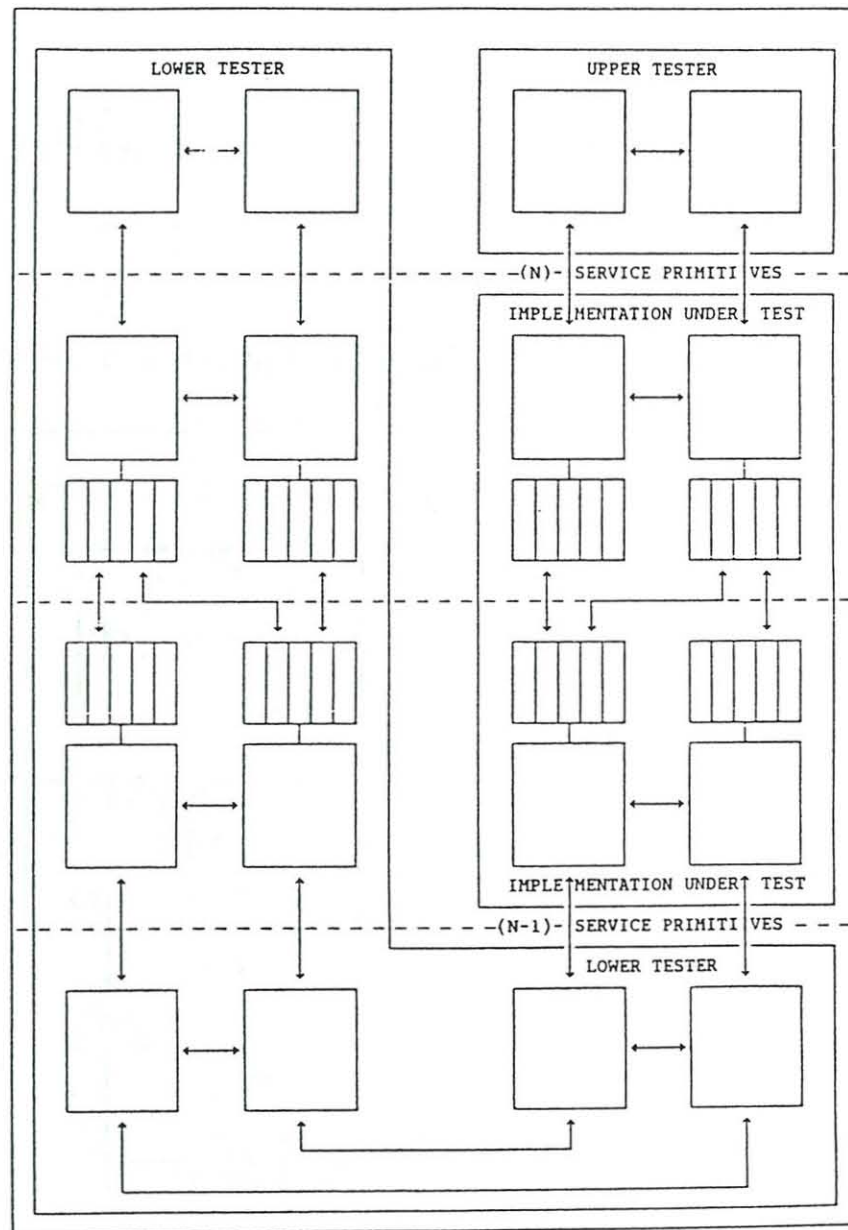


FIG. 5.2: RELATIONSHIP BETWEEN THE PROSIT COMMUNICATION PROTOCOL MODEL AND THE OSI LOCAL TEST METHOD

DISCUSSION

Professor Randell remarked that even in heterogeneous distributed systems, an "esperanto" communication language is not an efficient solution if similar computers communicate.

Dr. Burkhardt suggested that even in this case the "esperanto" language should be used.

Dr. Cohen drew a parallel between a possible measure of open-ness of a distributed system (e.g. this system is 93.4% open) and quality measures for programs (e.g. this program has 21 GO TO statements) and wondered if it is intended to define such an "open-ness" measure.

Dr. Burkhardt answered that no system is "open" or "closed" and the large spectrum of other possibilities (e.g. open for one type of system and closed for another type of system) should make very difficult the task of the definition criteria for such an open-ness measure very difficult.

Several questions about the interpretation of the presented USER COMMUNICATION MODEL and the active/passive role of the service initiator/service provider turned the discussion to other similar models:

- the client-server model,
- sender-receiver in the file transfer transfer model (Professor Tanenbaum), and to the definition of the terms active/passive (Professor Wells).

Dr. Zimmerman suggested that the terms "active/passive" should not be used and Professor Randell pointed out that the model presented is an interaction model only.

Dr. Cohen mentioned that the processing layers (layers 5,6,7) contain parameters for expressing the communication quality and only the communication layers (layers 1,2,3,4) are application independent.

Professor Randell suggested that one could see the relationship between processing and communication layers as independent in "one-way".

Dr. Burkhardt agreed with the term "one way" - independency.

