THE IMPORTANCE OF THE SERVICE CONCEPT IN THE DESIGN OF DATA COMMUNICATIONS PROTOCOLS [1]

Chris A. Vissers (1) and Luigi Logrippo (2)

(1)Department of Informatics
Twente University of Technology
7500 AE Enschede,
Netherlands

(2) Protocol Research Group
Department of Computer Science
University of Ottawa
Ottawa. Ont. Canada KIN 9B4

Abstract This paper analyses the level of recognition that the service concept has acquired in the world of protocol designers. Opposition against the concept and some significant cases of misuse are expounded and refuted. The paper argues for an increased role of the service concept, and its underlying architectural concepts, as the proper bases for designing protocol systems as well as suitable specification, verification, and testing techniques.

1 Introduction

Although the emphasis of the Workshop on Protocol Specification, Testing, and Verification naturally lies on the role of (formal) description techniques, and related verification and testing methods, the subject of this paper is architecture. The reason for this is the concern of the authors about the persistent lack of clarity about some vital architectural concepts underlying protocol systems. Whereas these architectural concepts ultimately determine the semantics of the subject of specification, testing or verification, lack of clarity about these concepts ultimately will lead to divergent interpretations about what a specification (technique) is about to express, and thus may lead to incompatibilities in protocol systems.

An important architectural concept is the service concept, and its derived topics such as service primitives, primitive parameters, etc. In spite of the fact that it has been well known for a number of years, the concept of service is still facing opposition and misunderstanding in the community of data communications protocols specialists.

It is not unusual to see or hear statements that show ignorance or misunderstanding of the concept, and still today data communications systems are being designed without having a clear insight in its architectural semantics and role. This observation cannot be better evidenced than by the fact that just recently (Feb. 1965) the second ISO Draft Proposal for an international standard on "Service Conventions" [OSC] was rejected because the defined service concepts were considered incomplete, inconsistent, contradictory and/or controversial. This rejection took place even in the light of the fact that several ISO service standards are referring to it. For this reason the service conventions document will appear as a Type 2 Technical Report until new work on the issue will bring more insight and consensus.

[1] This paper appeared, in slightly different form, in M. Diaz (Ed.) Protocol Specification, Testing, and Verification, V. (Proc. of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification) Elsevier-North-Holland, 1986, 3-17.

The authors believe that the concept of service has been a major advance in the theory and practice of data communications systems, and that its ignorance or misuse is very likely to lead to poorly designed protocols. On the other hand, they have noted that, while there are already several excellent early papers that explain the concept [BOC] [SCH] [BUS] [BOS] [SUN], there is a lack of recent papers dedicated to explaining its importance, to describe its uses in protocol design, and to review the current thinking on the subject. This paper sets for itself the goal of filling this gap. Hopefully, it will convert or inform some of the people who oppose or ignore the concept, and it will provide some points for reflection even to people who fully understand and use it. At least this paper can help to perform the role of systematically ordering the different perspectives on the service concept along which discussion can be organized and consensus can be reached. Undoubtedly it will cause some controversies, a result that the authors expect and encourage.

In Section 2, we provide an explanation of the concept of service. We define it, and discuss some implications of our definition. In Section 3, we analyze the opposition to the service concept. We list points of view we have read about, others we have heard, and others yet we feel we have good reasons to suspect. In Section 4, we review what we consider some common misunderstandings of the concept of service, especially by presenting examples of what in our opinion is misguided utilization of the concept. In Section 5, we provide some general reasons in favor of the concept of service. We show that it is useful from several, mostly closely related, points of view. On the basis of these general principles, in Section 6 we refute one by one the reasons for opposition to the concept that were mentioning in Section 3. Finally, in Section 7 we draw our conclusions, and we list our main theses in point form.

2 The Service Concept

The most commonly used model for protocol specification shows a number of protocol entities, the "service users", communicating with each other via another entity, the "service provider". This communication between protocol entities follows strict rules, called the "protocol".

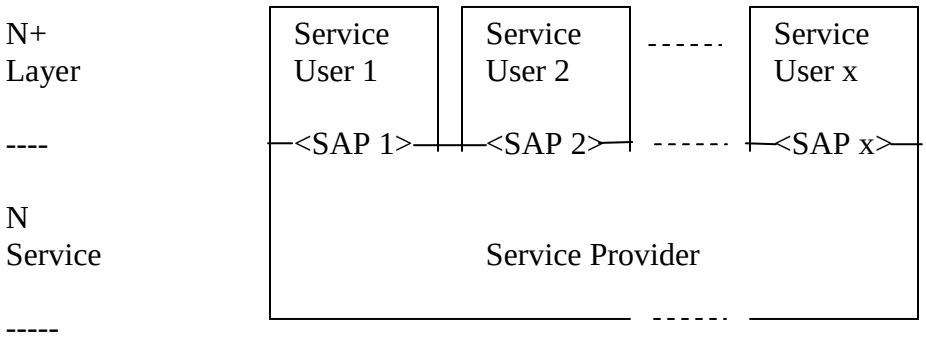


Figure 1: *Model of Service*

In order to communicate via the service provider, a protocol entity utilizes so called "service primitives". A service primitive can be considered as an elementary interaction between a service user and the service provider during which certain values for the various parameters of the primitive are established to which both user and provider can refer. The interactions are executed at the "common boundary" of a service user and the service provider, called a "Service Access Point" (SAP).
Since these SAPs represent internal boundaries in real world systems, service primitives must be defined and expressed at a high(est) level of abstraction, in order not to constrain valid implementations. This

implies that the implementer may use any mechanism that he finds useful to realize the execution of the service primitives such as procedure calls or hardware interfaces.

A Service Provider, therefore, is seen as an abstract machine accessible from a number of Service Access Points (SAPs). Execution of a service primitive at one SAP usually invokes the execution of another service primitive at another SAP whose parameter values may depend on the parameter values of the invoking primitive. The abstract machine is also capable of spontaneous "internal actions" leading to the execution of service primitives at SAPs. Therefore the specification of a service can be expressed in terms of the possible orderings of service primitives and their parameter value dependencies. Since this way of specification does not reveal any internal structure of the service provider it is often referred to as an observational, or extensional, specification: it defines the behavior of the provider as it can be observed by the users.

It should be noted here that a formal discussion of the concept of service requires the introduction of a formal model, and therefore is beyond the scope of this paper. The reader is referred to [BLP] [BO2] for a more formal discussion.

The protocol entities in Figure 1 can be considered as a layer of functions on top of the service provider. In multi-layered protocol architectures. such as OSI, a service provider can itself be made up of (N)-protocol entities which communicate by using another "lower-layer" (N-1)-service provider. Execution of the (N)-protocol by these (N)-protocol entities by using the (N-1)-service realizes the (N)-service. In other words, an (N)-service can always be described as the result of the combined action of the (N-1)-service and the (N)-protocol. Probably this way of describing the N-service should be called the N-protocol specification as it provides a much better architectural basis for a protocol specification and verification than the rather vague concepts currently in use. In contrast to the above mentioned extensional approach this way of service specification is called an intentional or generative technique as it reveals internal structure of the service provider.

This above decomposition can be repeated for the (N-l)-service. etc., and thus yields a layered protocol system with a set of nested services. (Note that the OSI model, apparently, does not recognize the medium as a service.) Therefore, we also find the following definition: "the (N)-service specification defines the global behavior of the (entities within the) layers below the (N+l)-layer as observable by the (entities within the) (N+l)-layer" [GUI]. Similar definitions are encountered in [BOC] [SCH], etc.

The above implies that there are in principle two ways of specifying the service:

- the extensional approach which we will call the (N)-service specification, and
- the intentional approach, which we will call the (N)-protocol specification.

Happily, but also necessarily, the (N)-service can often be described in much simpler ways than the (N)-protocol, as we will show by two examples.

As a first example, consider two (N+l)-entities that exchange streams of messages. The (N)-service provider they need for this exchange may be one that can be described in very simple terms, for example as a simple channel that does not lose or "invent" messages. The (N)-protocol needed to realize this simple service may, however, be quite complex, if for example the (N-l)-service provider can lose messages, or create spurious ones.

As a second example, suppose that the two (N+l)-entities need a (N)-service provider that, in addition, does not reorder messages, and has some buffer capacity. Again, this service provider can be simply described as a FIFO queue. Suppose, however, that this service is to be realized by using (N-1)-service providers that can reorder messages. An obvious (N)-protocol to realize the desired (N)-service is one that checks the order of messages by numbering them in some appropriate way. Buffering and/or retransmission mechanisms can then be used to recover from out of order messages.

In both examples, we see that a simple service description "masks" a variety of rather more complex protocol descriptions. One of the main theses of this paper is that a characteristic of well-designed data communication systems is exactly that the description of a service can be made much simpler than the combined description of the underlying protocols and services.

It would probably be quite difficult to trace back a history of the service concept. The idea as we know it today seems to be the result of committee discussions that also led to the formulation of the OSI Basic Reference Model [BRM]. Certainly the idea has grown, and is still growing, in maturity. The first ISO paper that introduces a more precise definition of the concept seems to be [GUI], dated June 1980. This paper also uses a picture showing a box surrounding lower layers as we do in figure 1, whereas [BRM] still does not contain this picture and talks about a service as a capability of a layer, and the layers beneath it etc.

Bochmann introduces the concept in his 1979 book [BOl p. 96], which however was written in 1977/8. Vissers defines the concept, but uses the term connecting architecture, in a 1976 paper [VI1], and shows a complex example in his 1977 thesis [VI2]. Sunshine discusses the concept in his 1979 paper [SUN], however he refers to pre-existing ISO work. The inspiration was provided by the methodology of software layering developed within Software Engineering [DIJ] [PAl] [PA2]. Clearly, this idea fell on a particularly fertile ground and a particularly congenial application area, although one should also be careful not to confuse layers of successive abstractions from layers of communicating functions. The latter is the case in protocol systems.

In recent years, there has been considerable cross-fertilization between research in software engineering and research in data communications system design, and some of the results obtained have increased the feasibility and desirability of the approach proposed in this paper. These recent developments include:

1. The design of specification languages able to express both service and protocol specifications precisely [LOT] [BLP] [BO2].
2. Proof techniques that make it possible to formally show that the joint operation of a (N)-protocol and a (N-l)-service does indeed realize a (N)-service.
3. "Layered" testing techniques using "prototypes" by which an executable (N)-service specification (a "prototype" of the service) can be used to support an (N+l)-protocol (in lieu of the actual implementation of the first N-protocol layers) [LOG] [LSU]. This specification will not, of course, be able to simulate the missing layers in all respects, but at least will allow some testing to start, fulfilling the function of a "fast prototype" [LSU] of the service provider.

3 Analysis of the Opposition to the Service Concept

From its very beginning, the service concept has been controversial. Its supporters won for it an important place in the Open System Interconnection (OSI) architecture [BRM] [ZIM], but this was not without opposition. During the ISO/TC97/SC16 plenary meeting in Berlin in 1980, even a strong attempt was made by some members to assert the point of view that service definitions should be simple annexes to protocol definitions, instead of self-standing standards. The attempt failed, however opposition continued, and still exists today. ISO and CCITT maintain that there are only "Service Definitions" as opposed to "Protocol Specifications". Furthermore, individual service specifications (see for example [TSD]) contain the proviso:

> "this International Standard does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces within a computer system. There is, therefore, no conformance to this standard".

The OSI Session Service [SSD] was not properly defined until for this reason (among others) some member bodies cast a negative vote on the layer's Draft International Standards.

Even worse, much of the protocol world continues to function as if the concept of service was of interest only within the OSI framework. Existing protocol architectures, such as HDLC, X.25, ARPA-Net, and SNA, are not being updated (or are only slowly being updated) to include the concept of a service, and this causes problems when they are to be related to architectures that instead use the concept. Even some new designs, such as ISDN [ISDN], do not specify services. Scientific papers are being written ignoring the concept, and it is not rare to hear that there is no clear distinction between protocols and services, or that the usefulness of a service specification is simply that it gives an unengaged overview of what a user can expect of a protocol [RRE].

What are the reasons for this opposition? By and large, they can be classified into the five main categories listed below. Points 3.1 and 3.2 can be called designer's concerns, while points 3.3 to 3.6 can be called implementer’s concerns.

3.1 Services may at times be difficult to define without reference to the underlying protocol(s). Typical example is the one mentioned above of the OSI Session Service, for which some people said that it could only be understood in terms of the session protocol.

3.2 One can define a service for which it is hard or impossible to design a supporting protocol. To avoid this potential problem, in the OSI standardization process, service descriptions are expected to progress simultaneously with protocol specifications.

3.3 Some people find it difficult to conceptually separate the service abstract machines, in terms of which services are often described [SCH], from the seemingly less abstract protocol machines, and to keep in mind that only the latter have to be implemented directly. In technical discussion, it is not uncommon to find that objections addressed to a service definition are in fact motivated by the way the implementation is envisaged.

3.4 Implementers, who are justifiably concerned with efficiency problems, often seek optimization opportunities that may be found by crossing or collapsing service boundaries. For example, two related functions found in two different layers could sometimes be implemented together. Clearly, the service boundary can be seen as an unnatural obstacle to this practice.

3.5 Implementers are afraid that users may submit procurement requirements where a device is required to conform to a service specification. This would add a further level of concern for them, who are used to think in terms of protocol implementations only.

3.6 Conformance with a protocol implies conformance with the related service, while of course the converse is not true.

4 Misuses and misinterpretations of the Service Concept

Some of the opposition to the service concept seems to originate in fact from confusions caused by misuses or misinterpretations of the concept that are often found in different contexts. Therefore, we dedicate this section to a discussion of some particularly revealing cases of misuse. The discussion is necessarily short and indicative and far from exhaustive.

4.1 Nature of the service primitive

One particular misunderstanding, leading to misuses of the service concept lies in the interpretation of the nature of the service primitive. Almost everywhere a service primitive is understood as a "kind of message" that is passed across a service boundary. In early ECMA Transport Protocol documents this wording can be found literally. It leads to statements in standards documents like "the direction of the service primitive" [OSC], or "a service primitive is passed across a layer boundary" [P8O2], or "the entity initiating the XXXxxx-service primitive" [OSC] [TSD].

The above interpretation seems harmless if at all times service primitives would indeed define the passing of information in one direction only. The OSI Session service [SSD], however, shows already an example of the contrary by defining bi-directional exchange of information in some of the session service primitives. The correct interpretation of the architectural concept on which the concept of service primitive is built is here clearly at stake.

We would like to stress that the service primitive is built on the architectural concept of interaction, which is enforced by the abstraction level required for boundaries between communicating processes (or entities) inside systems. This concept of interaction allows a much richer variety of parameter value establishment than just "simply" value passing.

As examples of this richer variety we mention value negotiation and value checking [VI3] [LOT]. Both mechanisms are extremely useful and necessary in handling problems, that are largely encountered in connection oriented service and protocol standards, for building up and distinguishing among different connections by means of connection endpoint identifiers. Ignorance of these possibilities gives rise to such highly unsatisfactory statements in standard documents as: "all service primitives must (!) make use of this connection endpoint identification mechanism, however this mechanism is not shown as a parameter of the primitives." [TSD].

But even if the execution of the service primitive would imply passing of information in one direction only, the interpretation of the concept is not always as harmless as one might expect. Poor interpretation leads to poor architectural design: in the recent IEEE P8O2 Local Area Network (LAN) protocol standards [P8O2], for example, a situation is envisaged where the service provider accepts a DATAGRAMrequest from a user and then responds to him with a DATAGRAMconfirmation.

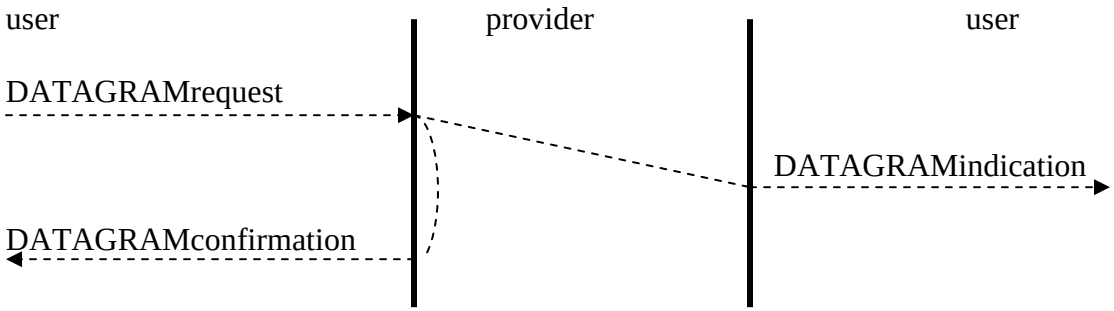


Figure 2: *P802 definition of confirmed datagram*

The meaning of the confirm primitive is not the confirmation of receipt, i.e., the confirmation that the DATAGRAMindication has occurred, but that the local end (!) of the provider to the best of its abilities (!) has carried out the DATAGRAMrequest.

Apparently the introduction of this confirm primitive is inspired by the idea that service primitives may be unreliable (we draw this conclusion also from the many discussions around this issue): information may be passed across the service border but the other side may ignore it, so it may get lost.

This is a misinterpretation of the interaction concept: at specification level one cannot accept the possibility of unreliable primitives: interaction requires participation of all involved parties and when information is passed but not accepted, apparently one of the partners was not involved and the service primitive simply has not taken place. It is an implementation concern, not a specification concern, to see to it that service primitives are executed reliably, and to take proper measures when errors are likely to occur. It is also the implementer, not the specifier, who is in the proper position to anticipate these potential errors (and cure them).

From the user's point of view, clearly the confirmation carries no new information for the user. If the confirmation is positive, there is no evidence for the user what to do with it since it is independent of the DATAindication to the other user, which may or may not occur. Also, what should the user do when the confirmation is negative? To answer this question we enter the field of system management. In our opinion, this useless primitive constitutes a misuse of the concept of service.

We conclude therefore that the above defined confirm primitive is useless to the user and a mixture of specification and implementation concerns.

4.2 The nature of the abstract interface

Another misunderstanding leading to potential misuse seems to lie in the nature of the abstract interface. If we take figure 1 as example and assume that we discuss the network service provider, then the local ordering of network service primitives and their parameter value dependencies is usually called the "abstract interface".

When the user is geographically separated from the network, he has to be connected to the network via a real (i.e. in contrast to abstract) interface including a physical medium. Suppose this real interface is built up of three layers of interface protocol, as shown below:

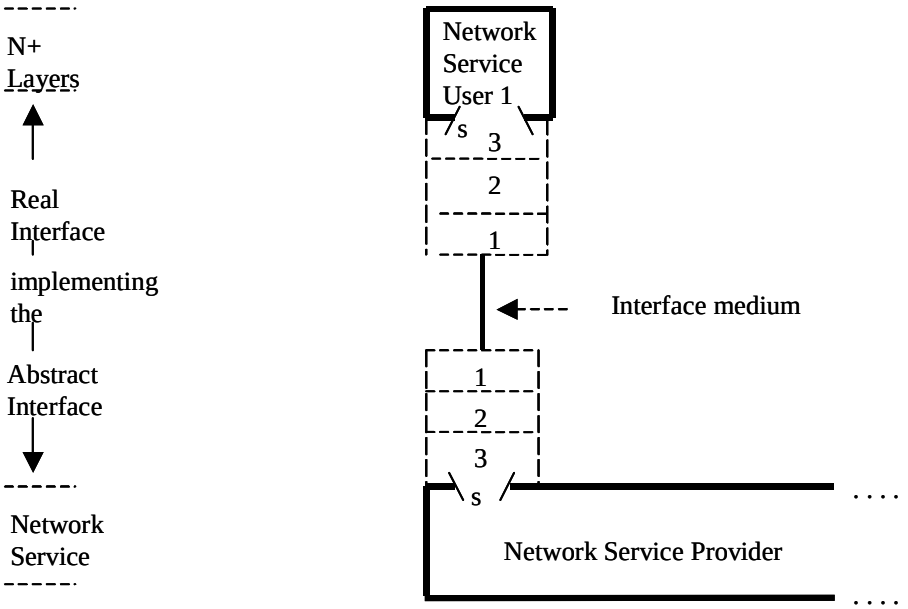


Figure 3: *Network Service User connected to the Network Service Provider by a real interface which is built up of three layers of interface protocol and uses a physical medium.*

The above picture shows that the network provider provides the end-to-end network service, and the real interface has only local significance. For the network user it is important to know how the network service primitives are to be mapped on the real interface primitives, i.e. how he participates in the implementation of the abstract interface, but he does not participate in providing the network service.

The picture can also be depicted in the following way, apparently inspiring a metamorphosis in the interpretation of the service concept:

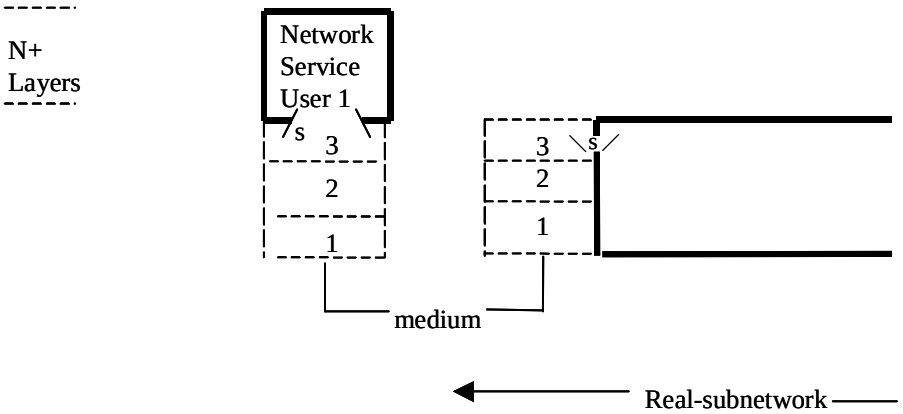


Figure 4: *Configuration of figure 3 drawn in a different way.*

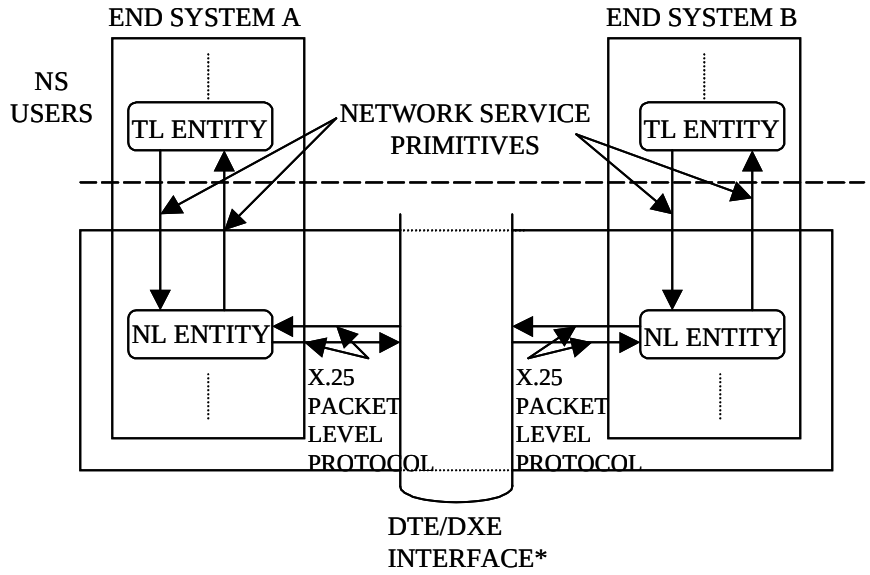
In an earlier version of [P802] dealing with the OSI Representation of Local Network the above interpretation can be found, but applied to the Physical Interface rather than the Network Interface shown in figures 3 and 4. It appears (and certainly the picture tempts to make this interpretation) that layers of interface protocol (in figure 4 indicated by 3, 2, and 1) are erroneously interpreted as layers of

OSI protocol, thus protocol layers necessary for interworking in an OSI environment. Clearly this is not the case as a real interface to the physical layer entity can be chosen arbitrarily.

Although this picture does not appear in the documents defining the LAN access methods, the above confusion still seems to proliferate in member body comments on the LAN standardization effort within SC6.

A more curious development, seemingly related to the above, has occurred around X.25. Originally this standard was called the X.25 "Interface", defining the interconnection of a User (DTE side) to a Public Network (DCE side). Thus one would expect that X.25 defines a real interface at network service level, whereas the PIT would be responsible for providing the network service. Also the structure of X.25, in particular the channel numbering mechanism, suggests this conclusion (although the architectural structure of X.25 has never been clear).

Recently, however, the term interface is disassociated from X.25, and one talks now about X.25 1984. A document has appeared defining how X.25 provides the OSI Connection Oriented Network Service [X25] which contains the following picture:



*** this interface consists of 0 or more network layer entities providing a network layer relay function**

Figure 5: *Operation of the OSI Connection Oriented Network Service and the X.25 packet level protocol (1984).*

The picture confirms, as figure 4 suggests, that by applying X.25 1984 the user is responsible for implementing a part of the network service provider. This interpretation is further enforced by the emerging standard DP 8648 about the internal organisation of the network layer [ION], in particular by examples showing the use of connection mode subnetworks (everyone reads here: e.g. 1984 X.25 "real subnetworks") to provide the connection-mode network service.

The CCITT I. series Recommendations on the Integrated Services Digital Network shows a similar approach as described above for 1984 X.25 (which is not surprising) [ISDN].

One could argue that in practice there is no difference in implementation effort whether one implements three levels of real interface protocol, or three levels of OSI protocol, so why stir up things. We would like to conclude by four points:

- first, it is necessary to achieve architectural clarity and consensus about the service concept, so people understand themselves and each other,
- second, by involving the end user in the provision of the network service, one should realize that the user potentially has to implement several (more than 3!) levels of subnetwork independent convergence protocols and subnetwork dependent convergence protocols, dependent on how many subnetworks are interconnected in tandem. The user equipment would be much simpler if the user was connected to a network service via a real interface,
- third, if the user is connected to a network service via a real interface, this interface needs not necessarily to be standardized,
- fourth, talking about real interfaces definitely is not appreciated within the OSI environment.

5 General Defense of the Service Concept

In this section, we discuss the main reasons in favor of the service concept.

5.1 Design and Abstraction

The concept of service is indispensable for the design of complex protocol systems. Nowadays it is agreed that layering is one of the main tools for managing software complexity [DIJ] [PAR1] [PAR2], and as such it should obviously be applied to data communications systems. For this reason, layer boundaries should as much as possible be chosen in order to enable maximum use of the principles of abstraction and separation of concerns, in such a way as to make it possible to express in simple terms the functions accomplished by an underlying complex system [SCH] [BUS] [BOC] [ZIM]. Used in this way, the layer boundary becomes a tool for preventing complexity from building up as more and more functions are added to a system. If on the contrary a (N)-service can only be presented and understood in terms of the (N)-protocol, it will become very hard, if not impossible, to design the (N+l)-protocol on top of it.

5.2 Growth

The service boundary should be seen as a stable boundary to support later protocol development. The OSI reference model shows a neat building of seven floors. This ideal picture, however, is unlikely to last as the variety of the applications pushes towards adding a variety of branches and roots to the basic model. For example, the Data Link Service could be offered by a Local Area Network, or the Network Service by the ISDN protocol. This could not possibly work without precise definitions of the services involved. This issue is addressed clearly, even if somewhat dubitatively, in [POS].

From this point of view, it is easily seen that the stability of service specifications may be even more important than the stability of protocol specifications. If a protocol is changed within a given layer, the effects are limited to the layer, while a change in a service specification may have consequences that will propagate upwards and affect several protocols.

5.3 User's Concerns

The service is the user's real concern. Most users are only interested in the end-to-end service that is provided to them, while the actual mechanism (the protocol) is often immaterial. As a consequence, the service should be expressed simply in terms of the user's needs. Even if many users will only be interested in the Application Service, others with specialized needs may be interested in purchasing lower layer services (say, the Transport Service), on top of which to build their own systems.

5.4 Correctness Proofs

The service concept is necessary to build correctness proofs of the system's design, and this role cannot be fulfilled unless the specifications are precise to the point of formality, and fairly simple. Much work has been done in recent years on this subject (a collection of recent papers can be found in [IFI]). Techniques are being developed, by which it is expected that one day it will be possible to thoroughly and formally prove that a protocol, used on a service, implements a higher layer service.

Clearly, it is the standardization bodies themselves that will eventually have to be responsible for developing such proofs.

Such a correctness proof will eventually make it possible to ease considerably the complexity of the testing task discussed under item 5.5. If it has been proved that in principle a (N)-protocol on top of a (N-l)-service implements a (N)-service, and it is known that the (N-l)-service is correctly implemented, then it is sufficient to test only the (N)-protocol to know that the (N)-service is correctly implemented.

Once a proof technique is available, one could then use similar principles to prove the conformance of implementations to specifications, and this would save a considerable deal of trial-and-error in getting specification and implementation correct.

5.5 Testing

In our view, access to service boundaries is indispensable for testing implementations. Unfortunately, still today it is common to hear the opinion that testing an OSI implementation should be done in the fashion of an observer that is only able to see what happens on the physical connection between a reference implementation and an implementation under test. This is the point of view usually held by implementers who do not believe in the service concept, or are concerned about testers attempting to probe their proprietary software design. Unfortunately, while such a method may be able to detect errors by trying some reasonable test sequences, it may be unable to pinpoint the origin of the error, nor is it able to check whether or not the user is provided with the expected service. "Layered testing" instead will be able to determine at least in which layer the error is to be found. The concept of layered testing by using prototypes, presented in Section 2, is also relevant in this respect [LSU].

5.5 Implementation

The concept of layered implementation is possibly the one to which protocol software manufacturers are most keenly adverse. Their point of view is that they can only be bound to provide a correct protocol, and how this result will be achieved is no one else's business. Several of the points we have discussed above cast, we believe, a wide shadow on this opinion. In addition, there are software engineering reasons for being sceptical. As mentioned above, the concept of layering was invented in software engineering where it is now well established. Protocol implementations will have to be layered, just because of the sheer complexity of the task. Of course, a clever software engineer could possibly find an organization of the functions among the layers which yields to better results (e.g., higher performance) than the one recommended in the OSI model. But would it be worth the effort? Unusually layered implementations would have to develop new conceptual models, different from those already developed within OSI and it might be difficult to ensure that the resulting protocols and services are completely equivalent to the standard. We feel that the best strategy for initial OSI implementation is to follow the reference model's layering. Once such prototype "reference implementations" are developed, some optimization may then be attempted.

Whatever layered model is used, it is important that precise service and protocol specifications be developed for it. Such specifications would fulfill several different goals:

- make it possible to develop the various layers in different teams, having some assurance that the end results will fit together.
- allow the testing of a layer independently of the implementation of the underlying layers (see the discussion in Section 2).

6 Refutation of the Reasons for Opposition to the Service Concept

The general principles discussed in Section 5 having been established, it is not difficult now to address one by one the reasons for opposition to the service concept that were analyzed in Section 3. Subsections 6.1 to 6.6 below correspond to subsections 3.1 to 3.6 above.

6.1 Services may at times be difficult to define without reference to the underlying protocol(s). We hope to have shown that, even if services may be difficult to define at times, the many advantages offered by the concept of service make this effort worthwhile. Software specifications are also difficult to formulate, however nowadays very few programming shops make do without them, and the trend seems to be towards more formal specifications. Furthermore, users should know what they want, and several of the arguments above (see 5.1 and 5.3) point to the conclusion that a hard-to-define service may be simply a poorly conceived one giving evidence of a poorly conceived protocol!

6.2 One can define a service for which it is hard or impossible to design a supporting protocol. This is a real argument. Of course, service specifications should not be defined in such a way that there exists no realistic protocol to implement them. Users should know what they want, but they should not be over-demanding. On the other hand, if a service can only be understood as the result of a complex protocol on top of a lower layer service, then we would dare to conclude that the layer is poorly designed, because its service specifications do not provide the conceptual advantage of abstraction that should be provided by a well-conceived layer.

6.3 Some people find it difficult to conceptually separate the service abstract machine from the protocol abstract machine. We have watched this difficulty in many protocol implementers, and we find that the only possible answer to this objection is that people need to be educated. Like all new ideas, the idea of service will take time to be absorbed by designers who have for years worked without it, however we have tried to show that this effort is worthwhile.

6.4 Implementation efficiency may be optimized by crossing or collapsing service boundaries. This point was, we believe, implicitly or explicitly present in almost all points of Section 5, especially 5.5.

6.5 Users may submit procurement requirements where a device is required to conform to a service specification. We understand the implementer's concern on this issue, however the discussion in point 5.3 shows, we believe, that users should be entitled to request that a certain service (rather than protocol) be provided. As well, some implementers may offer implementations of certain layers only, and the related service ought to be testable. Also see item 5.5.

6.6 Conformance with a protocol implies conformance with the related service, while of course the converse is not true. While of course this statement is true, the discussion in 5.3 and 5.4 indicates that in some cases conformance with services is what really matters.

7 Conclusions and Recommendations

We have argued the case for an increased role of service specifications in the design and development of communications protocols. To conclude, we would like to present our main theses in point form. We are quite aware of the fact that several of these theses are highly controversial in the protocol community, however we feel that they ought to be presented and that discussion on them must continue.

7.1 Service specifications should be recognized an importance equal to the one usually attached to protocol specifications. The term "service definition" found in OSI documentation should be replaced by "service specification" to correspond to "protocol specification".

7.2 Cases of discrepancy between service and protocol specifications should be considered as specification errors, instead of automatically giving priority to protocol specifications, as is done now (in fact, the discussion in point 5.2 may indicate that the reverse is what should be usually done).

7.3 The specification of services should precede or accompany, but definitely not follow, the specification of protocols.

7.4 A well-designed protocol should have simple and easily understood service specifications or, at the very least, service specifications will always be much simpler than protocol specifications.

7.5 A protocol system should normally be implemented by respecting service boundaries. If necessary, optimizations can be introduced after a working system has been obtained.

7.6 The statement of no conformance to service definitions, usually found in OSI documentation (see above) should be replaced by a statement specifying optional conformance.

7.7 Standardization bodies should be required to provide, together with the (N)-Protocol standard, a proof that (N)-Protocol + (N-1)-Service = (N)-Service. The degree of formality of such a proof would of course depend on the methods available.

7.8 Research should continue on the development of Formal Description Techniques and verification methods suited to deal with both protocol and service specifications and their formal verification.

7.9 Research should also continue on design methodologies based on the concept of (N)-protocol development from (N)-service and (N-1)-service specifications.

7.10 Clarity and consensus about the architectural definition, semantics and role of the service concept is urgently needed in order to promote its advance and avoid its misuse. A defining document in the form of an OSI standard would probably be the most appropriate way to reach this goal.

References

[BLP] Scollo, G., Pappalardo, G., Logrippo, L., Brinksma, E., The OSI Transport Service and its Formal Description in LOTOS. In: Csaba, L., Tarnay, K. and Szentivanyi, T. (eds) Computer Network Usage: Recent Experiences, North-Holland, Amsterdam, 1986. (Proceedings of the IFIP RC 6 Working Conference COMNET '85, Budapest, October 1985) 465-488.

[BO1] Bochmann, G.V., Architecture of Distributed Computer Systems, Springer, Berlin, 1979.

[BO2] Bochmann, G.V., Concepts for Distributed Systems Design, Springer-Verlag, Berlin, 1983.

[BOC] Bochmann, G.V., A General Transition Model for Protocol and Communication Services, IEEE Transactions on Communications, Vol. COM-28, No. 4 (April 1980), 643-650.

[BOS] Bochmann, G.V. and Sunshine, C.A., Formal Methods in Communication Protocol Design, IEEE Trans. on Comm., Vol. COM-28, No. 4 (April 1980) 624-631.

[BRM] International Organization for Standardization (ISO), TC97 Information Processing - Open Systems Interconnection - Basic Reference Model. International Standard ISO 7498 (1984).

[BUS] Burkhardt, H.J., and Schindler, S., Structuring Principles of the Communication Architecture of Open Systems, A Systematic Approach, Computer Networks Vol. 5 (1981), 157-166.

[DIJ] Dijkstra, E.W., The structure of THE Multiprogramming System, Comm. ACM 11, No.5 (May 1968), 342-346.

[GUI] International Organization for Standardization (ISO). TC97 Information Processing - Open Systems Interconnection - Proposed Guidelines for Specification of Services. Protocols and Interfaces - ISO/TC97/SC16/N380 (June 1980).

[IFI] Yemini, Y., Strom, R., and Yemini, S., (Eds.), Protocol Specification, Testing, and Verification, IV, Proceedings of the IFIP WG6.l Fourth International Workshop on Protocol Specification, Testing, and Verification, North-Holland (Amsterdam). 1984.

[ION] International Organization for Standardization (ISO). TC97 Information Processing - Data Communications - Internal Organization of the Network Layer - ISO/DP 8648.2 (Oct. 1984).

[ISDN] CCITT SG XVIII, Integrated Services Digital Network, I.series Recommendations, (October 1984).

[LOG] Logrippo, L., "Constructive" and "Executable" Specifications of Protocol Services by Using Abstract Data Types and Finite-State Transducers, in: Protocol Specification, Testing, and Verification, III. (H. Rudin and C.H. West, eds.), North-Holland, Amsterdam 1983.

[LOT] International Organization for Standardization (ISO), TC97 Information Processing - Open Systems Interconnection - LOTOS Language for the Temporal Ordering Specification of Observational Behavior -150 Draft Proposal ISO/DP 8807 (March 1985).

[LSU] Logrippo, L., Simon, D., and Ural, H., Executable Description of the OSI Transport Service in Prolog in: [IFI] 279-293.

[MYE] Myers, G.J., Software Reliability: Principles and Practices, Wiley, New York, 1976.

[OSC] International Organization for Standardization (ISO), TC97 Information Processing - Open Systems Interconnection - OSI Service Conventions - Draft Proposal ISO/DP 8509 (August 1984)

[P802] IEEE Project 802, Local and Metropolitan Area Network Standard,

- Draft IEEE Standard 802.1, part A (June 1983), (page 29),
- Draft IEEE Standard 802.4 (July 1984) (page A-3),
- Appendix A Functional Requirements Document, Version 5.4 (Draft-Oct. 19, 1981), page 4.

[PAl] Parnas, D.L., On the Criteria to be Used in Decomposing Systems into Modules, Comm. ACM Vol. 15, No. 12 (Dec. 1972) 1053-1058.

[PA2] Parnas, D.L., The Use of Precise Specifications in the Development of Software, in Information Processing 77 (B. Gilchrist, ed.), 861-867.

[POS] International Organization for Standardization (ISO). TC97 Information Processing - Open Systems Interconnection - ISO/TC97/SC21/NI08. Availability of Services in Partial Open Systems (June 1984).

[RAY] Rayner, D., A System for Testing Protocol Implementations, NPL Report DITC 9/82, August 1982. (also in C. Sunshine (ed.), Protocol Specification, Testing, and Verification, North-Holland (Amsterdam), 1982.

[RRE] Unpublished referee report, 1984.

[SCH] Schindler, S., The Distributed Abstract Machine, A Model for Service Specifications in Distributed Systems, Computer Communications Vol. 3, No.5 (Oct. 1980), 208-220.

[SSD] International Organization for Standardization (ISO), TC97 Information Processing - Open Systems Interconnection - Session Service Definition - Draft International Standard ISO/DIS 8326 (1984).

[SUN] Sunshine, C.A., Formal Techniques for Protocol Specification and Verification, Computer Magazine, Vol. 12, (Sept. 1979), 20-27.

[TSD] International Organization for Standardization (ISO), TC97 Information Processing - Open Systems Interconnection - Transport Service Definition - International Standard ISO 8072 (1984).

[VI1] Vissers, C.A., Interface, A Dispersed Architecture, Proceedings of the 3rd Annual Symposium on Computer Architecture, IEEE/ACH, (Jan. 1976), 98-105.

[VI2] Vissers, C.A., Interface, Definition, Design, and Description of the Relation of Digital System Parts, PhD thesis, Twente Univ. of Techn., 1977.

[VI3] Vissers, C.A., Architectural Requirements for the Temporal Ordering Specification of Distributed Systems, in T. Kalin (ed.) Proceedings of the European Teleinformatics Conference (EUTECO), Varese 1983, 79-97.

[X25] International Organization for Standardization (ISO). TC97 Information Processing - Data Communications - ISO/TC97/SC6/N3148 Use of X.25 to provide the OSI Connection Oriented Network Service (June 1984).

[ZIM] Zimmermann, H., OSI Reference Model, The ISO Model of Architecture for Open Systems Interconnection, IEEE Trans. on Comm., Vol. COM-28, No.4 (April 1980) 425-432.