ON PROTOCOL ENGINEERING

H. Zimmermann

Rapporteur: F. Hedayati

ON PROTOCOL ENGINEERING

Hubert ZIMMERMANN
Centre National d'Etudes des Télécommunications, P.T.T.
Paris, France

Distributed processing will play a key role in the development of data-processing and telecommunications. The technical novelty of distributed systems resides in the appearance of protocols to define the dialogue among entities participating in distributed activities. Protocol sciences are still in an early stage of very rapid development. The complexity of distributed systems, as well as the variety of applications and configurations require that their development be based on solid architectural and structured engineering practices. The experience gained so far in data-processing and in data-transmission have led to the definition of an architectural framework and an initial set of structuring tools for distributed systems and their protocols. The corresponding protocol engineering tools are identified in this paper, along with some areas in which further research should be conducted. The paper concludes that the key rules for safe development of distributed systems are simplicity, structuring and standardization.

1 - INTRODUCTION

There is no doubt that distributed processing is going to play a key role, in both data-processing and telecommunications.

The most visible part of this evolution started in the early seventies with the development of computer communications in research projects such as ARPANET [1] and CYCLADES [2]. This association between traditional data-processing systems and telecommunications facilities is now offered on a commercial basis by computer manufacturers in the form of informatic networks [3] as well as by Telecommunications Administrations in the form of telematic services [4].

As important, though less visible to the public, is the progressive introduction of distributed processing inside data-processing systems hemselves and inside telecommunications networks as well.

Inside data-processing systems, this evolution corresponds to the converging development of local area networks [5] and of multiple micro-processors which themselves can be treated as networks of communicating processors.

Inside telecommunications networks, distributed processing is appearing first within circuit and packet switching nodes which are now developed as local networks of processors, and second in "common channel signalling systems" (6) where the telephone switching network tends to be controlled by a separate computer network which transfers and processes all signalling information.

What exactly is this new technical domain of distributed processing ? And indeed, is it really new ? At first glance, the equation of distributed processing is as simple as :

[Distributed processing] =
[Local processing] + [Transmission] [1]

and, since both operands in the right hand part of eq. [1] are not new, one could conclude that distributed processing is just a new name for old methods.

Indeed, the difficulty resides precisely in the "+" operator in eq. [1], i.e. in how to associate local processing and transmission to form distributed processing systems. This operator was felt so important that it was identified as a technical domain of its own under the name of "protocols" [7] and that the equation of distributed processing was rewritten so that it reads :

[Distributed processing]=
protocols [Local processing, Transmission][2]

Protocols turn out to be one of the major keys to distributed processing, which, as outlined above, is an essential ingredient of future data processing systems and telecommunications networks.

Despite their recognized importance, one must admit that "protocol sciences" are still in their infancy, and that much work remains to be done to bring them to the required level of completeness and maturity.

This situation has some similarities with the evolution data-processing where product development has been lagging computer science, and where, nowadays, despite much progress, software engineering still needs serious work.

With the help of this experience, one should try to accelerate and smoothen the development of protocol science and hasten the establishment of solid protocol engineering practices.

This paper should be viewed as a contribution to this collective effort. It attempts to draw from initial experience in protocols and from past experience in data-processing, and give some indications on future directions. Following this introduction, section 2 of this paper insists on the importance of a reference architecture and draws attention to the OSI * Reference Model as a suitable basis for further architectural developments. Section 3 calls for the definition and establishment of simple structuring tools as a basis for protocol design and implementation, while developments in formal description, validation and certification techniques are mentioned in section 4. The conclusion in section 5 recommends that simplicity, structuring and standardization be the basis and the objective of all research and development in protocols and distributed systems.

2 - PROTOCOLS ARCHITECTURE

2.1 - Need for a reference architecture.

Distributed processing systems are potentially complex, for the following reasons :

[a] Contrary to traditional data-processing systems, usage of switched data-transmission networks allows one to envisage a large variety of configurations when assembling data-processing and data-transmission elements to form a distributed system.

[b] Contrary to traditional telecommunications networks, which offer highly specialized services, distributed processing systems will be required to perform any of the large variety of functions in the still growing field of applications of information processing.

[c] Contrary to traditional programming, which is 99 % based on sequentiality, the natural starting point for distributed processing is parallelism which is much more complex than sequential processing.

[d] Contrary to traditional data-processing, errors and failures are to be considered part of everyday's life in distributed processing, and thus error recovery procedures tend to add their own complexity to normal case procedures.

It should be noted that points [a], [c] and [d] above are well under control in telecommunications networks, while point [b] is no problem in data-processing. The inherent complexity of distributed systems comes from the conjunction of these constraints within the same system, where they tend to multiply rather than simply add.

[*] Open Systems Interconnection

How should one tackle the inherent complexity of distributed processing ? Indeed the well known trick for solving complex problems consists of dividing them into less complex sub-problems... In systems engineering, this trick has been heightened to the rank of a technique under the name of systems architecture. The answer to our question above is clear : systems architecture techniques should also be used for designing distributed systems. However, this is not going to be sufficient because the definition of distributed processing is actually recursive and should read :

[Distributed processing] = Protocols
[Distributed processing, Transmission] [3]

Indeed, most distributed systems will be assembled from already existing distributed systems, such as multiple micro-processors, local area networks, telecommunications networks, computer networks, etc...

This recurrent construction of distributed systems will be much facilitated if all distributed systems would refer to the same architecture and if this architecture would allow recurrent construction. In other words, what we need is a common reference architecture for all distributed systems.

How can we get such a common reference architecture ? If we were in a "normal" situation, this architecture would very likely emerge from a natural "trial and error" selection process among various architectures by which the best architecture would survive while the others would die... But we are not in that "normal" situation ! The market is growing so fast that huge investments will be made before any selection has taken place. At that time, selection would be synonymous with catastrophy... It is therefore essential that the absence of natural selection be supplemented by a voluntary process so as to obtain in time [i.e. very early !] the necessary common and stable reference architecture.

Such a voluntary process has been undertaken by the ISO* soon joined by the CCITT**, resulting in the now famous OSI Basic Reference Model (10) which covers the most urgent needs for a common reference architecture for informatic networks and telematic services.

2.2 - The OSI Basic Reference Model

2.2.1 Purpose of this subsection.

The purpose of this subsection is not to give a detailed description of the OSI Reference Model [the brief overview of the OSI Reference Model which follows is provided only as a reminder]. The reader interested in a more detailed description may refer to [8] which is the official ISO text, or to [9] for a summarized description.

[*] International Organization for Standardization
[**] Comité Consultatif International pour le Téléphone et le Télégraphe.

Rather in this subsection, we will attempt to analyse the design choices made in the OSI Reference Model and discuss in which measure they can apply to any distributed system. We will successively discuss : usage of modelling techniques [2.2.3], separation between data-transmission and data-processing [2.2.4], separation between end-to-end and network control functions [2.2.5], structure of the higher Layers [2.2.6.] and structure of the Application Layer [2.2.7].

2.2.2 Brief overview of the OSI Reference Model

The OSI Reference Model deals only with interconnection aspects, i.e. protocols, of distributed systems. It uses therefore a modelling technique in which each "real" open system is modelled by an "abstract" open system made of seven layers of "abstract" subsystems. This decomposition results in a layered network architecture made of seven layers, as illustrated in figure 1.

Each layer plays a specific role in the architecture, as outlined below :

[a] Physical Layer : The Physical Layer has two roles. First it is responsible for interfacing systems to the physical media for OSI. Second, the Physical Layer is responsible for relaying bits, i.e. performing the function of interconnecting data-circuits. Note that the control of this interconnection (namely routing) is performed by the Network Layer and not by the Physical Layer. (See c below).

[b] Data Link Layer :The basic function of the Data Link Layer is to perform framing, and possibly error detection and error recovery between adjacent open systems. The Data Link Layer may also be responsible for coordinating the sharing of multi-endpoint physical-connections [e.g. polling and selecting].

[c] Network Layer : The basic and essential function of the Network Layer is to perform the relaying of packets and the routing of both packets and data-circuits. In addition, the Network Layer may perform multiplexing, error control of and flow control when this is useful to optimize usage of transmission resources.

[d] Transport Layer : The essential functions of the Transport Layer are to perform end-to-end control and end-to-end optimization of transport of data between end-systems. The Transport Layer always operates endto-end. All functions related to the transport of information between systems are performed within the Transport Layer or in the layers below.

[e] Session Layer : The Session Layer performs those functions necessary to support the dialogue between processes, including initialization, synchronization and termination.

[f] Presentation Layer : The function of the Presentation Layer is to take care of problems associated with the representation of informations which applications wish to exchange or to manipulate. In other words, the Presentation Layer covers syntactic aspects of information exchange, permitting application-entities to be concerned only with semantic aspects of informations.

[g] Application Layer : The functions of the Application Layer are all those necessary for distributed applications and which are not available from the presentation service, [i.e. performed by the Presentation Layer or by any of the layers below it].

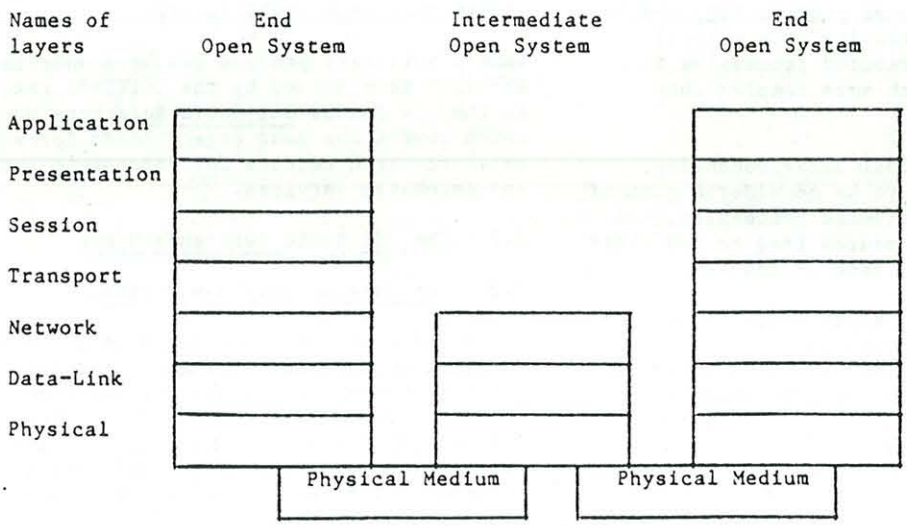

FIGURE 1 : The seven layers of the OSI Reference Model.

2.2.3 Usage of modelling techniques

The modelling technique used in the OSI Reference Model defines the behaviour of an open system as a whole, i.e. without making any assumption about its internal organization. This is very similar to what is done in high level programming languages which provide an abstract definition of a data-processing machine which can be adapted [by a compiler] to a variety or real computers.

This limitation of protocol specifications to interconnection aspects without imposing any specific internal organization is essential to permit adaptation [e.g. by means of gateways] and progressive evaluation towards OSI of systems which have been implemented before OSI standards were available.

This modelling technique is certainly a must for any distributed Systems architecture. It will permit partial evolution of implementation techniques and technologies as well as progressive introduction of new systems without making previous implementations [and the corresponding investments] obsolete.

It should be clear however that any system which is able to align its internal structure with the protocols architecture will benefit through ease of implementation and testing, and ease of adaptation to evolution of protocols. It is therefore likely that the reference architecture, despite it being an abstract model, will also be a reference for structuring real systems, just as high level languages tend to influence the design of real computers.

2.2.4 Separation between data-transmission and data-processing.

One key design decision in the OSI Reference Model is the establishment of a firm transport service boundary which splits distributed systems functions into a data-transmission domain on one hand and a [distributed] data-processing domain, on the other hand, as illustrated in figure 2

The reason for this separation is that data-transmission functions form a homogeneous domain which is "naturally" separated from data processing for the following reasons :

[a] Data-transmission is functionnaly primitive since it performs a highly specialized function while data processing is functionnaly complex since it covers an unlimited variety of functions worked on for years becoming a technical domain of its own ;

[b] Data-transmission and data-processing have been worked on separately for years, each becoming a technical domain of its own.

[c] Error recovery in data-transmission is much simpler than in data-processing, due to the fact that, when transmitting, a copy of the data is equivalent to the original, permitting easy retransmission and duplicate elimination. Conversely, error recovery in data-processing may be very complex and will often be application dependent.

[d] Techniques have been developed to perform data-transmission over network configurations, (including automatic reconfiguration i. e. adaptive routing) by taking advantage of the specificities of data-transmission. Conversely, most current configurations for distributed processing are simply either point-to-point or point-to-multipoint.

These considerations apply to all general purpose distributed system, leading to the conclusion that a firm transport service boundary is an essential feature of any distributed systems architecture.

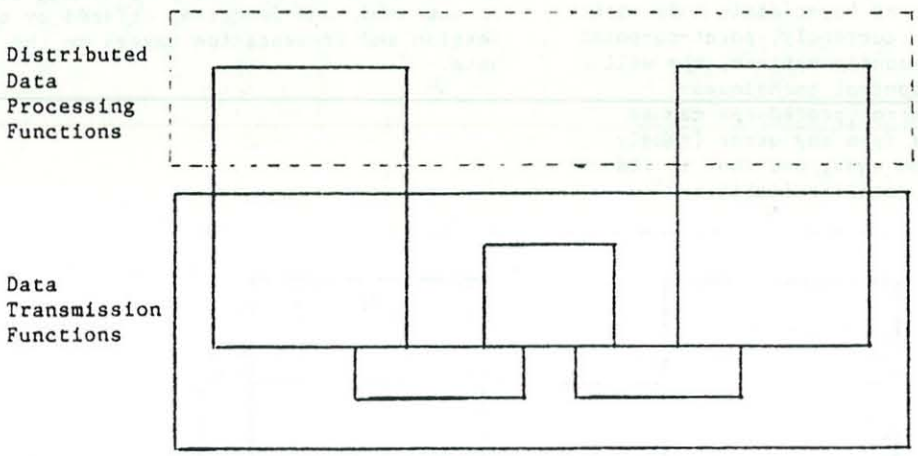

FIGURE 2 : Separation between transmission and processing

Of course, this does not mean that the current definition of the OSI transport service [10] is complete enough to satisfy all requirements of future distributed systems. The OSI transport service will have to be expanded to cover new modes of transmission such as connectionless [11] and broadcast. In this regard, it is interesting to note that, in the absence of a well established [connectionless] message transport service, message transfer must, for the time being, be consideredas a specific application built on top of connection oriented transport services [12]. The normal evolution will be to include it later in the standard transport service.

2.2.5 Separations between "end-to-end" and "network"transmission control functions.

Among the data-transmission functions, the OSI Reference Model clearly distinguishes between :

[a] end-to-end transport control functions on one hand, and
[b] network control functions on the other hand.

Contrary to what has sometimes been suggested, the reason for this distinction is not an administrative one [computer manufacturers would control the ends while PTTs would control networks...]. The reason is a strong technical one, as explained below.

The Network Layer deals with network configurations, routing [i.e. finding routes], adapting routes to changing conditions, recovering from line or node failure, controlling flow over network configurations. These network control functions call on specific techniques which attain a high availability by means of dynamic reconfiguration, but which are not well suited for error recovery.

Each of these two layers plays a specific role, taking advantage of its specific configuration. This would not be possible if the two layers were intermixed.

Conversely, the Transport Layer deals only with simple configurations (currently, point-to-point connections). In such configurations, the well known error and flow control techniques developed for line control procedures can be used to recover easily from any error [namely network errors] between ends, and thus to reach any desired level of overall reliability.

Looking at the current evolution of distributed processing systems, it appears that this separation will be essential. Indeed, the need for reliability network configurations become more complex since more and more networks of similar and of dissimilar types [e.g. packet or circuit switching, local or wide area, private or public, etc...] are being interconnected. The requirement to distinguish between the Network Layer and the Transport Layer will stand and even will increase, and thus, it is a wise choice for any distributed systems architecture to keep this separation.

2.2.6 Structure of the higher layers.

The OSI Reference Model structures the distributed data-processing domain into two parts [see figure 3] :

[a] The lower part [the Session and Presentation Layers] comprises basic functions of general use for the operation of distributed applications, while

[b] the upper part [the Application Layer] comprises all "application specific" [i.e. remaining] functions which make use of basic functions provided by the lower layers.

Despite its apparent novelty, this type of structuring is not completely new. It has been used successfully for many years in the data-processing field where applications are constructed on top of basic functions provided by operating systems and/or by the basic constructs of high level languages. Of course OSI generates new questions since it focuses on the communication and dialogue aspects while traditional data-processing has so far paid little attention to communication. Indeed, the analogy is worth being developed to get a clearer undestanding of the structuring offered by the Reference Model in the higher layers.

Keeping in mind that OSI focusses on communication, figure 4 provides an informal comparison among a few basic functions offered by operating systems and programming languages on one hand, and functions offered by the OSI Session and Presentation Layers on the other hand.

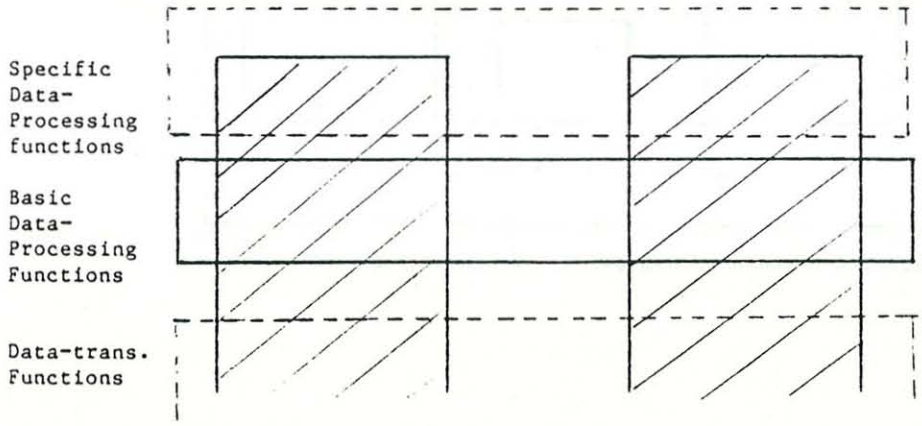

FIGURE 3 : Structuring of higher layers.

The table in figure 4 shows that primitive synchronization and communication functions offered by operating systems find their equivalent in OSI session functions. On the other hand, languages provide complete constructs and processing structure which serve as a reference for both synchronization and data definition. For instance, programs are structured into "procedures" which form the common reference for sequential synchronization by means of procedure calls and for declaration of local data and/or data-types.

The organization of functions in the Session and Presentation Layers have clearly been influenced by high level languages [which nowadays, tend themselves to include the types of functions foun in operating systems].

[a] The Session Layer provides the synchronization structure and synchronization functions required by the application [analogous with procedures and procedure calls]

[b] The presentation functions required by the application refer to the synchronization scheme used by the application [analogous with data declarations within procedures]

[c] Additional sub-synchronization may be required by presentation functions to handle presentation specific functions such as syntax negotiation.

The almost universal usage of high level languages is a clear indication of the validity of the approach taken in OSI for the higher layers of the architecture. It seems therefore reasonable to consider that any distributed system architecture could adhere to the same structure as OSI for the higher layers.

Of course, the current definition of the OSI Session Layer [13] provides only elementary synchronization schemes and will need to be enriched with other schemes [e.g. multi-party dialogue], though avoiding unnecessary complication of synchronization schemes.

2.2.7 Structure of the Application Layer.

In its current definition, the OSI Reference Model says very little about the organization of the Application Layer. It is clear however that the variety of functions and the need for different assembly of functions in the Application Layer will require general structuring tools which still need to be developed. In this regard, one should carefully follow the work started within ISO and CCITT about structuring of the Application Layer [14] where "specific application service elements" call upon "common application service elements", just as user programs call upon a library of common routines. We will come back in section 3 on the general question of structuring tools for protocols.

TRADITIONAL DATA-PROCESSING		O. S. I.	
Operating Systems functions	Prog. Languages functions	Session Layer functions	Pres. Layer functions
Run		Estab. Sess.Cx	
Enqueue/Dequeue		Send/Receive Normal data	
Post / Wait		Send/Receive Expedited data	
Semaphore		Token	
	Co-routines	TWA dialogue	
	Procedure Call	(Part of TWA)	
	Data Types		Syntax
	Declarations		Syntax Negotiat.
	End	Release Sess.Cx	
Check Points		Sync.	
Restart		Resync.	

FIGURE 4 : Analogy between a few functions and in OSI in traditionnal data-processing

3 - PROTOCOL STRUCTURING

3.1 General need for structuring tools

The OSI Reference Model provides a solid framework for organizing distributed systems by identifying major functional blocks [layers] and defining the relations among them. However, it does not specify the detailed functioning and protocols of each layer, and this remaining task is far from being trivial.

Indeed, despite the experience gained so far in networking, the design and implementation of protocols for each individual layer remains difficult and relatively risky. This difficulty results from the inherent complexity of distributed activities, from the evolutionary nature of distributed systems [new-applications, new technologies, new ideas, etc...] and from the lack of adequate tools.

One should realize that protocol tools are almost twenty five years late compared with programming tools, while the market for distributed systems is growing very rapidly and will soon equal today's data-processing market.

In this perspective and in order to make sure that today's investments will not be lost tomorrow, it is essential that protocols be designed and implemented in an open-ended fashion, so that future needs can be accommodated by extending existing distributed systems rather than by replacing them.

Such an open-ended approach to distributed systems and protocols implies a permanent effort for structuring both design and implementation of protocols, in order to retain control on their evolution. This should be facilitated by the provision of standard structuring tools for protocol designers to guide them towards well structured protocols.

A complementary effort will be needed to keep protocols simple and general while every one will be tempted to adapt and optimize protocols to one's particular environment.

It took more than twenty years for the data-processing community to discover [or admit] that structuring, generality and simplicity of constructs provided by high level programming languages were essential for software development. The networking community will hopefully learn much faster by building on past experience.

In the following subsections, we will discuss various aspects of the structuring of distributed activities and protocols, namely structuring in space [3.2] and in time [3.3], data structuring [3.4], error recovery [3.5], parameter setting [3.6] and assembly of distributed activities [3.7]. We will try to identify the basis upon which protocols can already be developed safely, and the areas in which further studies are still required.

3.2 Structuring in space

The structuring of an activity with regard to space, i.e. the configuration of the activity is the first aspect to be looked at, since the complexity of a protocol will be very much influenced by the configuration of the activityit drives, i.e. how many entities constitute it, and which entities communicate with which.

A first rule for protocol designers should be to always make explicit the configuration upon which a protocol operates. [Point to point , point to multipoint, n-party conference, etc]. This particularly important when the configuration varies in time [see section 3.3].

A second rule should be to always stick to the simplest possible configuration, making use, if necessary of additional substructuring.

The decomposition of an activity into several sub-activities [see section 3.7] should allow consider action of complex configurations as an assembly of simple configurations, or to restrict usage of complex configuration to simple functions such as, for instance, data-transmission [see section 2.2.4].

Point to point configurations are the simplest possible configurations and will be the basis for most protocols.

Star configurations can easily be reduced to point to point by considering that the central entity shares its time [multi-processing] among several point-to-point activities.

Cascades and loops are much more difficult to control, primarily in case of errors, and should therefore be used only for simple functions.

Meshed network configurations are even more difficult to control and should be carefully avoided, except for data-transmission, or for research type of experiments.

Of course, further research will permit identification of functions and modes of operation adequate for configurations other than point to point, but, for the time being, and for operational systems, it seems wise to keep within the limits indicated above.

3.3. Time structuring

The structuring of an activity with regard to time, i.e. the synchronization among entities is almost as important as its structuring in space. Indeed, as noted in section 2.1, distributed processing means multiprocessing and thus is naturally parallel. Synchronizing parallel processes has always been tricky and the tools for doing it are still very elementary. Despite all of the work done on protocols, the situation has not progressed much during the last few years. It turns out that many of the difficulties in designing correct protocols come from synchronization problems, i.e. making sure that all possible occurrences of events have been considered and properly handled. There is no reason to expect the situation to change suddenly. So, how should one organize synchronization within distributed activities ?

Here again, one should draw from experience in traditional data-processing where most activities are organized in a sequential fashion, and where usage of parallelism is restricted to the cases where it is easy to handle [e.g. independent activities in multiprogramming systems] or unavoidable. The same orientation should be valid for protocols. One should as much as possible organize distributed activities in a sequential fashion : alternate dialogue for point-to-point activities, token-controlled dialogue for multi-party activities. In the cases where parallelism is necessary, e.g. for performance reasons, one should stick to well known schemes such as producer/consumer communicating through a queue, or mutual exclusion by means of tokens used as semaphores or schemes directly derived from those used in data-link control procedures.

In cases where parallelism cannot be avoided, it will still be wise to structure the activity into sequential phases with well identified boundaries. Examples of such synchronization tools permitting the organization of sequential activities can be found in the OSI session protocol [15], under the name of "major synchronization points".

Further research is clearly necessary to develop more powerful synchronization schemes. In the meantime one coule only recommend to stick to the simple proven schemes mentioned above.

3.4 Data-structuring

Sophisticated and powerful data structuring techniques have been developed in traditional data-processing systems and in particular within high level programming languages and data base systems. Remote access to such data-structures should, of course, be no problem in distributed systems. However, this is not going to be sufficient since one expects more from distributed processing than just remote access. For instance, distribution should permit better protection by localizing each piece of data where it can best be controlled, or improved reliability by duplicating data in different systems.

Despite [and because of] their sophistication, data structures developed for traditional centralized systems turn out to be difficult if not impossible to distribute [because of overall synchronization problems in multi-access] and/or to duplicate [because of consistency and error recovery problems]. This area still needs research in order to get practical solutions, and it is likely that the constraints on distribution will lead to different data organization schemes than those currently used in centralized data-base systems.

3.5 Error recovery

Distribution means better availability since failures only have local impact. However it is necessary to recover from such failures by reconfiguring and continuing the activity on the rest of the distributed system.

Traditional operating system check-point/restart procedures have only a limited applicability in distributed systems since they imply transmission of all data corresponding to the context to be saved at each checkpoint. It is only if the corresponding amount of data is small or if the transmission bandwidth is high and cheap [e.g. on local area networks] that these check-pointing techniques may be usable, otherwise, new schemes are necessary.

The other source of inspiration for error control is datatransmission where much experience has been gained in on-line error control procedures. Basically, these procedures rely on the fact that redundancy of data-transmission is easy to obtain [a copy of the data is strictly equivalent to the original] and that duplicates are easy to eliminate. It turns out that this type of error recovery scheme is not generally applicable in the traditional organization of data-processing because the execution of a program usually results in a change in its environment and therefore a program cannot be executed twice. There are however domains such as process control where this type of error control applies [e.g. ordering the opening of a gate three times is more reliable but equivalent to ordering it just once]. It is not unlikely that this simple scheme could also apply to other domains, provided this constrain is taken into account in the early design of the corresponding activities.

It is clear that much research still needs to be done to design error control schemes adequate for distributed systems. Before such new schemes are available, it would be wise to stick to one of the two traditional schemes inherited from data-processing on one hand and from data-transmission on the other hand.

3.6. Parameter setting

It is often necessary to automatically set or adjust parameters in a distributed activity, and check that they are supported by the parties participating in the activity.

Since this problem does not arise in traditional data-processing or in data-transmission, one is tempted to get inspiration from another source, namely human activities. [It is interesting to note for instance that the term "negotiation" is often used to refer to parameter adjustment]. With this bias, a number of "negotiation schemes" inspired from diplomacy have been proposed, with "offers", "counterproposals", etc..., permitting in theory very refined mutual adaptation and global optimization. The drawback of these schemes is that they are often complex and difficult if not impossible to use in practice ; for instance optimization criteria are obscure as soon as there is more than a single parameter. What happens very often with such "negotiation" protocols is that implementers agree on parameters values to use for each type of application and simply do not use the negotiation capabilities of the protocol.

Keeping in mind that programs are much less adaptable than humans, the "negotiation" scheme could be limited, as in [16] to "Question-Answer" about parameters values supported and "Set" parameters to values decided by one of the parties.

In this area too, research is required, no to design new "negotiation" schemes or protocols, but to get a deeper understanding of what are the meaning and the possibilities of parameters adjustment and optimization in a distributed environment.

3.7 Assembly of distributed activities

In order to keep protocols simple, while performing complex activities, it is necessary to be able to assemble simple activities into more complex ones. Morever, in order to keep control of the resulting assembly, it is essential to keep the interface of each activity to its simplest definition. In other words, the internal functioning of an activity should not be visible outside.

This structuring into modules with functional interfaces is already a well established tradition in data-transmission and in dataprocessing as well. This technique has also been used in the OSI Reference Model where the definition of services provided by a layer is defined independently of the protocols used to provide these services. This decoupling between the services offered and the way they are provided internally should be used systematically in distributed systems.

In addition to this decoupling, there is a need to establish standard assembly schemes so that each designer need not invent his own.The two well-known standard schemes currently in use, namely layering and phases are not sufficient to cover the variety of situations which will soon be found in distributed applications. Further studies are urgently needed in this area.

4 - FORMAL TECHNIQUES

It should not be possible to conclude this paper on protocol engineering without noting that, in the discussion of protocol architecture [section 2] and protocols structuring [section 3] we have not mentioned the ongoing developments of formal description, and validation techniques for protocols [17], nor certification techniques for protocols implementations [18, 19]. These studies are also essential for the development of distributed systems, and the initial results are already very promising. However, these techniques will have very little impact if they do not rest on solid engineering practices concerning the organization of distributed systems. This is the reason why this paper insists on establishing solid protocols engineering practices which, while perhaps less formal, are simply vital for future networks and a necessary step in the development of protocol sciences.

5 - CONCLUSION

The future of data-processing and data-transmission relies in part on the smooth and rapid development of distributed systems. There is very little time to learn, since users already need products which do not even existing in research laboratories.

The only way to progress safely and to minimize the risk is through simplicity, structuring and standardization. SIMPLICITY is the only choice for rapidly putting real systems into operation. STRUCTURING is the only way to manage the inherent complexity of distributed systems, and to maintain control of their necessary evolution. STANDARDIZATION is the only means to ensure the required heterogeneity of distributed systems, by defining precisely the communication rules and interface specifications to be followed by each system, leaving the rest of their design open the innovation.

Hopefully, the necessity of standards for distributed systems has been perceived early, both in ISO and CCITT, and the highest priority has been given to structuring, resulting in the OSI Reference Model. As indicated in this paper, finer grain structuring is still required.

Although simplicity is not much favoured by the "agreement by compromise" process which is traditional in standardization bodies, one must admit that the results obtained so far are quite acceptable. Moreover it is very likely that "natural selection" will bring the additional simplification which may still be necessary in some cases.

As noted along this paper, much research is still necessary and researchers have a large responsibility in the smooth development of future distributed systems.

Moreover, the research and development cycle for distributed systems is so fast that researchers often have a direct responsibility in design choices for commercial products. The research community should not miss to apply the fundamental rules of simplicity, structuring and standardization. In this regard, the active participation of researchers in the standardization effort is worth being noted. To complement this effort to link research with standardization, one should strongly encourage research for simple distributed system.

When simple, structured, standard, distributed systems have been put into operation and are used, engineers and researchers will have more time and more experience to work on optimizing them, but this is the next step.

6 - REFERENCES

(1) L.G. Roberts, B.D. Wessler, Computer network development to achive resource sharing, Proceedings of the SJCC'70, 1970, 543-549

(2) L. Pouzin et al., The CYCLADES Computer Network, North-Holland, iccc Monograph n° 2, Amsterdam, 1982

(3) A. Meijer, P. Peeters, Computer Networks Architectures, Pitman, London, 1982

(4) CCITT, Telegraph and Telematic Services Operations and tariffs, Series F Recommendations, Yellow Book, 1980

(5) D. Clak, K. Pogran, D. Reed, An introduction to local area networks, Proceeding of the IEEE, Vol. 66, n° 11, November 78, 1497-1517

(6) CCITT, Specifications of signalling system n° 7, Rec. Q 701-741, Yellow Book, 1980

(7) L. Pouzin, H. Zimmermann, A tutorial on protocols, Proceedings of the IEEE, vol. 66, n° 11, November 1978, 1346-1370

(8) ISO, Open Systems Interconnection- Basic Reference Model, IS 7498, 1983

(9) H. Zimmermann, Progression of the OSI Reference Model and its applications, Proceedings of the NTC'81, December 1981, F8.1.1 - F8.1.6

(10) ISO/TC97/SC16, Open Systems, Interconnection-Transport Service Definition, DP 8072, 1982

(11) ISO/TC97/SC16, Working Draft for an addendum to ISO 7498 covering connectionless mode transmission, Doc N 1194, June 1982

(12) CCITT Special Rapporteur on Message Handling, Draft Recommendation X.MHS1- Message Handling Systems - System Model - Service Elements, September 1982

(13) ISO/ Basic connection oriented session service definition, DP xxx , March 1983

(14) ISO/TC97/SC16, Application Layer Structure, Doc N 904, January 1982

(15) ISO/ Basic connection oriented session protocol specification, DP xxx , March 1983.

(16) CCITT, Control Procedure for the Teletex Service, Recommendation S.62, Yellow Book, 1980

(17) C.A. Sunshine, Survey of protocol definition and verification techniques, Computer Network, n° 2, 1978, 346-350

(18) C. Sunshine ed., Proceedings of the Second International Workshop on Protocol Specification, Testing and Verification Idyllwild, California, North-Holland Ed., May 1982

(19) J. P. Ansart et al., Description simulation and implementation of communication protocols using PDIL, ACM SIGCOMM '83 on communica- tion Architecture and Protocols, Austin, Texas, March 1983.

DISCUSSION

Dr. Zimmermann pointed out that there was a need for the common reference architecture in order to cope with heterogeneity as a result of the rapid evolution of systems.

Dr. Panzieri stated that if systems evolve, then the common reference architecture would have to evolve likewise. Otherwise how could the reference model continue to be the best in the face of evolution?

Dr. Zimmermann replied that in order for systems to interact with ones that evolve in the future, they must have something in common. The common reference model would thus provide this minimal level of commonality that is required.

Professor Randell remarked that a limitation of the reference model was that it was 'flat' in the sense that the structure could not be applied recursively.

Dr. Zimmermann replied that the structure did not recognise a recursive construction of distributed systems but did permit such a construction.

Dr. Burkhardt pointed out that a recursive functionality was required at the application rather than the system level.