Design of a universal protocol subsystem architecture : specification of functions and services

tu/e
Research Report
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

Design of a Universal Protocol Subsystem Architecture: Specification of Functions and Services

by
M.R.M. Winter

EUT Report 89-E-230
ISBN 90-6144-230-3

December 1989

Eindhoven University of Technology Research Reports

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

Eindhoven The Netherlands

DESIGN OF A UNIVERSAL PROTOCOL SUBSYSTEM ARCHITECTURE:
Specification of functions and services

by

M.R.M. Winter

EUT Report 89-E-230

ISBN 90-6144-230-3

Eindhoven

December 1989

Final report of the post-graduate course "Information and Communication Engineering" of the Institute for Continuing Education (IVO) of the Eindhoven University of Technology, followed in the period April 1987 till April 1989.

Supervisor: Prof.ir. M.P.J. Stevens,
Digital Systems Group,
Faculty of Electrical Engineering,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB EINDHOVEN,
The Netherlands

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Winter, M.R.M.

Design of a universal protocol subsystem architecture: specification of functions and services / by M.R.M. Winter. - Eindhoven: Eindhoven University of Technology, Faculty of Electrical Engineering. - Fig. - (EUT report, ISSN 0167-9708; 89-E-230)
Met lit. opg., reg.
ISBN 90-6144-230-3
SISO 668.7 UDC 621.394.037.37 NUGI 832
Trefw.: datatransmissie; protocollen.

Design of a Universal Protocol Subsystem Architecture.

Specification of Functions and Services.

drs.M.R.M. Winter

Abstract -- This paper describes a framework of a Protocol Subsystem Architecture using a maximum of parallelism. As many protocols show a great similarity concerning connection establishment, maintenance and release, a start is made to model a Universal Protocol Subsystem. In addition to an informal description of the processes within the Universal Protocol Subsystem also a formal description will be given of most of these processes. Parallelism has been used where reasonable to perform a rate of data communication as high as possible. Multiple concurrent connections can be served in parallel. Transmission and reception of data can be performed in parallel. Processes can also operate in parallel if Protocol Data Units are pipelined through the Subsystem in analogue to pipelining within the Communication System. In analogue to multitasking on one CPU, multiple connections can be served by one Communication Entity quasi-parallel. A global overview of the several processes and their mutual communications will be given. Also additional requirements on management of these processes and on memory will be described.

CONTENTS

1. INTRODUCTION 1

2. INTERFACE AND SERVICES 3
2.1. Services of a universal layer 3
2.2. Interface of a universal layer 3
2.3. Sequences of service primitives for one connection 4
2.4. Verification of service primitive sequences 6
2.5. Interface with more than one SAP 7
2.6. Implementation of SAPs 7
2.7. Primitives in software 8
2.8. Primitives in hardware 9
2.8.1. Primitives without negotiation 9
2.8.2. Primitives with negotiation 10
2.8.3. Dynamic attachment of SAPs 10
2.9. Identifiers 10

3. FUNCTIONS WITHIN THE SUBSYSTEM 12
3.1. Set of functions 12
3.2. Relation between service primitives and functions 15

4. GLOBAL ARCHITECTURE OF THE SUBSYSTEM 17
4.1. SAPs and attachment control 18
4.2. Layer Management 18
4.2.1. Fault Management 18
4.2.2. Configuration Management 18
4.2.3. Performance Management 19
4.2.4. Security Management 19
4.2.5. Accounting 19
4.3. Datagram entity 19
4.4. Routing 19
4.5. Multiplexing/splitting entity 20
4.6. Communication entity 21

5. DATAFLOW AND TRANSFORMATIONS 22
5.1. Layer Management 22
5.1.1. Performance / Configuration Management 24
5.2. Datagram Process 25
5.2.1. Transmit process 25
5.2.2. Receive process 25
5.3. Layer Operation 25
5.3.1. Connect process 25
5.3.2. Disconnect process 29
5.3.3. Restart process 31

5.3.4. Connection Control process 32
5.3.5. Accounting 33
5.4. Data Transfer process 34
5.4.1. Sequencing 34
5.4.2. Flow Control 34
5.4.3. Data Transfer Control 36
5.4.4. Blocking / segmenting 36
5.4.5. Deblocking / re-assembling 36
5.4.6. Coding / decoding 37

6. FORMAL SPECIFICATION OF THE EXTERNAL BEHAVIOUR OF THE PROCESSES 38
6.1. Layer Management 38
6.2. Routing and Datagram process 38
6.3. Layer Operation process 40
6.3.1. Connection Control process 41
6.3.2. Connect process 43
6.3.3. Disconnect process 45
6.3.4. Restart process 49
6.4. Data Transfer processes 50
6.4.1. Sequencing process 50
6.4.2. Flow Control process 54
6.4.3. Data Transfer Control process 55
6.4.4. Blocking/ segmenting 55
6.4.5. Deblocking / re-assembling 58
6.4.6. Coding process 58
6.4.7. Decoding process 60

7. MULTIPLEXING / SPLITTING 61
7.1. Multiplexing 61
7.1.1. Control "down" 62
7.1.2. Control "up" 63
7.2. Splitting 65
7.3. Extension to Data Transfer processes in order to provide splitting 66
7.3.1. Extensions to Sequencing process 66
7.3.2. Extensions to Deblocking process 67

8. MANAGEMENT 69
8.1. Layer Management 69
8.1.1. Resource Management 70
8.1.2. Memory Management 70
8.1.3. I/O Management 72
8.2. Layer Operation 72
8.2.1. Connection Management 73

8.2.2. I/O Management and Memory Management 74
8.2.3. Event Management and synchronisation 74
8.2.4. Protection 74

9. CONCLUSIONS 75

REFERENCES 76

GLOSSARY 78

APPENDIX A A CCS SPECIFICATION OF PRIMITIVE SEQUENCES. 80
SERVICE USERS 80
SERVICE PROVIDER 81

APPENDIX B A CCS SPECIFICATION OF LAYER OPERATION .. 91
Connection Control process 91
Connect process 91
Disconnect process 92
Restart process 93

1 INTRODUCTION

In communications levels can always be distinguished. At the highest level, pure data is transferred, Every lower layer adds information to this data in order to be sure of correct transfer. For this purpose, each level performs special, closely related functions. In the OSI Reference Model [1], therefore a communication system is subdivided into seven subsystems. Each subsystem contains a number of entities (see figure 1.1). Entities can communicate with peer entities within another system via a connection. We will consider an entity as a process or program which can operate multiple connection quasi-parallel, but only one connection at a time.

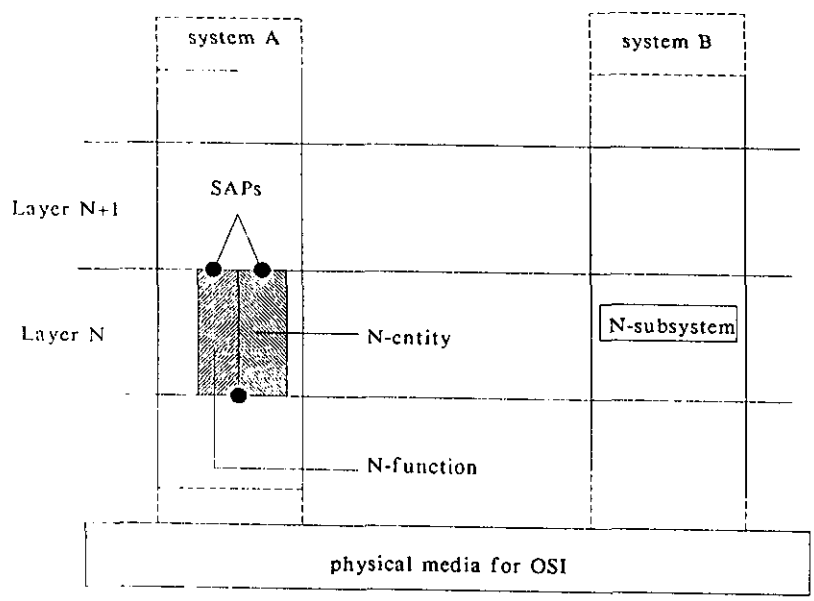

Fig. 1.1 Communication systems

For two entities to successfully communicate they must in advance agree upon a mutually acceptable set of conventions. This set of conventions is called a "protocol". The key elements of a protocol are syntax (data format, coding), semantics (control information, error handling), and timing (speed, sequencing) A protocol describes the horizontal communication. Because entities in a layer have to make use of the services of the lower layers for this communication, also vertical communication has to be defined. This is done by definition of service primitives.

The OSI Reference Model does not specify services or protocols for open system interconnection. It only establishes a framework for coordinating the development of standards. It is neither an implementation specification for systems, nor a basis for appraising the conformance of actual implementations.

The services and protocols described for various layer 2, 3, 4, 5, and 6 protocols (e.g. HDLC, SDLC, X.25, SNA, EHKP4, etc.) show many similarities, especially with

regard to connection establishment, maintenance and release. Therefore, each subsystem can be regarded as a special implementation of a universal protocol subsystem (UPS). This UPS performs the maximum of functionality and provides a maximum of services. A subsystem which needs less functionality can have the same architecture as the UPS with the elements deleted which are not needed.

In this report we will describe the requirements model of a Universal Protocol Subsystem. In chapter two we will give a description of the services which have to be provided by a universal subsystem, the interface between two subsystems and the sequences of service primitives across this interface. Chapter three gives a description of the functions needed to perform these services. The global architecture of a universal protocol subsystem will be described in chapter four and a more specified description of the processes is given in chapter five. In chapter six, a more formal description of some of the processes will be given. If we use multiplexing or splitting, some functions have to be extended. This functions and their extensions will be discussed in chapter seven. In order to control the subsystem, Layer Management and Layer Operation processes will be described. These processes show much similarities to multi-processor operating systems. A comparison will be made in chapter eight.

In this paper the processes within the subsystem are still described on a high level of abstraction. The implementation of the proposed architecture is for further study. Also, the combined behaviour of some processes has not yet been verified. In order to perform this verification, these processes have to be modeled in a formal language.

2 INTERFACE AND SERVICES

2.1 Services of a universal layer

Peer N+1 entities which are associated with each other through an N connection, can only communicate with each other by using the service of the N layer [7]. Each particular protocol provides a set of services [5]. The union of these sets results in a set of services which have to be provided by a universal layer. This set is listed below. The services provided by each layer individually are a subset of this union. Most layers provide a set of services with only a few elements less.

Table 1 Services of a universal layer

IDENTIFICATION
a) address identification b) connection identification c) connection endpoint identification
ESTABLISHMENT
a) connection establishment b) quality of service establishment
DATA TRANSFER
a) SDU transfer b) sequencing c) flow control d) error notification e) reset/restart f) expedited data transfer
RELEASE
a) release connection

2.2 Interface of a universal layer

The interface between an N+1 entity and the N layer is provided by N Service Access Points (N SAPs). An N SAP is considered to identify the N+1 entity it is attached to. Not all N SAPs are permanently attached to an N+1 entity but attachment can be asked dynamically from the N layer. An N SAP may be attached to more than one N entity, and each N+1 entity may be attached to more than one N SAP. Each SAP may contain more than one connection endpoint (CEP) [1], [2]. (see fig 2.1)

According to the definition of ISO, an N SAP is a point at which N services are provided by an N entity to an N+1 entity. Because more than one N entity, and only one N+1 entity may be attached to an N SAP, a better definition would be a point at which

an N+1 entity can ask the N layer for services.

As we considered an entity to be a process that could operate only one connection at a time, there is no need to provide an entity with more than one SAP to the lower layer. Although, if attachment to more than one lower layer entities is required, a splitting entity can be used. Attachment of SAPs to lower layer entities is controlled by Layer Management in this lower layer.

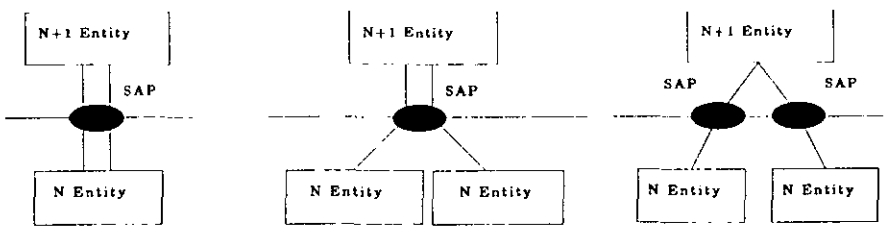

Fig. 2.1 Attachment of entities to SAPs

In order to communicate through an N SAP with an N entity, an N+1 entity uses so called "service primitives". A service primitive can be considered as an elementary interaction between a service user and a service provider during which certain values for the parameters are being established to which both service user and service provider can refer. Thus a service primitive is not just an atomic action across the boundary. Therefore, there is no need for the N layer to send an acknowledgement of a service primitive to the N+1 entity. However, if we define service primitives at a high level of abstraction, we can consider them as atomic actions.

The names which are defined for the service primitives are composed of a generic name, e.g. CONNECT, DISCONNECT, and of a specific name; request, indication, response, confirm. In this paper these names will be abbreviated. (e.g. CONreq, CONind, DISCreq)

If SAPs and service primitives are implemented correctly and independent of the protocol that is used to provide the specified services, this means that different protocol implementations can be used. One can select whatever protocol implementation one wants. Only the external behaviour is fixed.

The implementation of SAPs and service primitives will be discussed in section 2.6, following the methods described in other reports [13], [14].

2.3 Sequences of service primitives for one connection

In figure 2.2 (see below) a diagram of service primitive sequences is given which is extracted from the union of several protocols [1], [3], [5]. The diagram contains the sequences which are allowed at an N interface between two universal layers. These sequences specify the external behaviour of the entities within the subsystem at one connection end point (CEP). The "DATA" primitives in the diagram both represent requests and indications. Also the "DATAGRAM" primitives represent requests and indications. These primitives are used for connection-less data transfer.

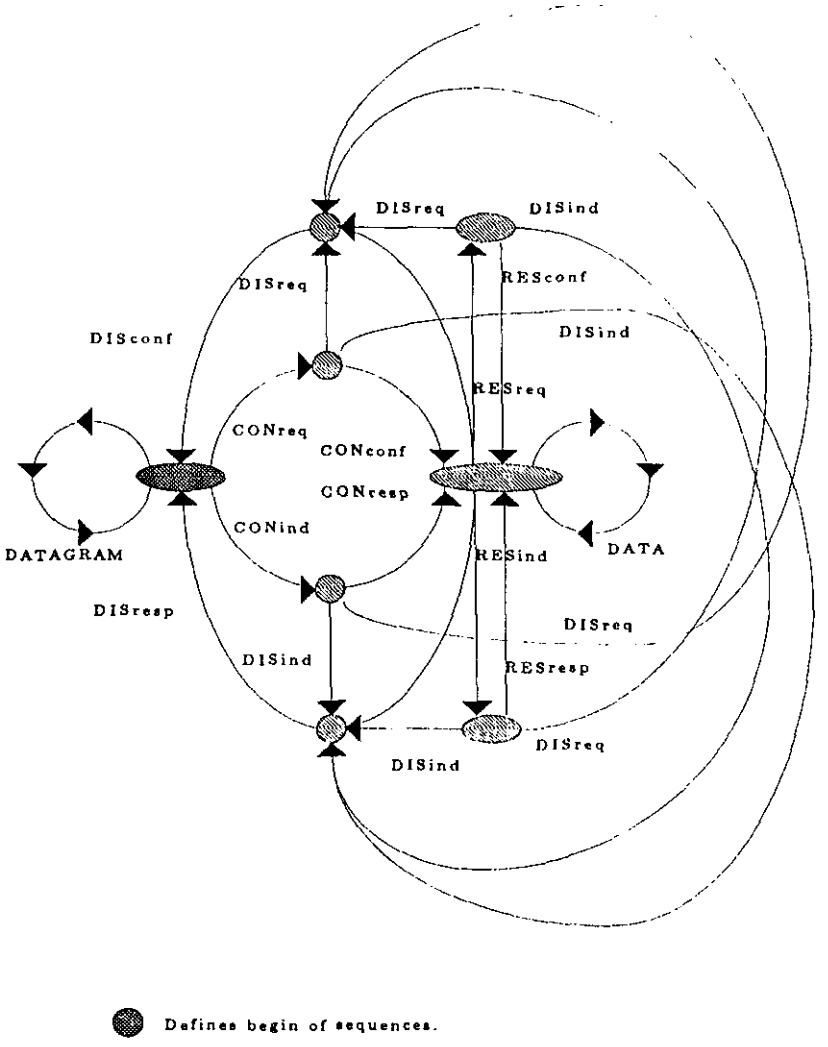

Fig. 2.2 Sequences of primitives at one CEP

In the diagram, a disconnect-request is followed by an acknowledgement. This is only needed in some protocols in order to be sure that the lower layer has released the connection. In protocols where a disconnect command does not require an acknowledgement we can see the acceptance of the disconnect primitive by the layer as an indication (e.g. X.213). There are also protocols in which no connect-response is needed (e.g. LLC). In that case the protocol assumes that a connection can always be established, so in such cases acceptance of the connect-indication can be seen as a response.

2.4 Verification of service primitive sequences

The sequences of primitives at one connection endpoint, as described in the previous section, have been be modelled in CCS [8], [9], [17]. In the CCITT X.213 recommendations [3], a service provider is described which can also be modelled in CCS. By calculating the combined behaviour of a CEP and this service provider, we can prove the absence of deadlock in the combined process and we can see that two CEPs can communicate well using these sequences (see Appendix A). Absence of livelock cannot be guaranteed. In that case the N layer has to resolve from this error.

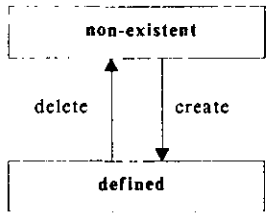

Fig. 2.3 State diagram of a SAP or CEP

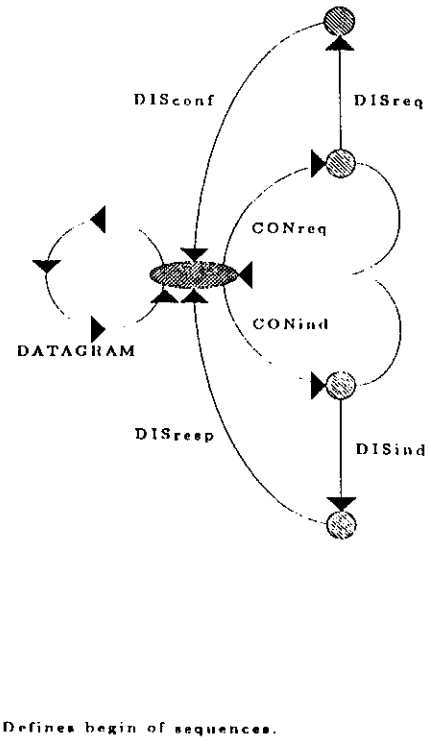

Fig. 2.4 Sequences of primitives at a management SAP

2.5 Interface with more than one SAP

If an N layer can provide services at more than one SAP, each with one or more CEPs, the interface may contain more than one SAP operating in parallel. Every SAP and every CEP can be in one of the two states "non-existent" or "defined" (see figure 2.3). Creation of a new SAP and deletion of it are performed by the N layer.

In figure 2.4, the sequences of service primitives that occur at the management SAP are given. Also, the datagram primitives are defined here although these primitives are sent and received by a special datagram process. Management and datagram have not been separated here. After a connect-request or -indication, Layer Management can start an entity and attaches it to a "new" SAP. This is shown by the nameless arrow in the diagram. If an error occurs during establishment, no connection will be established, and Layer Management will invoke a disconnect-primitive. After creation of an entity and a SAP, Layer Management specifies the beginning of the service primitive sequence through this sap. Whether this sequence will start with a connect-confirm or a connect response. This is shown in figure 2.5 below by two initial "states".
Although we use the word "state", the diagrams do not define the states of processes as with state diagrams. The diagrams only describe the sequences of external actions of a process. Between two actions, a lot of process states can pass. Together with these service primitives, the service is provided by a set of parameters per primitive. (e.g. Identification is provided by an address parameter, QOS by a QOS parameter and error notification by a reason parameter in the
primitives). Other services are implicitly provided by the primitives themselves. For an example of parameters, see Table 5/X.213. CCITT recommendations X.213.

Initiation procedures or peer-to-peer coordination services can be realized by use of data primitives of the lower layer (e.g. set mode frames in HDLC). These data frames sometimes need an acknowledgement by another frame. E.g. a disconnect-request-frame needs a disconnect-response-frame as acknowledgement. The exchange of management information utilizes a so called management protocol which gives the following services : event notification, information acquisition and control activities The protocol mostly used for this data exchange is a stop and wait protocol.

2.6 Implementation of SAPs

According to the CCITT, service primitives represent the logical exchange of data in an abstract way. They neither specify nor constrain the implementation. A service primitive is used to give commands or responses, and for transfer of parameter values between entities in adjacent subsystems. According to the Reference Model, besides value passing value negotiation has to be possible too. Invocation and acceptance of service primitives in that case goes as follows :

1) primitive invocation and value passing
2) value negotiation
3) primitive and value acceptance

The negotiation phase is only needed in higher layers for Quality Of Service negotiation. In lower layers value passing is sufficient. In the next sections, we will give some possible implementations of service primitives in software, and more detailed in hardware.

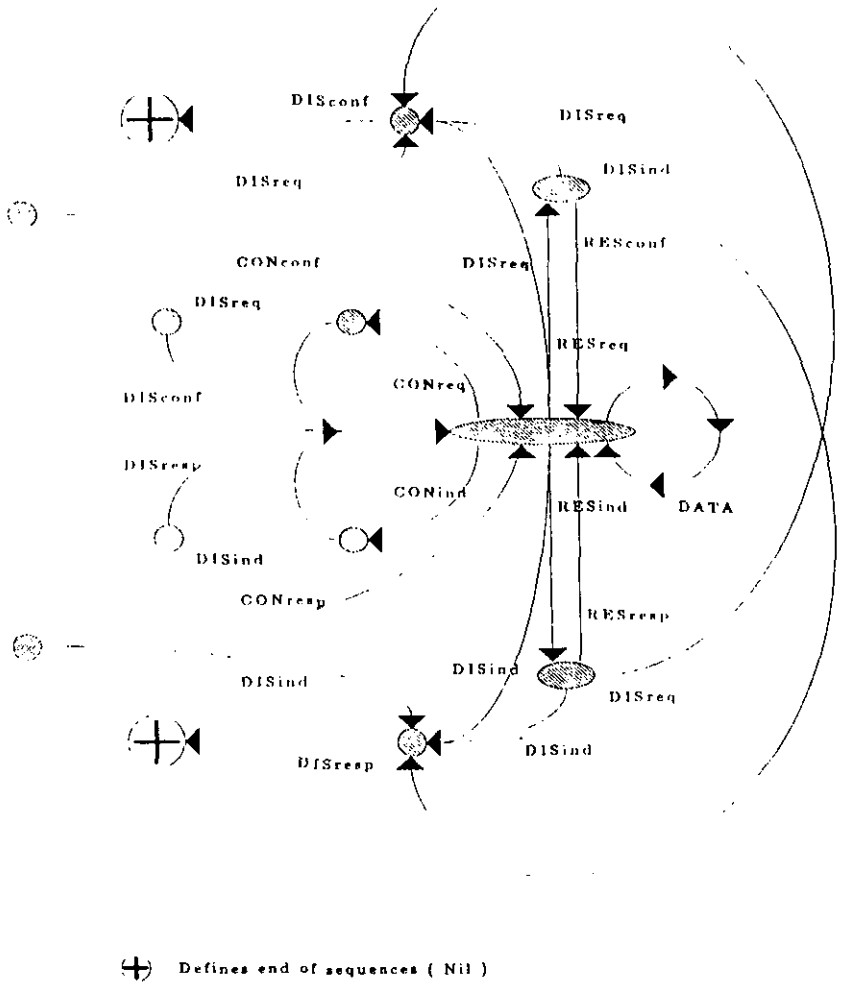

Fig. 2.5 Sequences of primitives at a SAP

2.7 Primitives in software

The implementation of primitives in software depends on the complexity of the layers. In layers with only single entities with single tasks, primitives can be implemented in functions or routines. In layers with more multi-tasking entities, we have to use tools as mailboxes or semaphores. For example primitives for one entity can be stored in a mailbox with a counter indicating the number of waiting primitives. The entity can read from this mailbox and has to send an acknowledgement to the primitive sender when it accepts the primitive.

2.8 Primitives in hardware

2.8.1 Primitives without negotiation

A hardware primitive consists of a set of registers for parameter passing, and a control unit that controls the handshaking between primitive sender and primitive receiver.

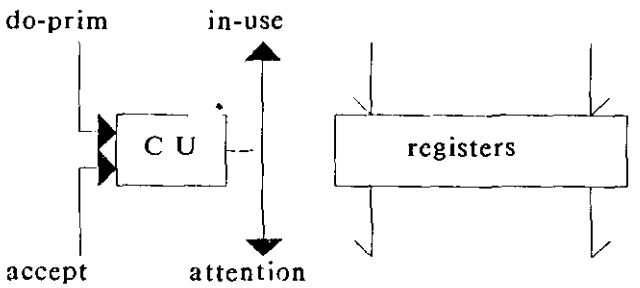

Fig. 2.6 Implementation of a primitive controller

If a primitive is passed, the following actions have to be possible :

1) ask attention of service primitive receiver
2) parameter passing
3) confirmation of primitive acceptance

It should also be possible to cancel the service primitive until the arrival of a confirmation. A primitive is actually transferred to an adjacent subsystem only after it has been confirmed. Before that, "primitive negotiation" is still passing.

With the above hardware, the primitive sender may only start writing primitive registers if the in-use signal is inactive. After insertion of the parameters, it will activate the "do-prim" signal. This results in the activation of the attention signal. The primitive receiver reads the parameter registers and can activate the acceptance line immediately after reading. The in-use signal will be deactivated now, and the primitive sender knows that the primitive has been accepted. It is however also possible for the primitive receiver to delay activation of the acceptance line until it has executed some functions related to the primitive. The acceptance signal can be used as a kind of confirmation in that case. This suggests that the execution will always succeed. If an error occurs, the receiver has to report this to the sender by using another primitive.

A primitive sender can cancel the primitive until the moment the in-use signal is inactivated. The in-use / attention signal is then inactivated, and the primitive receiver notices the cancellation, i.e. if it had already noticed the primitive ,or will never notice the primitive. If the receiver therefore delays the confirmation, it has to notice all changes on the attention line.

2.8.2 Primitives with negotiation

If we want to implement primitives with parameter negotiation, more hardware is needed. We also need registers for the parameter values the primitive receiver wants to accept, and we need extra hardware to show that the values in a register have been changed. Another possibility would be to use shared memory for the parameter values. The control unit does not differ in that case from the unit discussed in this section.

For each parameter we add a "new-bit" to the registers, which will initially be inactive. The primitive sender now acts as before if it wants to invoke a primitive. If the receiver accepts the values, it just activates the acceptance line as before, if it does not accept the values, it stores the acceptable values in its registers and activates the new signal. The primitive sender now reads these new values and can also store new values in its registers. After that, it activates the new signal. The primitive receiver deactivates the new signal and can read the new values of the sender as if it was an original request. This negotiation can go on but will in practice finish here. If the primitive sender does not accept the new values of the primitive receiver, it can cancel the primitive by inactivating the do signal. If the primitive receiver wants to end the negotiation, it has to send a different primitive.

2.8.3 Dynamic attachment of SAPs

In the above case, we have only handled a SAP which was permanently attachment to the entities. In case of connection splitting more entities can read from one SAP or write to one SAP. The control unit of such a SAP will be more complex. It has to perform an arbiter function in case multiple entities want access at the same time. Only the entity of which the request signal is granted has to be acknowledged. For this purpose, every entity is attached to the control unit via a "do line", an "acceptance line" and a shared "in-use / attention line" (see figure 2.7).

If the control unit notices a "do signal" from an entity, it activates the corresponding "acknowledge line" to indicate the request is granted. Then it activates all the attention signals and after it has received an acceptance signal from an entity, it acknowledges it and inactivates the "in-use / attention line". When two or more do-signals arrive at the same time the control unit has to decide which will be granted first. The same will happen when two or more acceptance signals arrive.

In this situation an acceptance signal cannot be delayed by an entity as was the case in the previous situation because the other entities have to know that one entity is already executing the primitive.

2.9 Identifiers

As an interface can have more SAPs, each SAP has to be uniquely identified by a SAP Identifier (SAPI). This SAPI can be a permanent number of the SAP. Also each SAP has to contain an identifier for each connection ending within this SAP, a

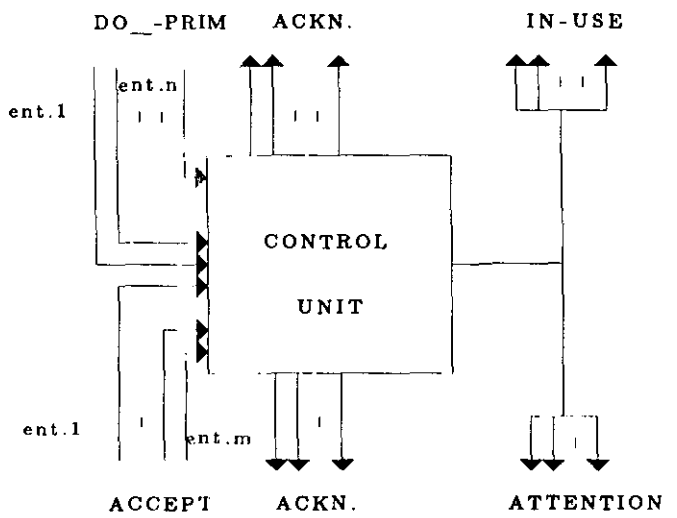

Fig. 2.7 Control unit of a SAP with dynamic attachment

Connection EndPoint Identifier (CEPI). These numbers are not permanent but depend on the connection which is used for communication. The upper entity sets this CEPI, the underlying entity can not change the CEPI. The naming of the CEPs is controlled the Layer Management and the Layer Operation of the layer below a SAP.

3 FUNCTIONS WITHIN THE SUBSYSTEM

In order to provided the specified services, each layer in the OSI reference model contains a set of functions. Not all functions are necessarily invoked in each layer for every communication. It is still being discussed what will be considered the minimum functionality needed of any individual layer. A universal layer has to provide a maximum functionality. Therefore, a universal layer has to contain the union of these sets of functions. This set of functions, extracted from various recommendations and descriptions, is listed below.

3.1 Set of functions

The set of functions is subdivided into four classes. One class contains the very important management functions, the other classes perform a typically connection oriented subdivision of the functions. Functions needed for connection-less transmission form a subset of the four classes.

Table 2 Set of functions for a UPS

LAYER MANAGEMENT

* select functions that will be operational
  (configuration-, security-management)
  + establish optimal PDU size
  + connection multiplexing or splitting decision making

* activation, modification and deletion of entities and relations.
  (configuration-, performance-, accounting-management)
  + scheduling
  + naming

* error control
  (fault management)
  + detection
  + correction
  + testing

ESTABLISHMENT PHASE

* establish connection on lower layer connection
* addressing (routing)
* data transfer

DATA TRANSFER PHASE

* data transfer
* expedited data transfer

Table 2 Set of functions for a UPS (continued)

DATA TRANSFER PHASE
* multiplexing
* splitting
* sequencing
* flow control
* error detection
* error recovery
* segmenting
* blocking (concatenating)
* SDU delimiting
* synchronization

* reset/restart

RELEASE PHASE
* release connection
* notification of reason of release
* data transfer

The Layer Management functions perform overall control of the subsystem (see chapter 8). These functions will be executed by special control processes. The other functions will be executed by communication entities (see chapter 4) which are only active during communication. For this communication, the information between peer entities that are related will be transferred in so called protocol data units (PDUs).

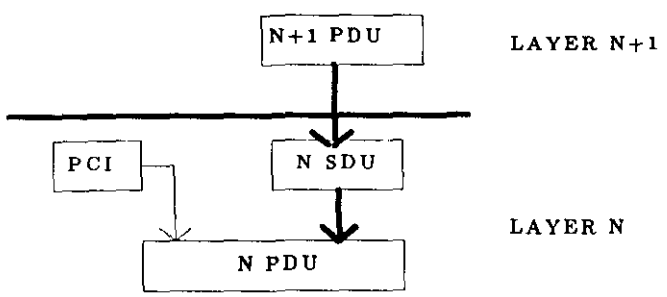

Fig. 3.1 Transfer of PDUs without special mapping

These PDUs may contain only data, or data plus protocol control information (PCI). In order to transfer PDUs to a peer entity, a service data unit (SDU) will be transferred to the lower layer subsystem (see figure 3.1). This subsystem handles the SDU as data, adds PCI to it, and generates a new PDU. A layer can also map one SDU into more PDUs

(segmenting, fig 3.2), or more than one SDU into one PDU (blocking, fig 3.3). It is also possible for an entity to map more PDU into one SDU of the layer below (concatenating, fig 3.4).

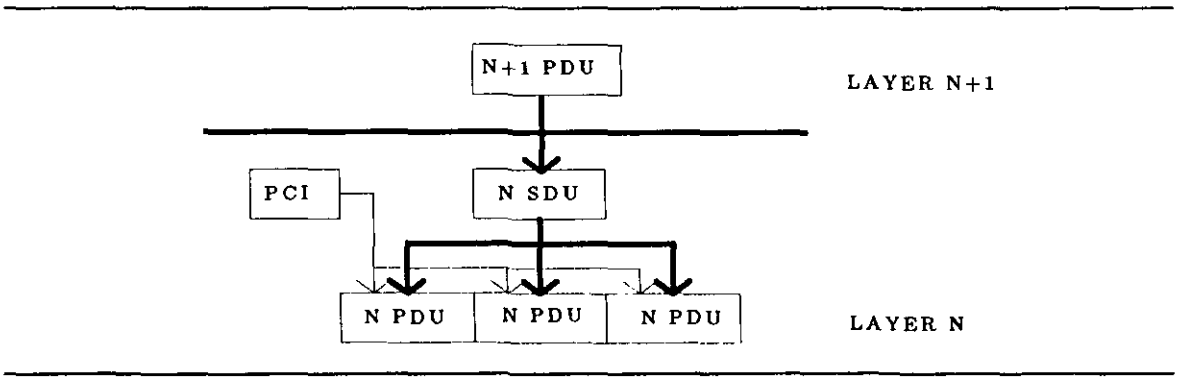

Fig. 3.2 Segmenting

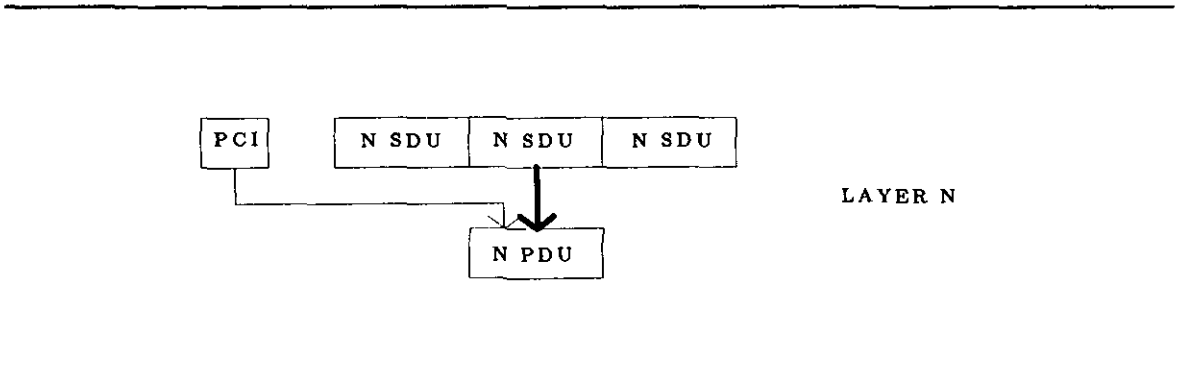

Fig. 3.3 Blocking

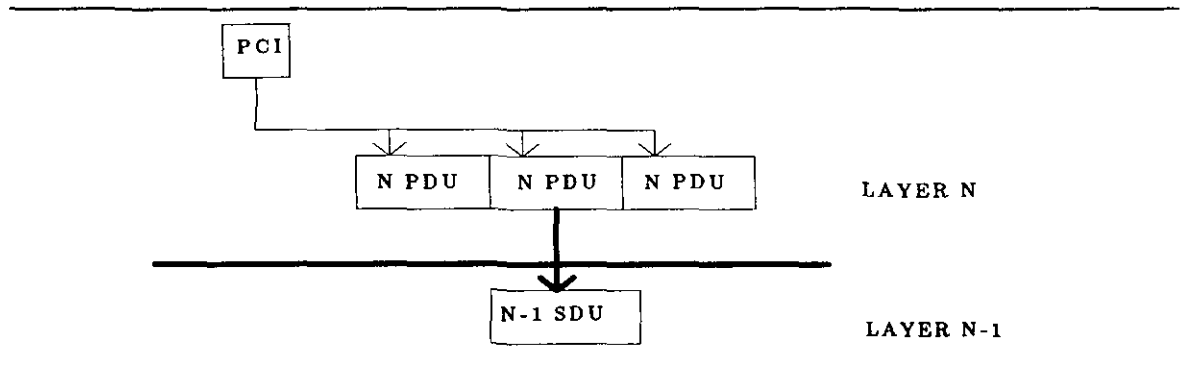

Fig. 3.4 Concatenating

3.2 Relation between service primitives and functions

A function or a group of functions within a subsystem forms an entity. Activation, modification and deletion of entities is controlled by Layer Management. As a result of service primitives or certain events an entity can be in one of the states given in figure 3.5. The entity that provides the management services always has to be existent, otherwise the subsystem cannot be activated by subsystems below or on top of it.

If a subsystem wants to communicate with another subsystem the management entity creates entities for this communication, initializes them according to the service parameters in the connection primitives, activates these entities. Data has to be transferred to the remote layer management to ask connection and initialisation. This data can be sent by using datagram primitives; the lower layer makes no connection, or by using the information field in the connection primitives; the lower layer now establishes a connection too. The invocation of these service primitives can be done concurrently with the management of the new entities when all required parameters are available. After connection confirmation, the new entity is in the "communicating" state.

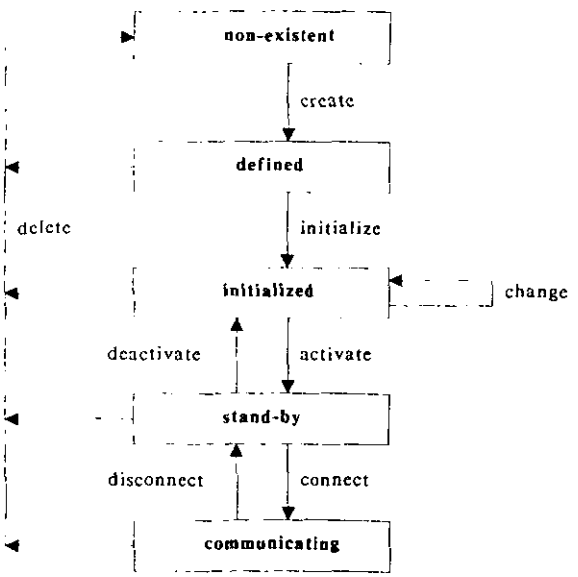

Fig. 3.5 State diagram of an entity

The occurrence of a service primitive may result in activation of certain functions to provide the desired service. For each service primitive of the previous chapter we will list the set of related functions that may be activated :

CONreq
- select functions that will be operational
- activation and modification of entities
- error control
- establish connection on lower layer

	connection - addressing - data transfer
CONind	- idem, except connection establishment
CONresp/conf	- modification of entities - error control - addressing - data transfer
DATAreq/ind	- data transfer - multiplexing - splitting - sequencing - flow control - error detection - error correction - segmenting - blocking - SDU delimiting - synchronization - reset/restart
EXP.DATAreq/ DATAind	- idem, plus expedited data transfer
RESTreq/ind	- reset/restart - sequencing - flow control
DISCreq/conf	- data transfer - release connection - notification of reason of release
DISCind/resp	- data transfer - release connection - notification of reason of release - functions of connection request when new connection wanted

In chapter five we will give a more detailed description of the relations between service primitives and functions.

4 GLOBAL ARCHITECTURE OF THE SUBSYSTEM

Now that we have described the external behaviour of the system and we have specified the functions which have to be executed to establish this behaviour, we will discuss the internal structure of the subsystem. In chapter five, we will give a functional description of the processes within the UPS. This description could be given independent of an architectural model. However, functional decomposition and architectural decomposition are in most cases closely related. Sometimes, architectural requirements, e.g. use of shared memory, put extra constraints on the processes within the architecture. In this chapter we will therefore give a global architectural model of the UPS (see figure 4.1).

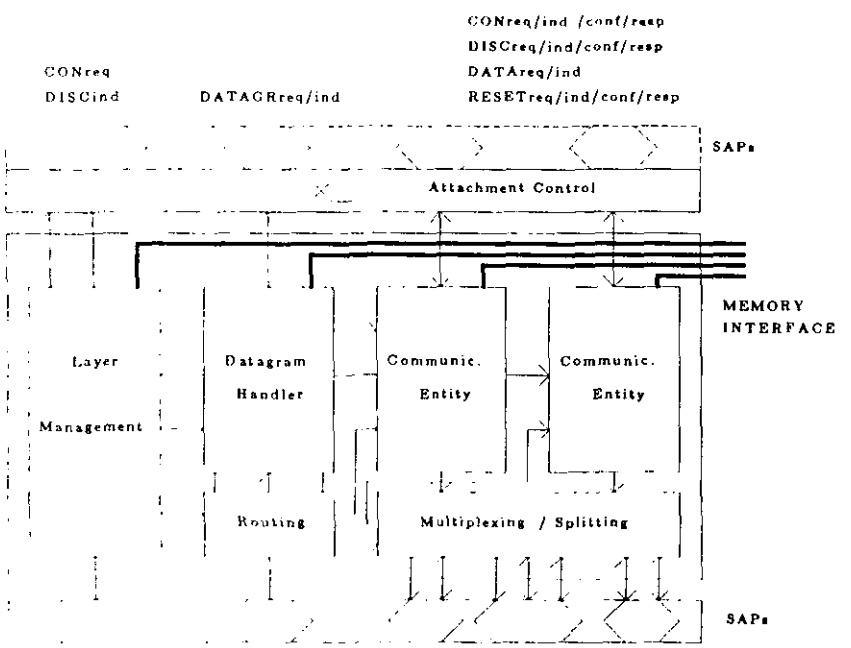

Fig. 4.1 Universal Protocol Subsystem architecture

In this architectural model, an UPS with two Communication entities is given. These entities can be considered as processes which can make several connections. However, only one connection can be operated at a time. These Communication entities are used for connection oriented communications in contrary to the Datagram entity, used for connection-less communication. We will discuss a subsystem with two such Communication entities. A subsystem with more than two Communication entities is in essence not different from this one. A subsystem with only one Communication entity does not need as many management functions as we will discuss here.

The SAPs and attachment control define the interface to adjacent layers. Implementations like we discussed in chapter two can be used here. In the model, one centralised routing function is used. Each of the blocks will be discussed in the next sections.

4.1 SAPs and attachment control

Every entity within a layer has one SAP below it. Only the splitting block has more access points to the subsystem below. These SAPs are attached to the attachment control unit of the lower layer which controls attachment of the SAPs to the entities within the subsystem below. Initially, all SAPs are attached to the Management entity of the lower layer. Only the SAP of the Datagram entity of the layer on top is directly attached to the Datagram-entity, or -entities, of the lower layer. The management entity overviews all access points alternately, and when a CONreq is noticed, it starts an entity, initializes it and activates it. This entity now establishes a connection to the lower layer by itself. The attachment control connects the entity to the SAP of the entity on top which asked the connection. For that purpose, it performs a switching function. Layer Management does not control this SAP any more until it receives a message that the entity is stopped. Although attachment control has been model separate from Layer Management, it is actually an I/O Management task (see chapter 9).

4.2 Layer Management

There are five types of concern for the Layer Management which are : Fault management, Configuration management, Performance management, Security management and Accounting management. Also remote control has to be possible. For this purpose, exchange of management information is needed via a so called management protocol, mostly a stop and wait protocol using datagram frames.

4.2.1 Fault Management

If Layer Management notices that a connection cannot be made within a certain time, it has to report this to the entity which has asked this connection. Also if Layer Management notices that all the connections within its layer are in use, it can report to other management entities that no more connections can be made. In that case a congestion is noticed as early as possible by other entities, and routing information can be changed. Fault management also deals with testing the layer and the underlying layers.

4.2.2 Configuration Management

After entities within the layer are created or deleted, the configuration of the layer is changed. This has to be scheduled and possibly reported to other management entities.
If a new connection has to be established, entities have to be created, initialized and activated. After connection release, the entities can be "deleted".

4.2.3 Performance Management

The Performance management monitors the effectiveness of the entities and schedules the connections. If special services are asked, this function can decide whether the QOS can be given and how this has to be done.

4.2.4 Security Management

If and when an entity within another layer asks a connection to an entity within this layer, security management checks whether the entity is allowed to make that connection. If the connection is not allowed, a disconnect primitive is sent.

4.2.5 Accounting

After activation of an entity, accounting can be started. As only Layer Operation of an entity notices that the entity is connected, this function should preferably be performed by that management process.

4.3 Datagram entity

The Datagram handler controls connection-less transfer of data frames. No connection will be established in advance. A data frame is extended with a header containing the address of the receiver. Similar to mailing of letters, this frame will be transmitted and a response frame from the receiver is required in order to acknowledge the frame. Sequenced delivery cannot be guaranteed. This type of data transfer can be used on links with low error rates. Because the OSI MOdel does not contain connection-less data transfer, we will not discuss the related functions in this report in detail.

4.4 Routing

The routing process is quite a straightforward process. If an address from a CONreq or from a CONind is given to this process by Layer Management or by Layer Operation, the process generates routing information by using a hierarchical routing table. This information contains the SAP-address of the peer entity connected to the addressed entity, or the first entity in route to this entity. The information can also contain the name of the entity in the layer which has to be used to make the connection. Routing information has to indicate as well whether a CONreq or a CONind has to be sent. For example, if a CONind arrives with an address of an entity not in the corresponding system, a CONreq to another system has to be sent. The entity now operates as a relay entity. If an address is given to the routing process for which there is no information in the table, an error message is generated to show that the address is unknown.

In order to update the routing table, e.g. change the hierarchy in case of congestion, Layer Management can give new routing information to the routing process.

As nearly all entities within a layer require routing information to perform their tasks, the routing process has to be extended with an arbiter process which handles the requests.

4.5 Multiplexing/splitting entity

In this model, the multiplexing entity can multiplex two connections onto one connection. Through a SAP, only one connection can be operated at a time however. In case of multiplexing, extra identifiers have to be added to the data-units in order to address the units to the right entity. The multiplexing entity therefore must add these identifiers to a DATAreq and read them from a DATAind. The entity cyclic looks at each SAP of the entities on top, and serves each entity in turn. An entity which multiplexes more than two connections onto one or more connections works the same.

One connection can also be mapped onto more lower layer connections in order to improve the reliability, provide the required grade of performance or obtain cost benefits by utilization of multiple low cost lower layer connections, each with less than the required grade of performance. In case of splitting, some associated functions are required as are scheduling the utilization of the lower layer connections, and re-sequencing of the PDUs associated with the connection, since they may arrive out of order, even when each lower layer connection guarantees sequenced delivery.

Extensions to processes that are required for multiplexing and splitting will be discussed in chapter 8.

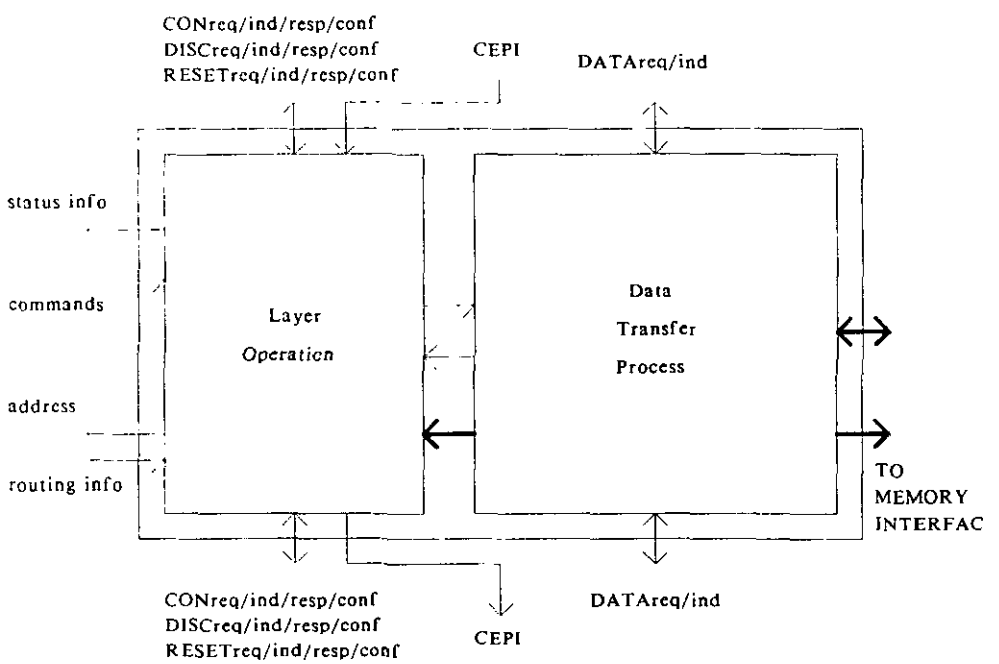

Fig. 4.2 Architecture of a Communication entity

4.6 Communication entity

A communication entity has been subdivided into a Data Transfer process and a Layer Operation process. The Data Transfer part executes the transfer of data and controls the dataflow and sequencing. Layer Operation is the control unit of the entity. It initializes the functions in the Transfer process or changes them if necessary. Establishment, release and restart of connections is controlled by Layer Operation. Also if more connections are served by one entity, the switch from one connection to the other is under control of Layer Operation.

5 DATAFLOW AND TRANSFORMATIONS

In the architectural model, we have subdivided the subsystem into several blocks, each performing its own function. In the following sections, we will describe the dataflow and transformations within these blocks in more detail. For this purpose we will use a method described by Hatley and Pirbhai [12], [11]. In this method, each process is modelled as a transformation. Data to and from these transformations is modelled as dataflows. Hatley and Pirbhai separate data and control signals. Since we model the processes at such a high level of abstraction, we mostly do not want to make distinction between these two. Only in some diagrams, control signals have been modelled. These signals are however treated similar to data signals. Control will be performed by the processes themselves, instead of by a special control unit.

At the top level, the UPS has been modelled as one process. Service primitives only can be transferred to this process and invoked by this process. Additionally, an external management process can give commands to this process directly [22], (see also figure 8.1). In order to be able to cope with the complexity of the problem, the subsystem will be decomposed into a number of sub-processes which can operate in parallel. (see figure 5.1).

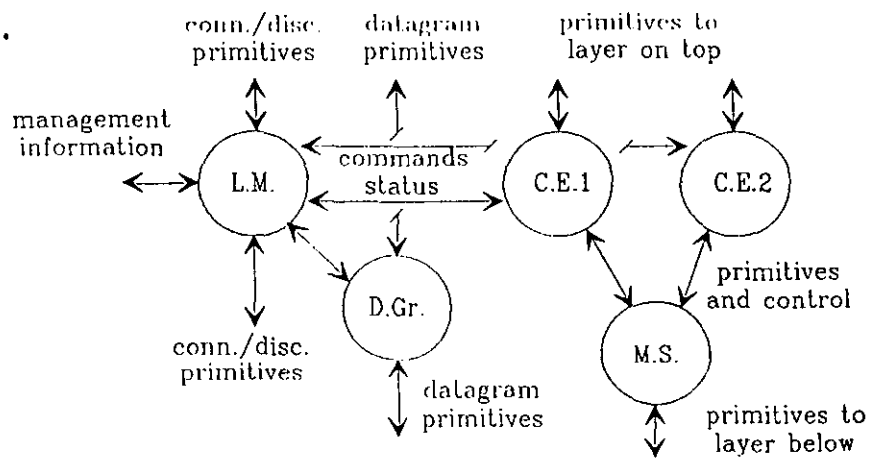

Fig. 5.1 Dataflow diagram of a UPS

In the following sections, we will describe each of these sub-processes in more detail. In chapter six, the same descriptions will be given more formally.

5.1 Layer Management

Layer Management is activated by either a connect-primitive or by an indication that a datagram frame has been received for this process. After an entity within the subsystem has stopped, this has to be reported to Layer Management so configuration information can be updated. The external behaviour of this process is modelled in an abstract way in section 6.1. Internally, the process can be separated into four

sub-processes (see fig 5.2). Only the Performance / Configuration Management (PCM) process will be discussed in more detail in the next section. The other processes are less complex.

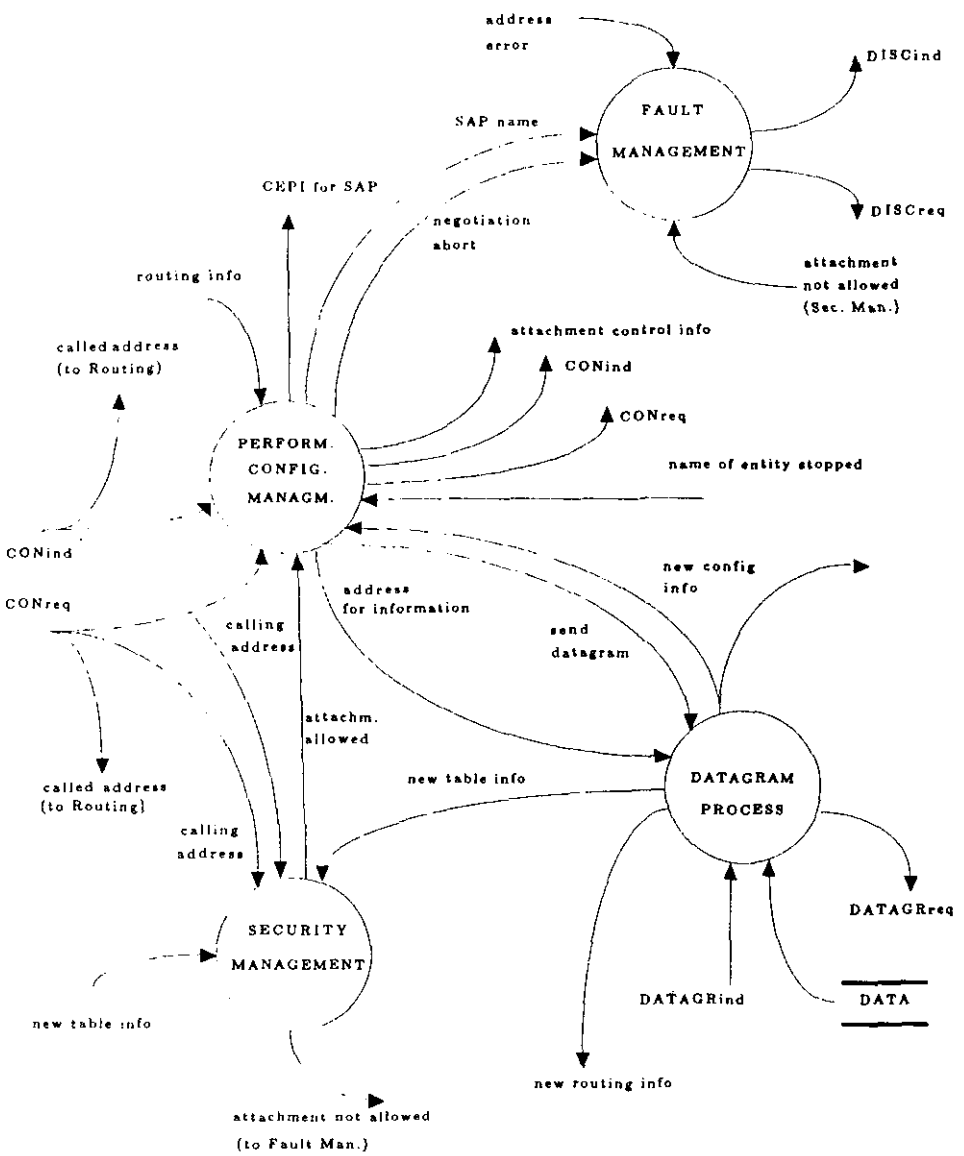

Fig. 5.2 Layer Management Data Flow Diagram

5.1.1 Performance / Configuration Management

The PCM process performs the three main management tasks which are Resource Management, Memory Management and I/O control. If a new connection has to be established and establishment is allowed, all related control functions will be executed by this process. Negotiation about the quality of service (QOS) is started. If this negotiation terminates successfully, entities will be started and initialised or modified. This is done via the signals named "CONreq" and "CONind" in the dataflow diagram. Also, the entities have to be attached to the SAPs, and memory must be allocated to these entities. After all these functions have been executed, the new configuration may need to be stored and monitored to other systems. For this purpose, datagram frames can be used. Consequently, datagram frames can be received containing new configuration information of other subsystems within the system, or information of other systems. If during establishment an error occurs, e.g. unknown address or unauthorised connection request, establishment is ended, and Fault Management handles the error notification.

The PCM process has to make very complex decisions, however, the external communication is rather simple. Therefore, we will not model this process in more detail using this method. In chapter eight we will discuss the tasks of this process in relation to all other control functions within the subsystem.

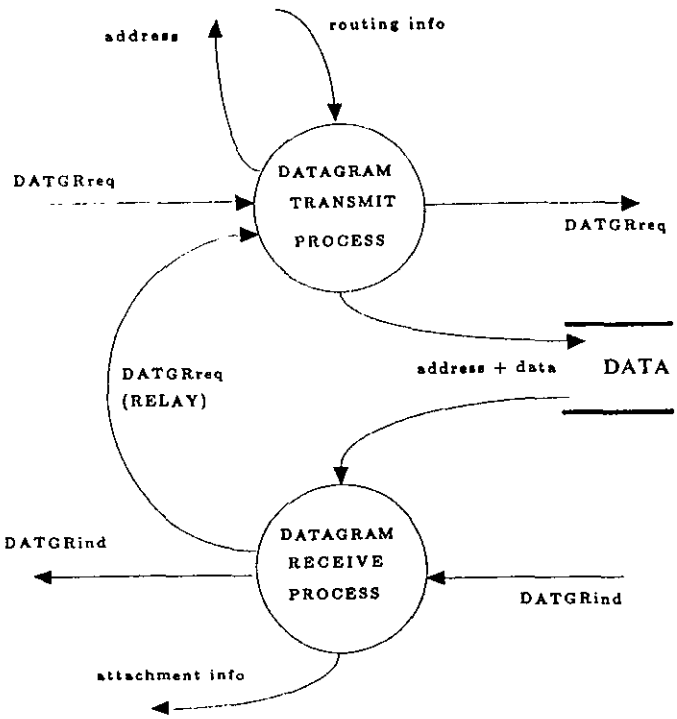

Fig. 5.3 Decomposition of Datagram process

5.2 Datagram Process

The Datagram process within the subsystem controls connection-less transfer of data frames. Correct or sequenced delivery can not be guaranteed, and each frame has to be acknowledged individually if required. If links are used with a low error rate, this transfer method can be used together with negative acknowledgement. We will assume the lower layer also provides datagram services, so no connections have to be established and released as mentioned in section 4.1.3. In that case, the process is subdivided into a transmitting and a receiving part.

5.2.1 Transmit process

A datagram frame has to contain the address of the receiver. This address will be sent to the Routing process which returns an address of the SAP of the entity below of the receivers address, or of the entity first in route to that entity. This address is stored ahead of the frame, and a datagram request will be invoked to the lower layer. If the frame has to be acknowledged by the receiver, this has to be indicated in a control field within the frame transmitted. If this acknowledgement does not arrive within a certain time, indicated by a timer, the frame may be re-transmitted. A special field must show however that this frame is a re-transmission.

5.2.2 Receive process

The occurrence of a DATGRind activates this process. The process reads the header from memory. If the address is unknown, the frame is given to the Transmit process together with a DATGRreq, the entity now operates as a relay process. Otherwise the header is stripped and a DATGRind is given to an above entity. Attachment control will be provide with the routing info for this indication.

5.3 Layer Operation

Connection oriented data transfer is actually performed by the Communication Entities (C.E.). If an entity has been started, it can operate parallel to, and independent of all other entities within the subsystem. Each Communication Entity can be subdivided into a Layer Operation process and a Data Transfer process. Layer Operation performs control functions of the Communication Entity. Therefore, all control primitives and commands from Layer Management are transferred to this processes. If multiple connections are operated quasi-parallel, this process will schedule the different connections and will serve them alternately. Layer Operation is subdivided into four parallel processes (see figure 5.4) corresponding to the phases of a connection.

5.3.1 Connect process

The Connect process controls establishment of new connections and re-establishment of restarted connections for which the lower layer does not provide restart services. For this control, all connect primitives, connect frames and reconnect signals

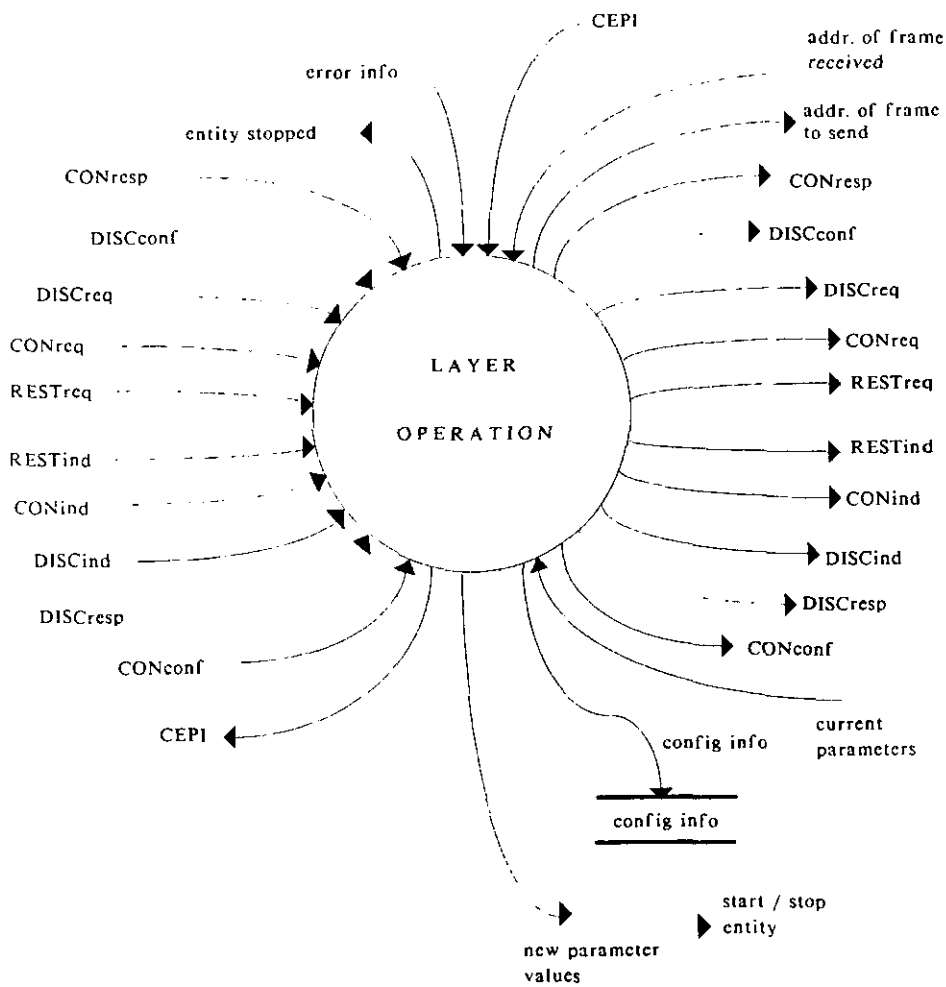

Fig. 5.4 External signals of Layer Operation

will be transferred to this process. After each of these signals, the process executes different functions and shows different behaviour. These different behaviours and functions will be discussed in the following sections.

5.3.1.a CONreq(calling addr., called addr., QOS, data)

After occurrence of a CONreq, the called address and the QOS are read from this primitive. During negotiation about the QOS parameters, the SAP address can be sent to the routing process. After successful negotiation and receipt of routing information, a header is made for the connection frame and is stored in memory. If an address-error occurs, an unrecoverable-error signal will be given and establishment is stopped. If no error occurred and the entity has not been attached to a lower layer entity, a CONreq

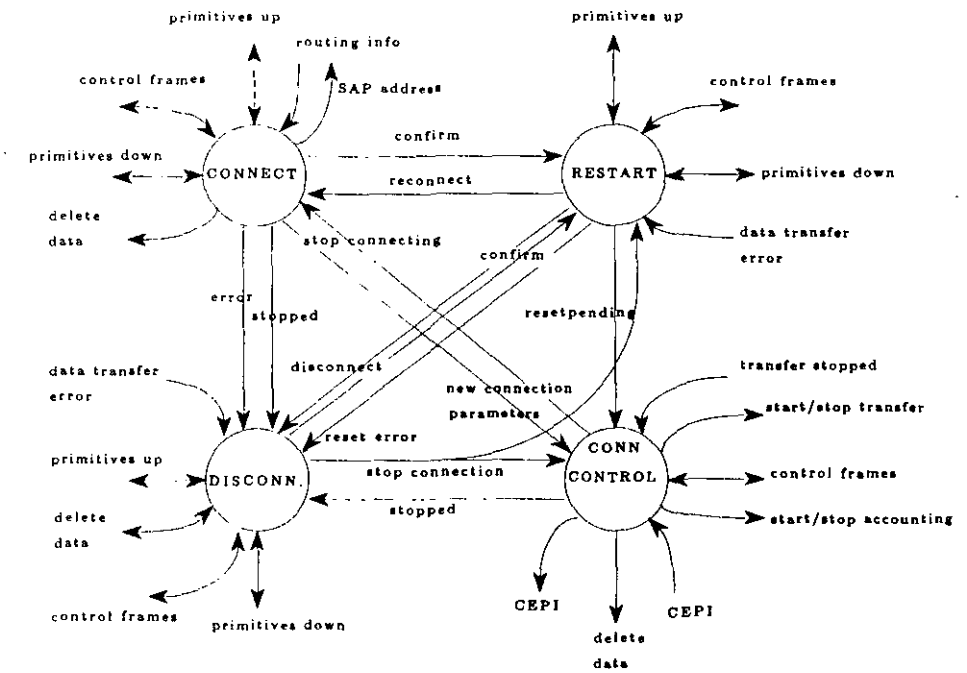

Fig. 5.5 Dataflow diagram of Layer Operation

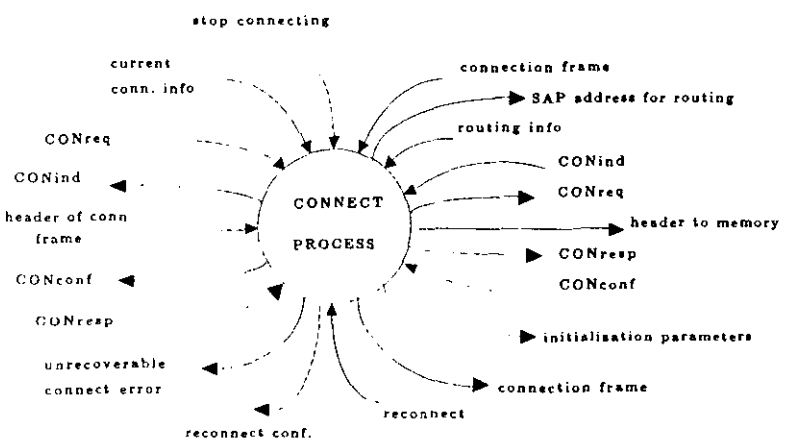

Fig. 5.6 Dataflow diagram of Connect process

is given to Layer Management of the lower layer. If the entity is already attached to a lower layer entity, the CONreq is given to this entity or the connection frame is transferred in a data packet in case the entity is already connected to the entity called. The entity can also activate the splitting process so a second SAP connected to another entity in the lower layer can be used. To be sure establishing of the connection will succeed within a certain time, a timer is started after the CONreq is given or the connection frame is sent. If and when this timer expires, a new CONreq can be given or an unrecoverable-error signal is given, depending on the protocol. The connection is established if a CONconf arrives after a CONreq or a connection confirm frame is received in a data packet. In that case, the Connect process sends initialization parameters to the Connection Control process corresponding to the QOS parameters in the confirmation. A CONconf together with the stripped frame is given to the entity on top, and the connection request header can then be deleted from memory.

5.3.1.b CONind(called addr., calling addr., QOS, data)

If the entity has not been attached to the underlying layer already, Layer Management within this underlying layer can invoke a CONind to the entity. Also an attached entity within the underlying layer can invoke a CONind. In these cases, the included called address, calling address and QOS are read from memory. Routing information will be obtained from the address via the Routing process, and the QOS parameters have to be examined. A CONind has to be given to the layer on top containing the right QOS parameters and addresses. A returned response contains the final parameters for the initialisation of the Data Transfer processes. Concurrently with the initialisation, a CONresp can be invoked. If the CONresp from the layer on top does not occur within a certain period, or a disconnect primitive arrives, the Connect process will stop and send an unrecoverable error signal. The Disconnect process control rejection of establishment in that case.

If Layer Management notices that the address of the called SAP is not in the system, it can start two entities in its layer which have to operate as a relay. In that case the two entities will be attached to each other via the attachment control process. The CONind is given to one entity without the user data and a CONreq is given to the other entity with this data. The connection is established in the two directions, and the Data Transfer Process is initialized to operate as a relay.

5.3.1.c Connect frame in memory

While an entity is communicating, it can receive a connect frame within the data stream. This connect frame is given to Layer Operation by the control unit of the Data Transfer Process, and is handled in exactly the same way as a CONind. The CONresp will not be sent to the lower layer this time, but will be sent via the control unit within a data frame to the entity at the other side. This situation can only occur if an entity can make more connections through the same SAP.

5.3.1.d reconnect

The restart process can force to restart a connection by releasing it and afterwards re-establish it. This re-establishment will be forced by a reconnect signal. After this command, the connect process acts the way it usually does after a CONreq. It uses the current connection info as parameters. The establishment confirmation will be sent to the Restart process instead of to the layer on top.

After a stop signal, the connect process stops all activities and removes all headers from memory which have been stored but not yet sent.

5.3.2 Disconnect process

The Disconnect process controls release of the connections. This process is activated by either a disconnect primitive, an unrecoverable-error signal from the Connect process, a received disconnect frame or a signal from the Restart process to disconnect the current connection. After each signal, the process has to execute different functions as described below.

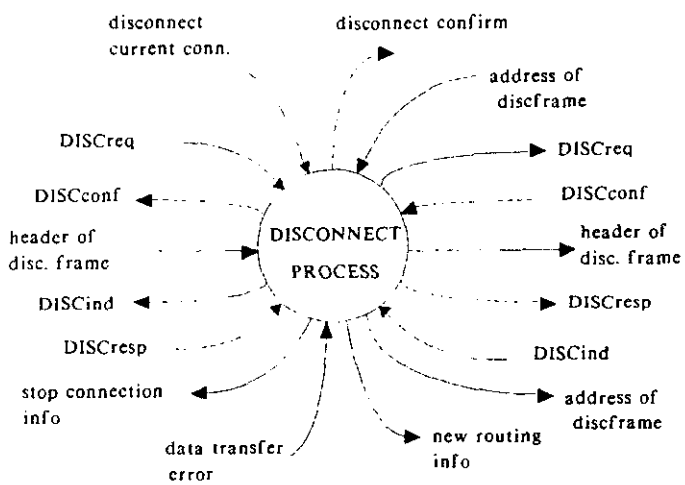

Fig. 5.7 Dataflow diagram of Disconnect process

5.3.2.a DISCreq(reason, data, responding addr.)

{ The responding address is present only if the primitive is used to indicate a rejection of a connection establishment. }

After a DISCreq, the process either sends a DISCreq to the lower entity after storage of a disconnect header in memory, or sends a disconnect-frame to the control unit of the Data Transfer Process. After that, the process waits for a DISCconf or a response on the disconnect frame. Release of the connection is mentioned to the Connection Control Process which in that case removes the corresponding data from

memory. A DISCconf may have to be sent to the entity on top. Note that a DISCconf does not contain data and gives no extra information. Therefore, most protocols omit both DISCconf and DISCresp. If the confirmation does not arrive within a certain period, the process restarts or releases the connection without confirmation.
5.3.2.b DISCind(originator, reason, data, responding addr.)

After a DISCind, the included parameters are examined and, for example, new routing information can be given to the Routing process. A DISCind has to be given to the entity on top or, in case of a relay system, to the other entity as a DISCreq, and the process must wait for a DISCresp. After reception of this response, information about the released connection will be sent to the Connection Control Process and a DISCresp will be invoked to the lower layer.

5.3.2.c Disconnect frame received

After reading the header of the frame, the Disconnect process acts like it does after a DISCind. If the header shows that the disconnection was forced to establish a reset, a RESTind must be given to the above entity instead of a DISCind. The Connection Control Process knows from the stop connection info that the connection will be reconnected. In this case, the process has to wait for a RESTresp. In case of connection release in stead of reset, the process has to wait for a DISCresp.

5.3.2.d Unrecoverable connection error

If an error occurs during connection establishment, the connection process stops all activities, and other processes have to be notified of rejection of establishment. If a CONreq is pending already, the process acts the way it does after a DISCreq, and a DISCind will be given to the entity on top. If no CONreq is pending, only a DISCind will be given. In order to preserve more connection errors, routing information can be updated.

5.3.2.e Disconnect current connection signal

The Restart Process can decide to restart a connection by releasing it and after that reconnect the entities. This will be forced by the above disconnect signal which corresponds to a DISCreq. Therefore, the process acts the same way as after a DISCreq. The only difference is that in the header of the disconnect frame or primitive a parameter has to show that the release was forced to establish a reset. The entity at the other side has to invoke a RESTind to the layer on top, and must wait for reconnection.

5.3.2.f Data transfer error

If the Data Transfer process notices an error which it cannot resolve from, an error signal can be sent to either the Restart process or to the Disconnect process. The

Disconnect process will act after such an error similar as after a DISCreq from the entity on top. In stead of a DISCconf however, a DIscind will be invoked and a response is waited for.

5.3.3 Restart process

The Restart process will be activated when a reset primitive occurs or when a reset frame is received by the Data Transfer process. The Data Transfer process can also force a reset itself by giving a data-transfer-error signal. This will be done in case of errors within the Data Transfer process.

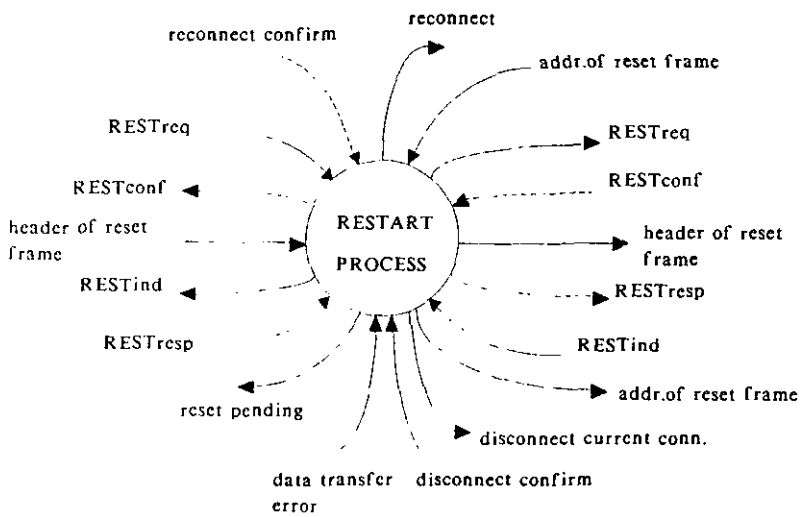

Fig. 5.8 Dataflow diagram of Restart process

5.3.3.a RESTreq(reason)

An entity within the layer on top which invokes a RESTreq, wants to reset data transfer over the connection. In that case, all pending frames have to be removed, a reset frame must be sent to the entity within the other system and a RESTreq may have to be given to the lower layer entity. If the lower layer does not provide reset services, it may be necessary to release and re-establish the connection. In that case, all data frames pending will be removed automatically. After the connection has been reset, no more data will arrive before a RESTconf, thus after reception of a RESTconf, a RESTconf can be given to the entity on top too. This confirmation only indicates that data transfer can be restarted. The primitive gives no extra information. The layer on top also can see the acceptance of the request as a confirmation. At the moment that the DATAreq is accepted, the entity knows that the connection has been reset. This is also noticed at the moment that a DATAind arrives.

If a reset frame is sent without disconnecting or resetting the lower layer connection, the process has to wait for a reset response frame. Meanwhile, no data can

be accepted from the layer on top and all data that is received for the layer on top has to be deleted. After reception of the reset response frame, a RESTconf can be given but special care must be given in order to be sure that no data will be transferred that was sent before the reset. This can occur if the lower layer does not guarantee sequenced delivery (e.g. in case of splitting). In that case therefore, the lower layer entities has to be informed of the reset. If this is not possible, the only solution is release and reestablishment of the connection.

5.3.3.b RESTind(reason, originator)

A RESTind indicates that the underlying entity resets the connection, removing all data from memory that has not been acknowledged already. If this primitive arrives, the Data Transfer process has to be restarted, and thus, a reset signal has to be given to the Connection Control process. All data that has not been acknowledged yet, has to be resent. A RESTresp can be given to the lower layer concurrently with the reset signal to the Control Process. This primitive gives no extra information and thus may be omitted. In that case, acceptance of the RESTind can be seen as a response.

5.3.3.c Reset frame received

If a reset frame arrives within the data stream, it will be transferred to the Restart process. This frame indicates that the entity at the other side resets the connection. This can either be in one direction or in both directions. The reset in the opposite direction has to be performed by this entity. Depending on the protocol, a reset response frame has to be sent. The entity has to remove all frames from memory and must give a RESTind to the layer on top. Data transfer can be restarted after a RESTresp.

5.3.3.d Data transfer error

When the Data Transfer process detects an unrecoverable error, the connection can be reset or disconnected. If a reset is asked, the Restart process acts like it does after a RESTreq but will invoke a RESTind to the entity on top instead of a RESTconf.

5.3.4 Connection Control process

The Connection Control process directly controls the Data Transfer process. It initialises this process before connecting, starts and stops transfer and it can send control frames. If two entities have been connected, it also controls accounting.

After connections have been created, this process receives initialization parameters from the Connect process. These parameters have to contain initial values of transfer functions and identifiers of SAP and CEPs. Corresponding to the CEPI within the SAP on top, the Data Transfer process will be initialised, the CEPI within the SAP below is set, data transfer and accounting are started. If the CEPI changes, a new connection has to be operated. For this purpose, the values of the current connection have to be saved and the values of the new connection have to be loaded (i.e. sequence numbers, window sizes, data addresses). Possibly, some control frames have to be sent

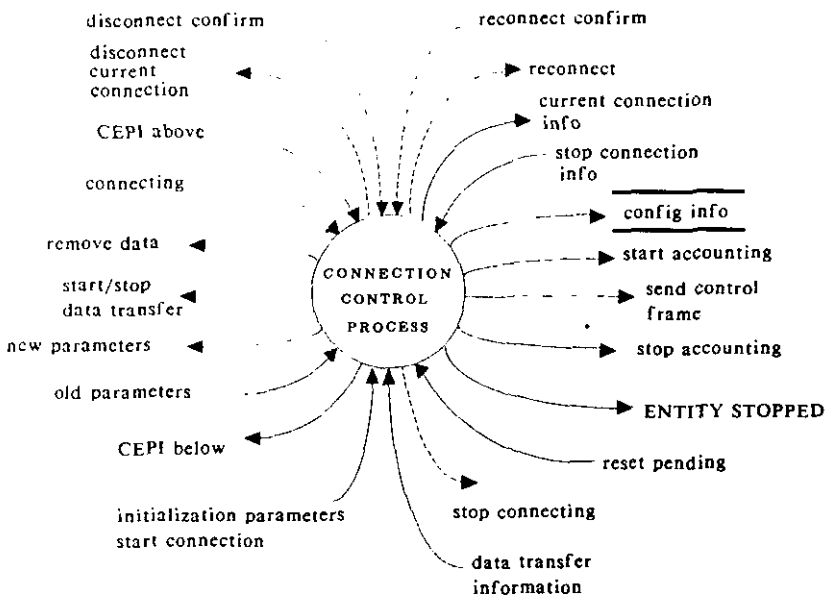

Fig. 5.9 Dataflow diagram of Connection Control process

before stopping the previous connection (e.g. RNR) and before starting the new connection (e.g. RR).

After a "stop-connection-signal", the parameters of the connection can be removed in case the disconnect was not forced in order to reset the connection. If the entity was still connecting, the Connect process will be stopped. In all cases, data in memory has to be removed and if no more connections are operated, a signal has to be given to Layer Management which indicates that the entity has stopped.

After a signal indicating that a reset is pending, the Data Transfer process will be stopped, re-initialised, and all data has to be removed in case the signal parameters says to do so. The "current connection" parameters have to be sent to the Connect process in case the connection is restarted by means of disconnection and re-establishment.

In some protocols, a connection will only be established at the moment that data has to be transmitted. After data transfer, the connection will be released immediately if no more data is received from the layer on top, or from the layer below during a certain period. After occurrence of a DATAreq, the connection will be re-established in the same way as after a restart.

5.3.5 Accounting

Within some layers, accounting has to be performed during communication. Since costs are calculated only during actual attachment, accounting has to be stopped when the connection is suspended. The Connection Control process handles this suspension and scheduling, and therefore also has to control accounting. Within the Layer Operation process, an accounting process has to be executed. This process has not been modelled in the diagram. Only the signals to this process have been modelled. Additionally, an external management unit must have access to the accounted cost information.

5.4 Data Transfer process

During the data transfer phase of a connection, the transmission and reception of data frames is supervised by the Data Transfer process within the Communication Entity. SDUs from the layer on top have to be transformed into PDUs. These PDUs have to be transmitted to the peer entity. Analogue, PDUs from the peer entity have to be transformed into SDUs which have to be transferred to the layer on top. Control over the Data Transfer process is performed by Layer Operation. In section 3.1, we have listed the functions that have to be controlled. Some of the functions listed there will be executed by other processes. E.g. multiplexing and splitting. As transfer of expedited data does not differ significantly from normal data transfer, except some administration and memory management functions, we will not discuss this option in this report.

In figure 5.13, the processes within the Data Transfer process are described. Each process performs its own functions. Some of the functions have been combined in one process. E.g. error detection / correction can be performed by the decoding process. In the next sections we will describe each of the processes in more detail. Some processes contain a timer which can be used by that specific process only. In the diagram, this is shown by a circle inside the transformation containing a "T".

5.4.1 Sequencing

Layer Operation initializes the length of the sequence numbers and the window size via the "new values" signal. A "new-signal" from the Blocking / Segmenting process activates the process to generate a new sequence number if the window is not full, otherwise the process will wait until the window decreases. The window will only decrease after some frames have been acknowledged by the peer entity. If these acknowledgements do not arrive in time, the Flow Control process controls retransmission of frames. After a frame has been received, values within the sequencing process will updated. The number of frames acknowledged via a received sequence number will be transferred to the Flow Control process. If there is no data to send during a certain period and frames have to be acknowledged by this entity, the process forces transmission of a control frame.

After a reset from the Flow Control process, the sequencing is restarted at the last acknowledged number. The Blocking / Segmenting process will be informed of the reset too. After a connection restart, sequencing has to restart at a predefined number, mostly zero. This can be forced by Layer Operation by stopping the process, and re-starting it with initial number zero. Since data frames that have been received before a reset response frame must be deleted,the Deblocking process will be forced to delete all frames that arrive before the response. In chapter seven, the behaviour of this process will be described formally.

5.4.2 Flow Control

This process controls the data flow in one direction by manipulating retransmission delay. If congestion occurs, the delay will be increased, and after acknowledgement of some frames, the delay can be decreased again, assuming the

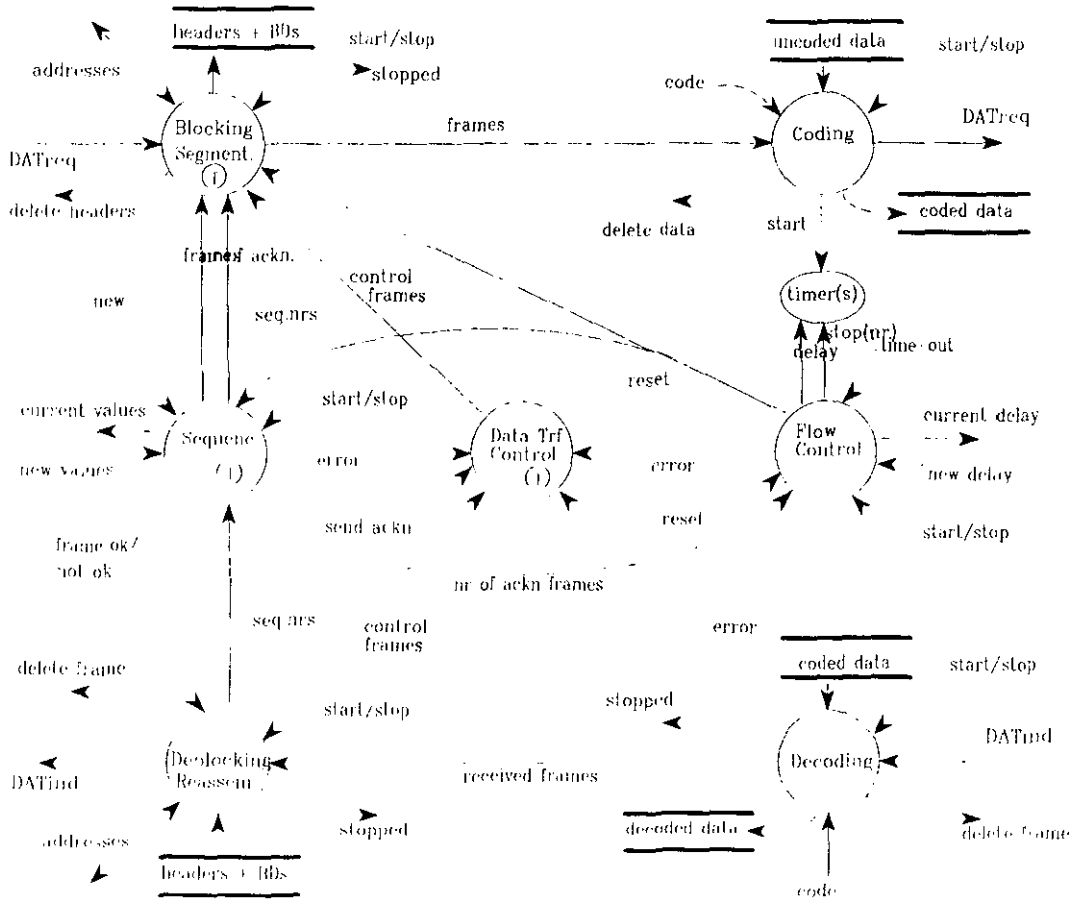

Fig. 5.10 Dataflow diagram of Data Transfer processes

congestion has disappeared.

Most protocols use one timer for every frame in the window. This means that the number of timers is equal to the window size. When using great window sizes, we have to implement these timers in software using one hardware timer. Each frame transmitted starts a timer and each acknowledged frame stops one. For this goal, a linked list of timers can be used [4]. After a time-out, the process sends a reset to both the Sequencing and the Blocking / Segmenting process. Another possibility would be to start a timer only if the window is full. In this case, only one timer will be required. The average time before an error will be noticed is increased by the time required to transmit a half window. After a time-out, the whole window must be retransmitted. If the window is not filled completely, the timer can be started earlier and restarted after transmission of a

new frame.

If the Flow Control process notices that no normal transfer of data is possible, it can force a restart or release of the connection. If the Flow Control process is stopped, the current delay can be saved by the Layer Operation process.

By using the above method of flow control and sequencing, only the go-back-N method of retransmission is implemented. For selective rejection of a frame, the reset signal should contain information about the kind of rejection. In the above case, retransmission will always start with the first not acknowledged frame.

5.4.3 Data Transfer Control

Errors occurring during the data transfer have to recovered by this process. Also, transmission and reception of control frames will be handled by this process. Within a data frame, control information can be sent to the Data Transfer Control process. This information may concerns flow control (e.g. RR, RNR, REJ,SREJ in LAPB) or establishment, restart or release of a connection. Most of the information will be routed to one of the Layer Operation processes. Only flow control information can be executed by the process itself. The process has its own timer for retransmission of the control frames. In most protocols, every control frame has to be acknowledged by a response control frame so one timer is sufficient. Because this process controls the protocol during the transfer phase, its structure is protocol dependent.

5.4.4 Blocking / segmenting

After a DATAreq, the corresponding SDU has to be transformed into one or more PDUs. For every PDU, the process has to create a header and must send the new frame to the Coding process. A "new-signal" to the Sequencing process will be sent indicating that a new sequence number is required. If segmenting is used, the frame will be subdivided into several blocks, and for each one a header must be generated together with a "new-signal". A special information field in the header has to indicate that segmenting has been used. Also, if blocking is used, this must be shown in the header. In case of blocking, the process will wait for more DATAreqs before a header is created and a frame is sent. Only if a DATAreq is not followed by another one within a certain period, the frame will be transmitted not completely filled. If a frame is segmented, Block Descriptors have to be generated (see chapter 7). If a frame is blocked, Block Descriptors have to be linked.

When the process receives a reset or a selective-reset command, one or more frames will be resent. For this purpose, all frames which have not yet been acknowledged have to remain addressable. Only after acknowledgement of frames, the header can be deleted from memory. Not the whole frame will be removed because an upper layer entity may want to save the data.

5.4.5 Deblocking / re-assembling

Decoded data received by the process is transformed into SDUs. In case only normal data has been received, headers have to be removed, sequence information must

be sent to the sequencing process and a DATAind will be invoked. In case control data has been received, this data will be transferred to the Control process.

5.4.6 Coding / decoding

In most layers, these processes are not required. Only at layer two, data is " coded" by a bit stuffing process. At layer four and two, some protocols use frame check sequences for error detection and correction. At layer six the data is coded using encryption methods. The processes thus not only encipher data but also performs all other bit oriented functions on the data. After the Coding process has transformed the data and invoked a DATAreq, it has to start a timer which will be used for re-transmission. The Decoding process transforms encoded data frames into uncoded PDUs. If during this decoding an error occurs, this has to be reported to the Control process, and the frame will be deleted.

6 FORMAL SPECIFICATION OF THE EXTERNAL BEHAVIOUR OF THE PROCESSES

In chapter five, we have described processes and subprocesses within the subsystem. Each of the processes described at the lowest level of abstraction executes a special function and shows a related external behaviour. This external behaviour has been described in an informal way in chapter five. In this chapter, we will give a formal specification of their behaviour. This formal specification shows the relation between the events and actions and provides us with the ability to verify the combined behaviour. We can compare this combined behaviour with the primitive sequences of chapter two and with other functional requirements. The abstract, formal specification, together with the specification of the internal decision, also provides a means of process simulation. For that purpose, the specification as given in the following sections, has to be implemented in software.

For the formal specification of the processes, we have used a graphical representation of CCS [17], together with concepts from SDL [18] and state charts. A graphical representation has been used because of the compressed form. A diagram gives better overview than the corresponding lexical representation. If we want to calculate the combined behaviour of processes, we are forced to model these processes using a lexical representation, e.g. CCS. Especially, each subprocess of the Layer Operation process has been described in CCS in order to calculate the combined behaviour of these subprocesses.(see Appendix B) Because of program limitations however, we could only combine two subprocesses at a time and prove absence of deadlock between these processes. To prove absence of deadlock in the overall process this is not enough. Therefore, we had to describe the subprocesses in a more abstract way. The corresponding combined behaviour has been calculated and did not show deadlock, so we may conclude that the processes described below will neither cause deadlock. The internal behaviour of the processes will be described shortly in commentary. A formal specification of this internal behaviour will be for further study.

6.1 Layer Management

Layer Management controls the subsystem. The internal decisions of this process are rather complex while the external communication is minimal. We therefore confine ourselves to giving the diagram of the traces of external actions performed by the process without further commentary (figure 6.1). In chapter 8, we will discuss the tasks of Layer Management more specified. In the diagram, attachment info not only contains information for the attachment control process but also contains information to start other entities within the layer (e.g. multiplexing / splitting).

6.2 Routing and Datagram process

As well as Layer Management do the Routing and Datagram processes have few external communications too. They are activated by one or two signals and do not require other communications to perform their tasks. As most processes within the subsystem do need routing information, the Routing process has to serve multiple processes. For this purpose, an arbiter has to be added to the process so no conflicts will arise. This arbiter has to control access to the Routing process. If a process, which asked

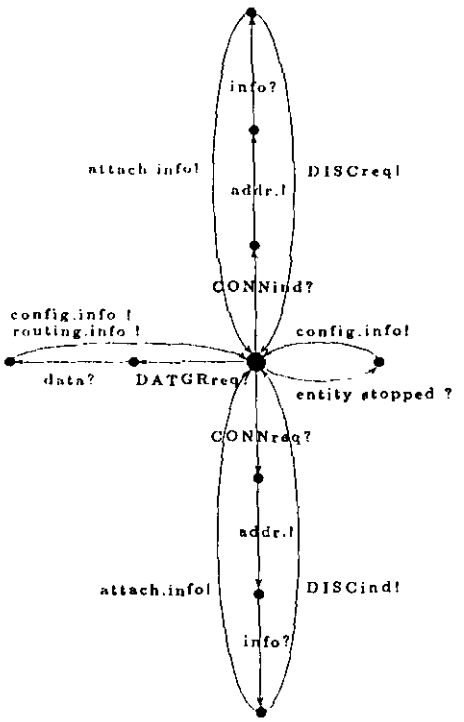

Fig. 6.1 External behaviour of Layer Management

information is deleted before this information is read, the arbiter has to delete the data to preserve deadlock, and must schedule another process. The Routing process and the arbiter will not be described in further detail. For solutions of related problems we refer to literature.

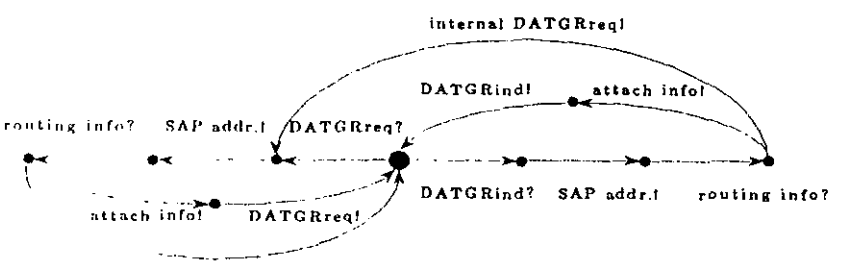

Fig. 6.2 External behaviour of Datagram process

The events and actions of the Datagram process are described in figure 6.2 below. In this diagram, transmission and reception of datagram primitives have not been separated although this would be possible. In that case, both subprocesses discussed in section 5.2 can operate parallel to each other. Requests and indications may arrive concurrently. An internal request however, must wait until the transmission process is ready for it.

6.3 Layer Operation process

In chapter five, we have noticed that the Layer Operation process is an important and complex process within a Communication Entity. All signals concerning connection control are transferred to this process. In order to cope with the complexity, we have subdivided the process into sub-processes, each performing one specific control function. Between these processes, a lot of information has to be exchanged for initialisation and synchronisation. In the next sections, we will specify each subprocess individually.

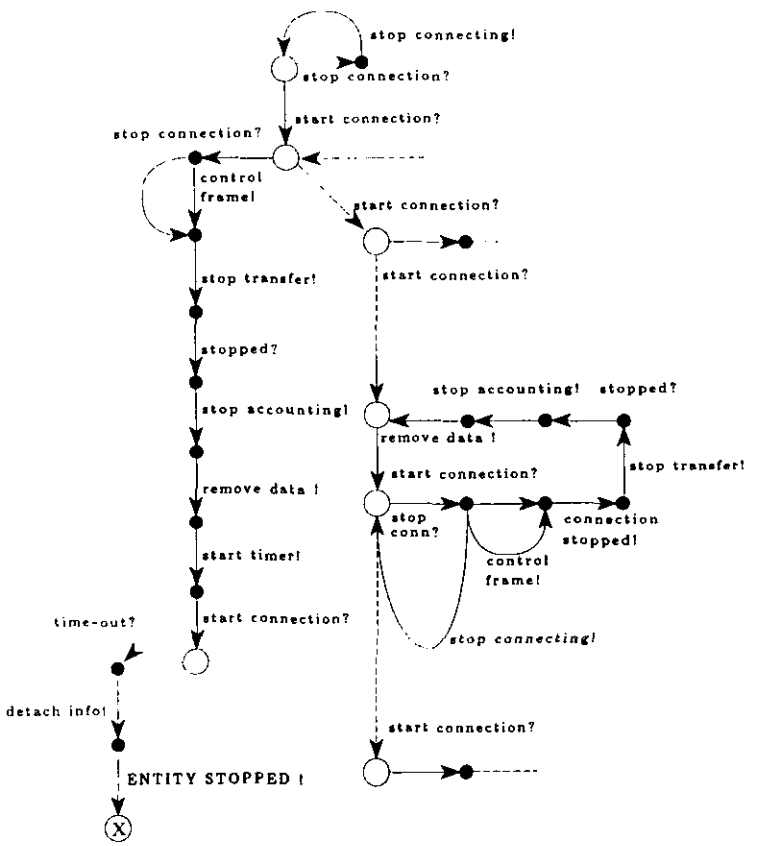

Fig. 6.3.a Creation and deletion of connections by Conn. Ctrl. process

6.3.1 Connection Control process

The external behaviour of the Connection Control process will be described in figure 6.3.a and 6.3.b below. Initially, the process will be in a "super state" represented by an open circle in the diagrams. If a new connection has been established, the corresponding parameter values have to be stored by this process. This has been modelled in figure 6.3.a via the "start connection" signal. If the Disconnect process wants to release a connection, the process deletes the corresponding values, or stops the Connect process via the "stop connecting" signal. Before a connection can be released, a control frame has to be transmitted, Data Transfer processes will be stopped and data has to be deleted. If no other connections are being operated, a timer is started. If this timer expires, the entity stops.

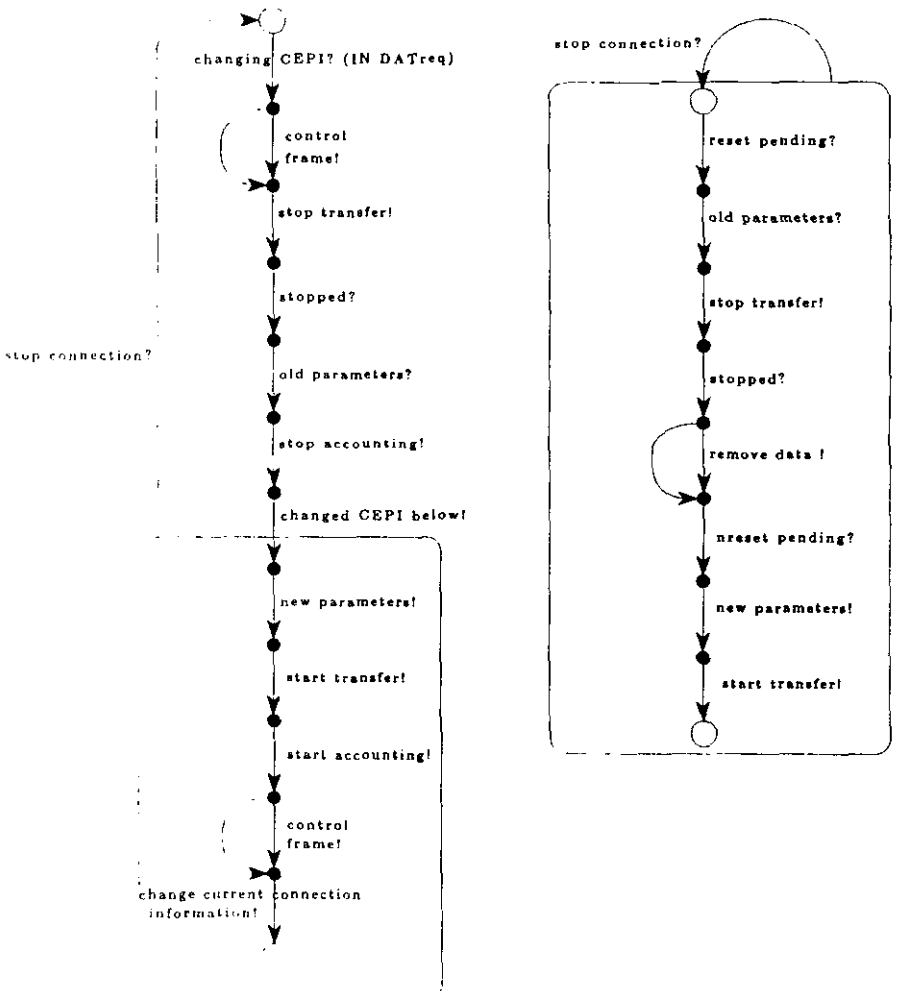

Fig. 6.3.b Connection switch and restart

As more connections can be served, an upper layer entity can change the CEPI within a SAP. This change will only activate the process if it is not currently deleting or changing the connection (see figure 6.3.b). After the "changing CEPI" signal, a control frame may have to be transmitted. Data transfer has to be stopped, current information about the process states and data has to be stored within the Connection Control Block (see chapter 8), and accounting has to be stopped. During these actions, the process may not be interrupted. After all values have been stored, the process may be halted in every consecutive state by the "stop connection" signal. This is shown in the figure by a rounded rectangle around these states. In order to operate a new connection, new values have to be sent to the Data Transfer process and accounting has to be started. Also, a control frame may have to be transmitted prior to the data transfer.

As well as a connection switch, a reset can be performed if the process is in a "super state" only. In order to preserve errors, immediately after a RESTreq or arrival of a reset frame, data transfer has to be stopped. No DATAreq can be received any more, and thus the connection cannot be changed before the reset has been executed completely. If the process is performing a reset, a "stop connection" signal for the corresponding connection will stop the procedure. In this case, the connection will be released.

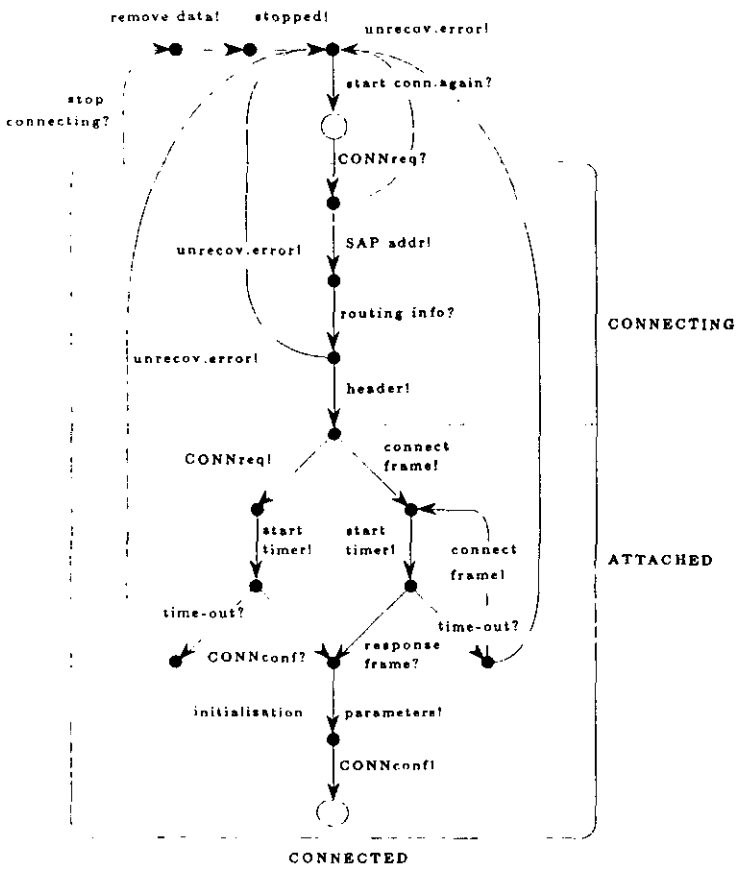

Fig. 6.4.a Behaviour after CONrequest

6.3.2 Connect process

Independent of the Control process, the Connect process can be activated by either a request primitive, an indication primitive, a connect frame or a reconnect command. After each of these events, the process shows a different external behaviour as will be described in figure 6.4.a..d. Until the moment the entities are connected, a stop-connection signal resets this process and all data which has been stored is deleted. This is shown in the figures by the "stop connecting" signal leaving from the rounded rectangle around the states. After this stop signal, dat may have to be removed and a confirmation has to be returned containing an indication of the state of attachment.

If an error occurs during establishment, the process stops and sends an error message to the disconnect process together with information about the command which activated the establishment. The Disconnect process now knows the reason of rejection and the actions which have to be performed. After the Disconnect process has executed all actions to release the connection, the Connect process can be restarted.

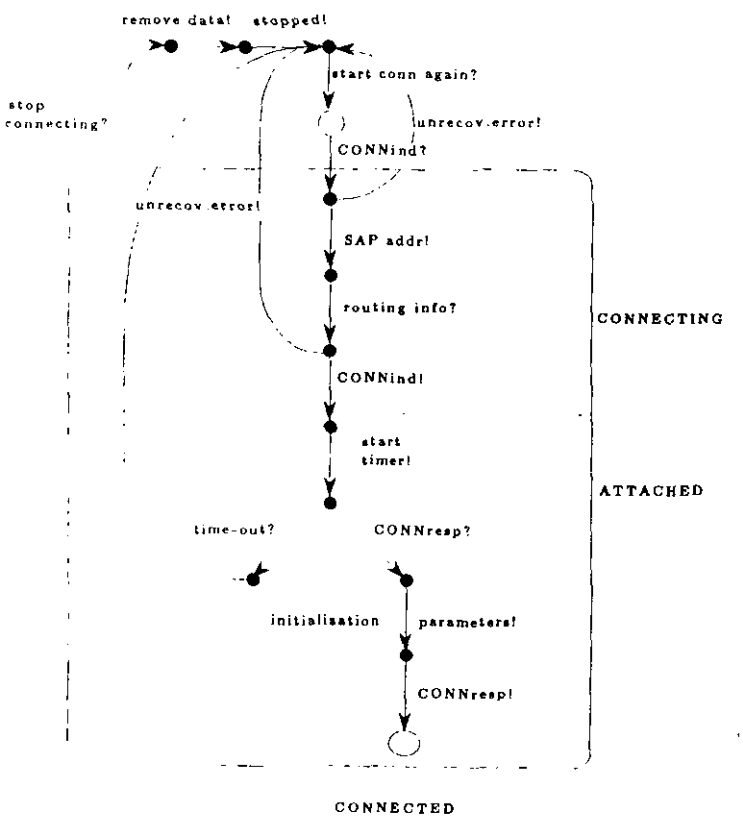

Fig. 6.4.b Behaviour after CONindication

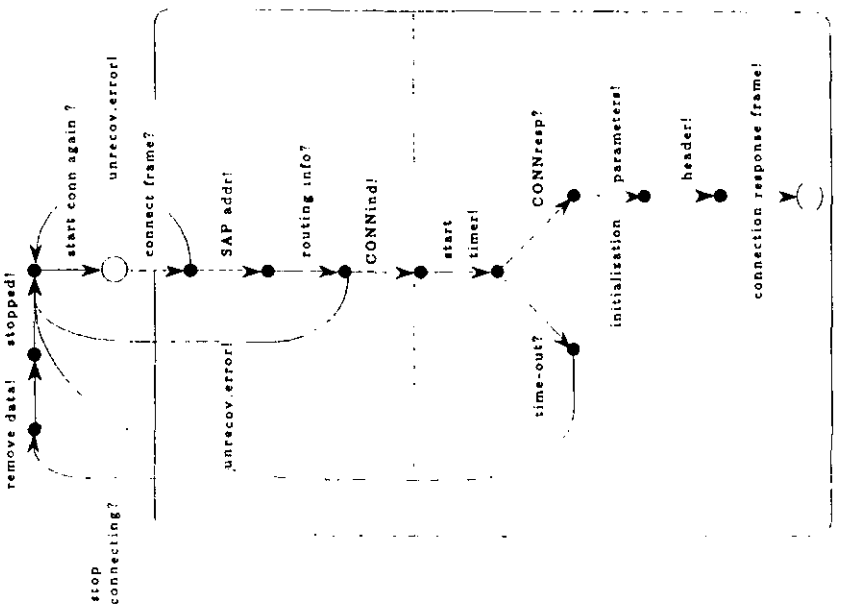

Fig. 6.4.c Behaviour after Connect frame received

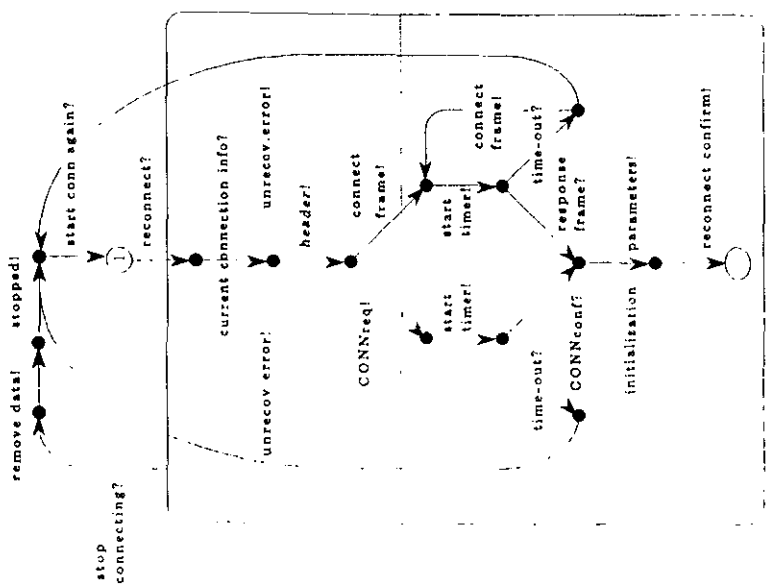

Fig. 6.4.d Behaviour after a reconnect signal

6.3.3 Disconnect process

The external behaviour of the Disconnect process will be specified in the next six figures, fig 6.5.a..f. Each figure describes the behaviour of the process after one of the six events that activate this process. These events are DISCreq, DISCind, disconnect frame received, unrecoverable error of the Connect process, disconnect command from the Restart process or a data transfer error from the Data Transfer process. The behaviour of the Disconnect process after each of these events has been described in chapter 5 informally. Therefore, we will present the "action traces" without much comment.

After a DISCreq primitive, the Connection Control process will be informed. The confirmation of the Connection Control process has to contain information about the state of the connection. If the connection had not been established yet, establishment has been stopped and a DISCconf can be invoked immediately. If the entities were attached, the connection has to be released.

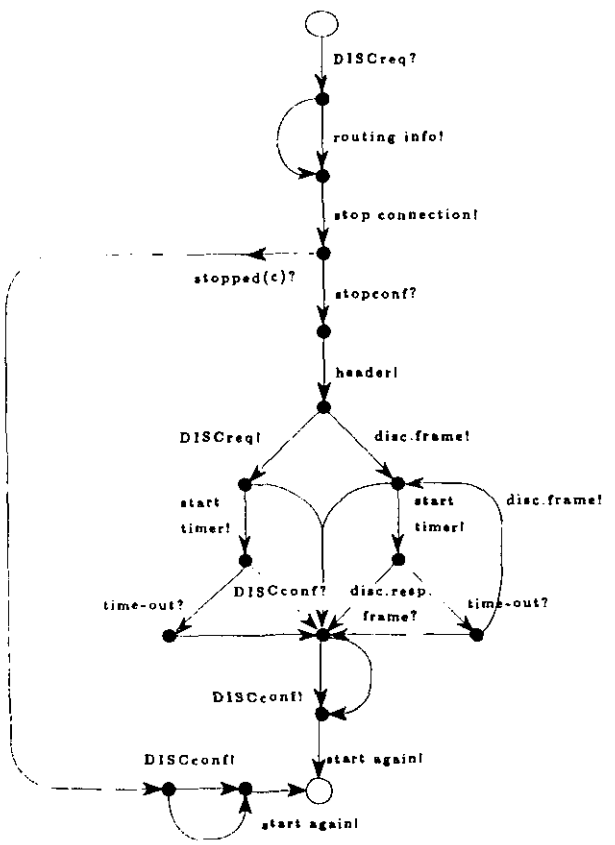

Fig. 6.5.a Behaviour of Disconnect process after a DISCrequest

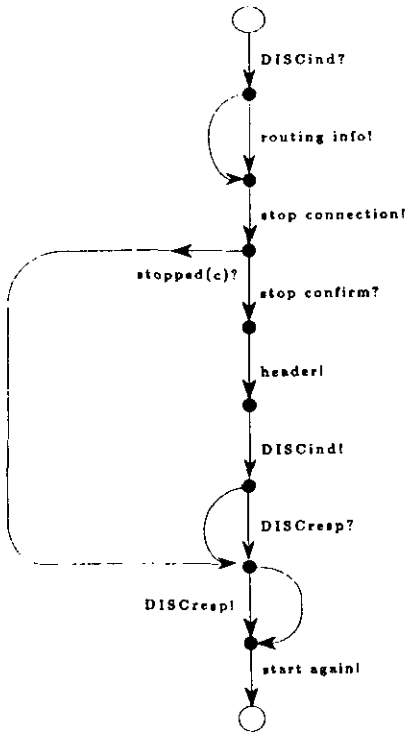

Fig. 6.5.b Behaviour after a DISCind

The actions that have to be taken after a DISCind primitive are specified in figure 6.5.b. for more comments about the behaviour of the process we refer to section 5.3.2.b.

After arrival of a disconnect frame, almost the same actions have to be taken as after a DISCind (see figure 6.5.c). A control field within the frame may indicate that the disconnect had been invoked in order to restart the connection. In that case, a RESTind has to be invoked to the entity on top instead of a DISCind. Also, the lower layer connection has to be released after arrival of a disconnect frame.

The "unrecoverable error" signal from the Connect process has to contain information about the state of the connection, including the direction in which the connection had to be established. Depending on this information, the process takes different actions to release the connection again. (figure 6.5.d)

In order to restart a connection, the Restart process may decide to release and re-establish this connection. Release will be forced by the "disconnect" command to the Disconnect process. Similar actions will be performed by this process as after a DISCreq. The confirmation, however, will be returned to the Restart process in stead of to the entity on top. (figure 6.5.e)

The behaviour of the Disconnect process after a data transfer error is similar as after a DISCreq (see also section 5.3.2.f)

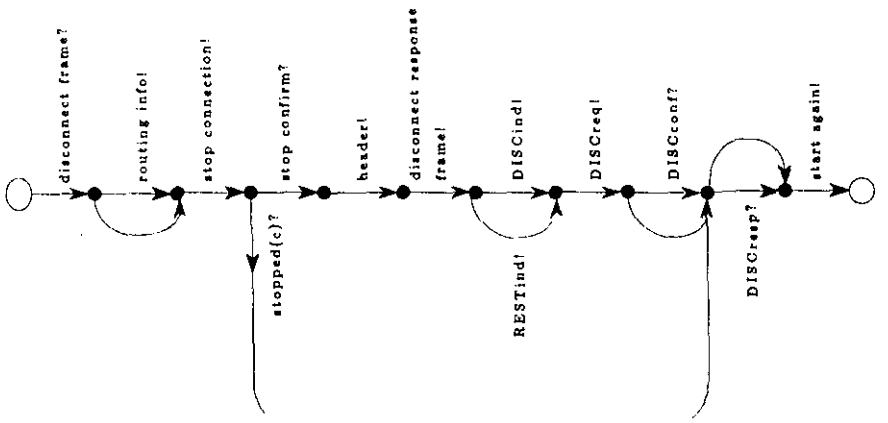

Fig. 6.5.c Behaviour after reception of a disconnect frame

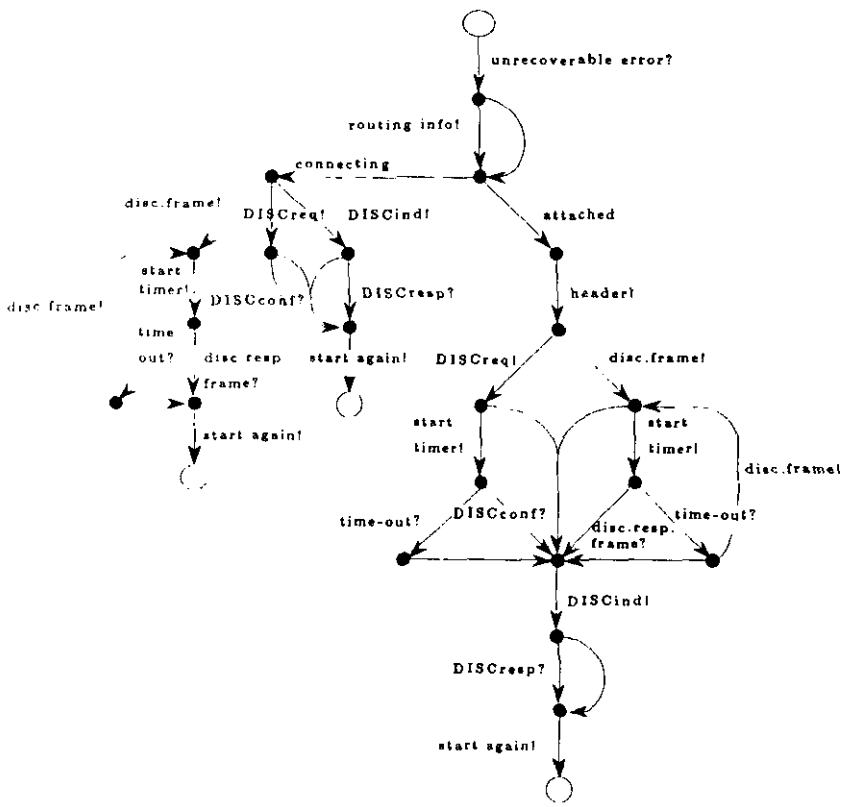

Fig. 6.5.d Behaviour after an unrecoverable error

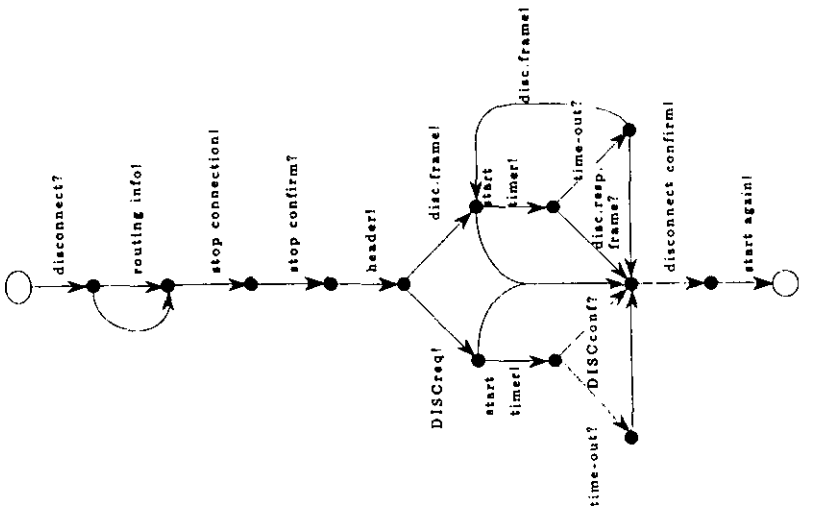

Fig 6.5.e Behaviour after a disconnect command

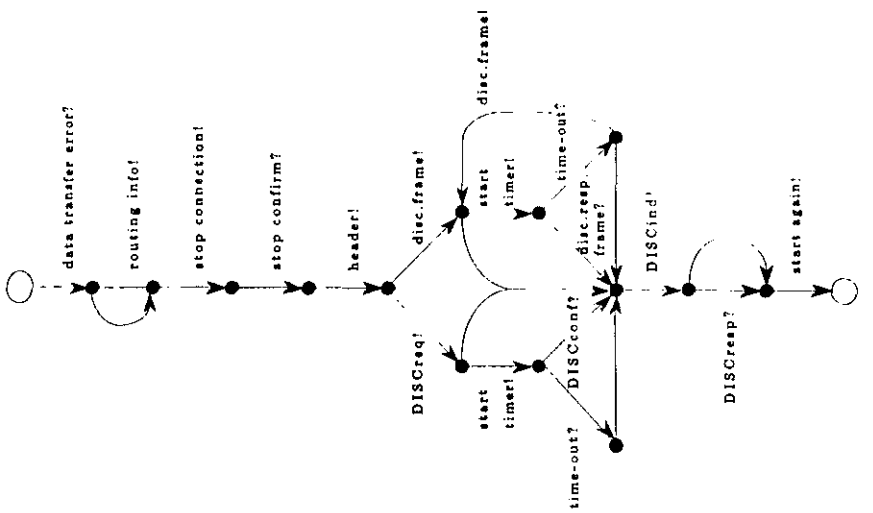

Fig. 6.5.f Behaviour after a data transfer error

6.3.4 Restart process

After an error, the communication can be reset either in one or two directions, depending on the error and on the protocol that is used. If one direction has to be reset, this can be in the reset invoking direction or in the opposite direction. Each option requires its own actions. We will therefore specify for each option, the traces of actions after a REST-request, -indication or arrival of reset frame. After an unrecoverable data-transfer error the entity has to decide in which direction it will restart the connection.

When the process is activated, the Connection Control process is notified of the reset by a signal indicating that transfer has to be stopped and indicating the data frames that have to be deleted. After the connection has been reset, a signal which indicates that transfer can be restarted has to be sent. In all states, the process stops after a signal which says the current connection is being released. The process will be preset to a state shown by a solid circle in the diagram. The process can be restarted by a "restart" signal.

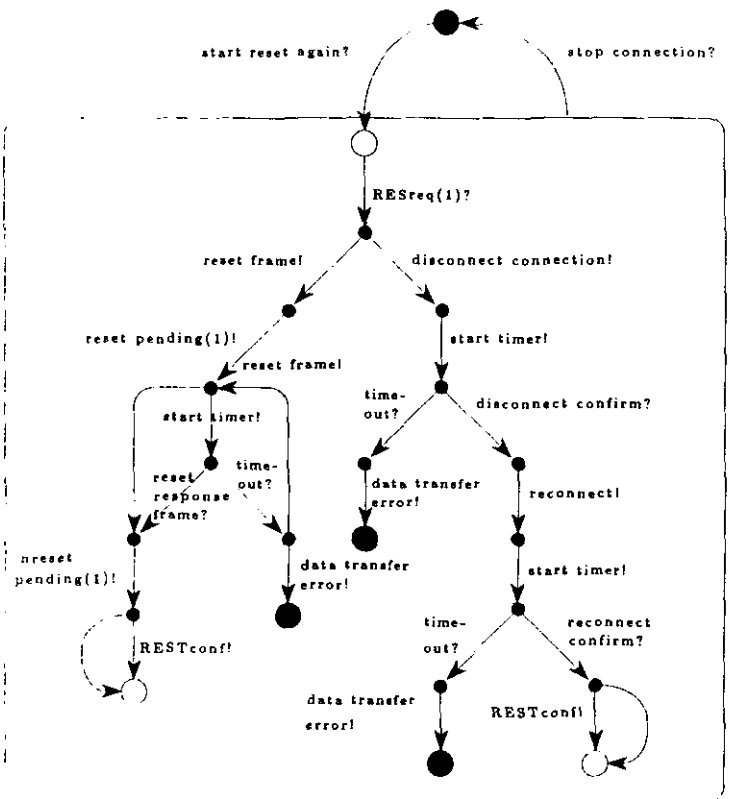

Fig. 6.6.a Behaviour of Restart process after a RESTreq

If the entity on top wants to reset the connection in both directions, it invokes a RESTreq with a parameter indicating that both directions have to be reset. In the next diagram this is indicated by a number 1. The connection is reset by either sending a reset frame, or by releasing the connection and after that re-establish it. All frames have to be deleted by the Connection Control process. This is forced by the "reset pending(1)"

signal, or by the "disconnect" signal. If only one direction need to be reset, the entity on top has to indicate this. In that case is the behaviour similar as above. Only control information within the primitives or frames has to indicate the direction of the reset.

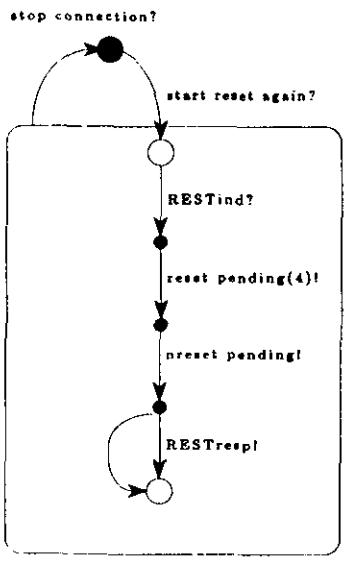

Fig 6.6.b Behaviour after a RESTind

If a RESTind arrives (figure 6.6.b), data transfer has to be restarted. This is noticed to the Connection Control process via a "reset pending" signal. After that, a RESTresp can be given if required.

The entity at the other side of the connection can force a reset by sending a reset frame. A control field within the frame has to indicate which direction will be reset. If the sending direction is reset, the entity on top has to be informed of the reset (figure 6.6.c, left). Otherwise, no notification has to be given to the entity on top (figure 6.6.c, right)

If the Data Transfer process notices an error which it cannot recover from, an "error" signal is given to the Restart process (if available, otherwise the signal will be given to the Disconnect process). The connection will be reset now. The behaviour of the process is the same as after a RESTreq. In stead of a RESTconf however, a RESTind will be given in some cases.

6.4 Data Transfer processes

6.4.1 Sequencing process

In order to guarantee sequenced delivery of data frames to the entity on top, sequence numbers must be added to the frames. The sequencing process controls assignment of values to these sequence numbers. For this purpose, four numbers have

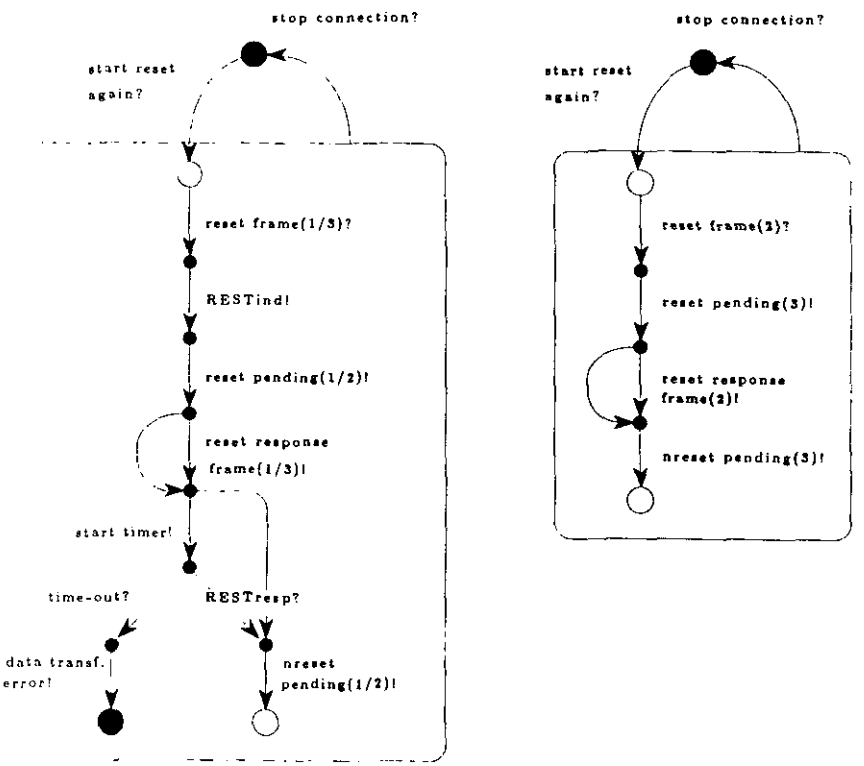

Fig 6.6.c Behaviour after a reset frame has been received

been defined :

ns indicates the number of frames already transmitted by this entity. (modulo the range of the numbers)

nr indicates the number of frames already received correctly by the entity.

nra indicates the number of received frames which are acknowledged to the other entity.

nsa indicates the number of frames which are acknowledged by the other entity.

Before the process can be started, above values have to be initialised (see figure 6.7.a). Also, window sizes and range of sequence numbers have to be initialised. After a "start" signal, the process enters the state with values (ns,nr,nsa,nra). A "stop" signal resets the process, while the current values of ns,nr,nsa and nra will be sent to the Connection Control process. If no data has to be transmitted, and frames have been received which have not been acknowledged yet, a timer will be started. This timer is only started once in each state. When this timer expires, a control frame has to be sent in order to acknowledge the received frames.

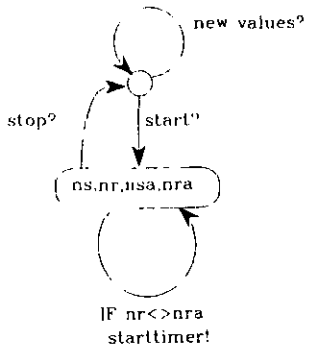

Fig. 6.7.a Initial behaviour of Sequencing process

If a frame has to be transmitted, the Blocking / Segmenting process requests a new sequence number for this frame (figure 6.7.b). Only if the window is not full ((ns-nsa) $\leq$window), this number can be generated and, together with the number of frames which have to be acknowledged, it may be returned. The Sequencing process now has to update the internal variables ns and nra. The Blocking / Segmenting process has to send a "new" signal for every new number it requires.

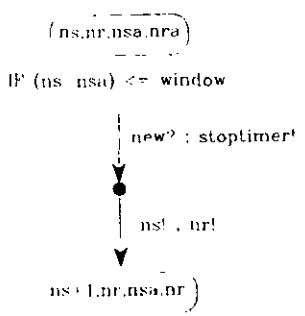

Fig 6.7.b New sequence numbers requested

If a frame is received correctly (figure 6.7.c), the number of frames that have to be acknowledged increases. Within the received frame, the value of NRA shows the number of frames received correctly by the other entity. This number updates the value of nsa. Also the value of nr has to be updated according to the value of NS in the frame. (i.e. the number of frames sent by the other entity, and received by this entity.) An error occurs if more frames are acknowledged by the other entity than this entity has sent. Also, if the received frame contains a number not expected, the frame has to be deleted and an error message has to be given. If the frame is a duplicate of a frame already received correctly, no serious error occurs. The frame can be deleted, and no other actions are required.

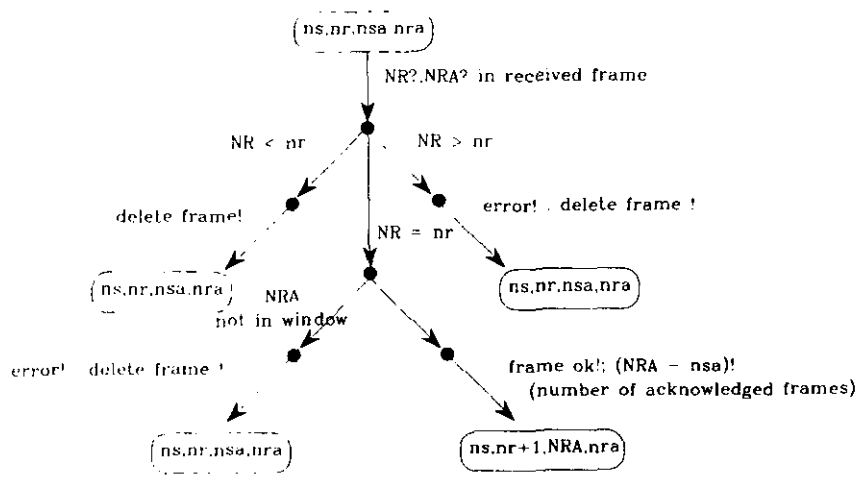

Fig 6.7.c Sequencing process after reception of a frame

If during a certain period, no frames have to be sent and a time-out arrives, the Data Transfer Control process has to send a control frame with an acknowledgement of received frames (figure 6.7.d, right).

If the Flow Control process notices that some frames have not been acknowledged within a certain period by the other entity, it will reset Sequencing process and Blocking / Segmenting process. The original frames cannot be resent with the original sequence numbers because the number of received frames can be changed. Therefore, also the Sequencing process has to be reset (figure 6.7.d, left).

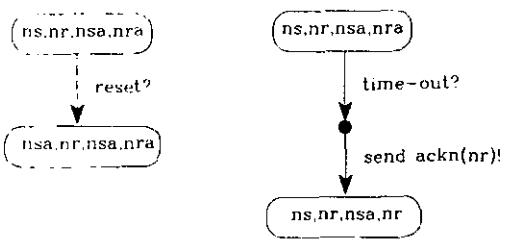

Fig 6.7.d Sequencing process after a reset or time-out

If a connection is reset, all frames with sequence number not equal to zero have to be deleted. This can be forced by the Deblocking / re-assembling process. Because correct frames may arrive immediately after a reset response, they have to be saved. If the reset procedure is finished, these frames can be transmitted to the entity on top.

6.4.2 Flow Control process

In section 5.4.2, we have described the behaviour of the Flow Control process. We have discussed that either each frame, or a (full) window can start a timer. In this section, we will describe the process more formally using one timer per frame. This provides us with the ability to implement selective rejection of frames later. Initially, the delay of the timers is set by Layer Operation. If congestion occurs, the delay will be increased by the Flow Control process to preserve further congestion. After acknowledgement of a number of frames, the delay can be decreased again, assuming congestion disappeared. Various methods exist to calculate the new value of delay.

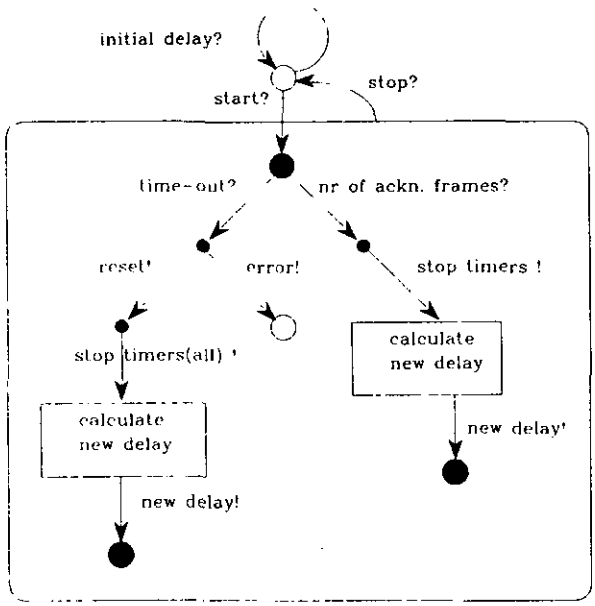

Fig. 6.8 Behaviour of Flow Control process

After initialisation, Layer Operation starts the process by a "start" signal. The process will be stopped by a "stop" signal. These signals do not have to contain information. The started process will be activated by either a time-out, or a signal containing the number of frames which have been acknowledged by the other entity. This latter signal require a higher priority than the time-out, because acknowledged frames do not have to be retransmitted and therefore, corresponding time-outs can be ignored. After reception of the acknowledgement, the corresponding timers are reset and new delay can be calculated and sent to the timers. If a timer expires, the process forces retransmission of all frames not yet acknowledged, starting with the frame causing the time-out. All timers can be reset and a new value for the delay can be calculated to preserve new congestion. If during a certain period too many time-outs occur, the process can stop, and send an error-signal to the Transfer Control process.

6.4.3 Data Transfer Control process

In chapter five, we have described this process informally. We noticed that the structure of this process is, for a great part, protocol dependent. All control information is transferred by this process (figure 6.9.a). During data transfer, this process stays active. If an error occurs in one of the Transfer processes, the process is informed of this error and must recover from it.

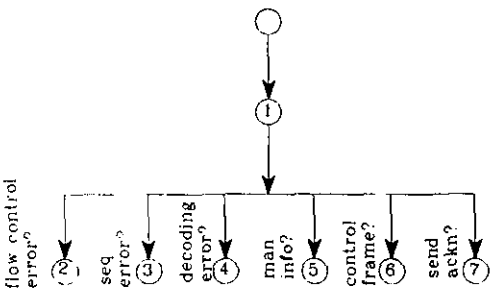

Fig. 6.9.a Initial state of Data Transfer Control process

Errors which can occur are :

a) flow control error
* A serious congestion noticed by Flow Control process.
b) sequencing error
* NR in a received frame to high.
* NRA in received frame not in window.
c) decoding error
* A damaged frame has been received

If management info is received by the process, a control frame will be generated and has to be sent to the other entity. A control frame must also be sent in case received frames have to be acknowledged. If a control frame has been received, the header indicates which Layer Operation process the information is meant for. The Data Transfer Control process will route the information to this process.

Other processes within the Data Transfer process do not notice errors. Errors which are reported to this process have to be stored for administration. Actions must be taken to recover from the error. A control frame may have to be sent. If that does not solve the problem, a signal is sent to Layer Operation to reset or release the connection (figure 6.9.b).

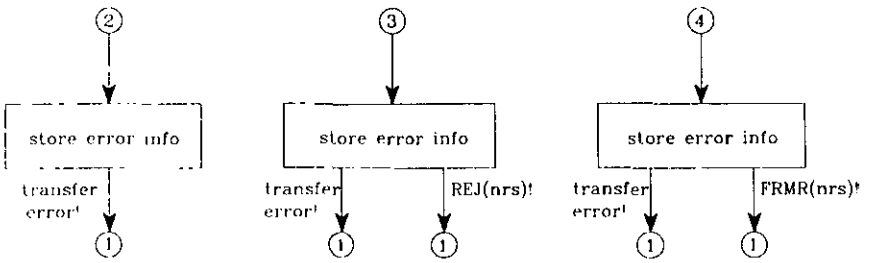

Fig 6.9.b Behaviour after an error notification

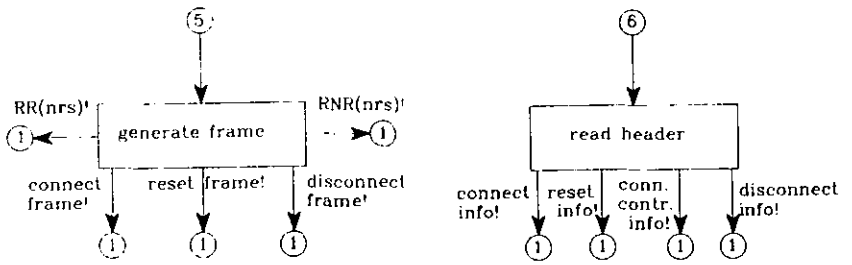

Fig. 6.9.c Transfer of control information by the process

6.4.4 Blocking/ segmenting

This process transforms SDUs into PDUs. For this purpose, it has to generate headers which contain sequence numbers and information about blocking or segmenting. Also, the header has to distinguish control frames from data frames. The process is described in figure 6.10.a..c. The process goes after activation to state 1. In this state, the process can be stopped by a "stop" signal. A DATAreq activates the process. If normal data transfer is used, sequence numbers will be requested, and a header is generated. After that, the PDU is transferred to the coding process. If blocking is used, the process goes after a DATAreq from state 1 to state 2, if segmenting is used, to state 3 (figure 6.10.a).

In case of blocking (figure 6.10.b), a timer will be started after the first DATAreq in order to control the transfer delay. DATAreqs will be accepted and combined into one PDU, until this PDU is full. After that, and when the timer expires, the process goes to the state in figure 6.10.a indicated by the solid circle.

If an SDU is subdivided over several PDUs (figure 6.10.c), a corresponding number of Block Descriptors has to be used. After one segment has been transferred, the process has to calculate the rest of data to execute, and may return to state 3.

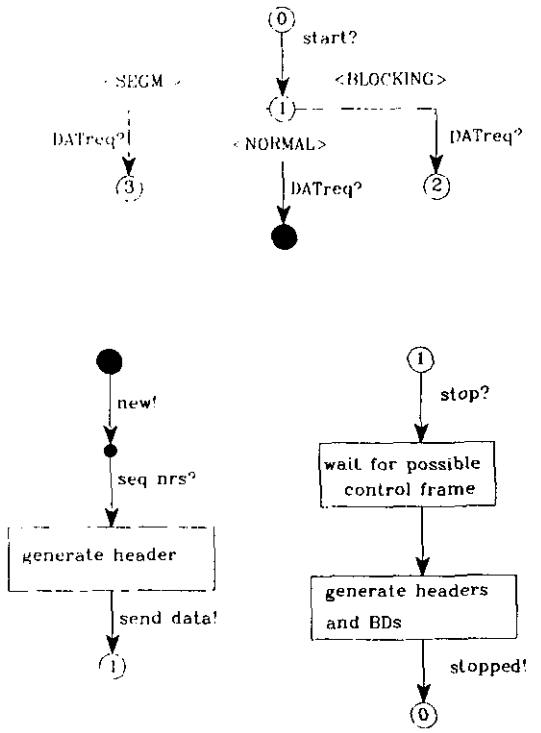

Fig. 6.10.a Actions after Arrival of a DATAreq

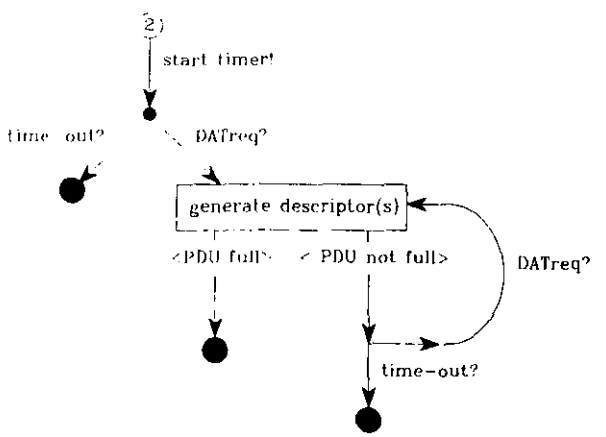

Fig. 6.10.b Blocking

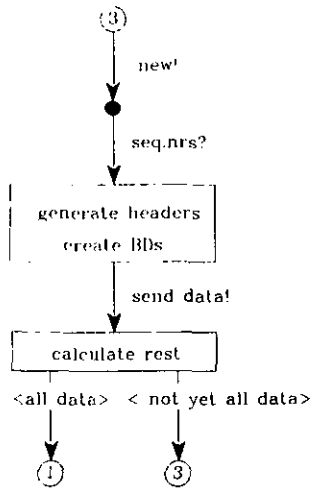

Fig. 6.10.c Segmenting

6.4.5 Deblocking / re-assembling

This process shows a "mirrored" behaviour of the Blocking process (figure 6.11.a). If data has been received, and did not contain errors, the process will read the header(s). Included sequence numbers will be sent to the Sequencing process which indicates whether the frame is "in sequence", or not. In case of no sequencing errors, and if only normal data transfer is used, a DATAind will be invoked to the entity on top.

If SDUs have been blocked by the other entity, more DATAinds have to be invoked (see figure 6.11.b, De-blocking). For each DATAind, a Block Descriptor (BD) has to be generated. If an SDU has been segmented by the other entity, the process has to wait for more data if not all segments have been received yet. This is shown in figure 6.11.b (Re-assembling) by the un-named arrow to state 1. Received segments will be linked together by the BDs, and a DATAind can be invoked after the last segment has been received.

In case the connection is reset, this process still has to stay active. However, the Sequencing process must give a negative acknowledgement for all frames which have to be deleted. The process may only transfer SDUs to the entity on top which have been received after the reset. If the header in a frame indicates that the frame contains control info, the frame (BD) is transferred to the Transfer Control process. This process can route the frame to the right process in Layer Operation if necessary. Control frames do not contain sequence numbers, so they are always transferred to the Control process.

6.4.6 Coding process

All bit oriented functions will be performed by this process. For this goal, "uncoded" data will be read and, according to the code, transformed into "coded" data. Ciphering functions may be performed or a FCS can be generated and added to the frame. Also flags may be added or bit stuffing may be performed. In case of coding, the

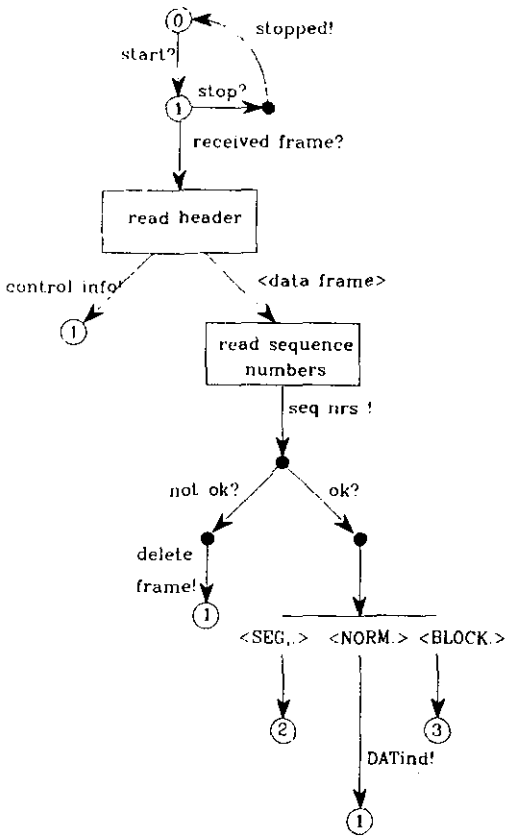

Fig. 6.11.a Behaviour after reception of a frame

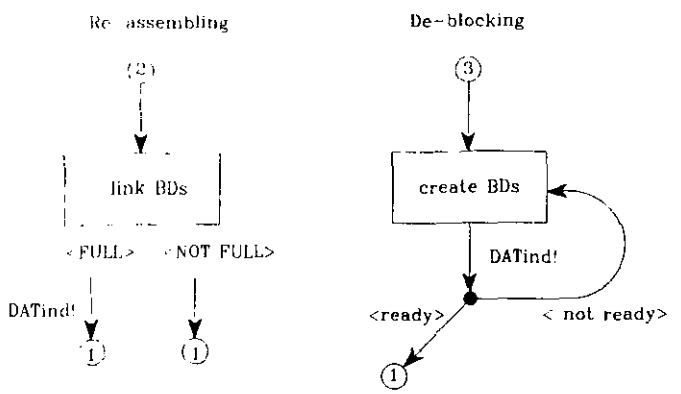

Fig. 6.11.b De-blocking and re-assembling

coded data has to be stored as well as the uncoded data. Coded data has to be stored into "new" memory, therefore. Via Memory Management, access to this new memory can be obtained.

If all data has been coded, a DATAreq will be invoked by the process and a new frame can be accepted from the Blocking / segmenting process. The DATAreq to the lower layer will start a timer used for flow control. If the process is stopped, all data can be deleted.

6.4.7 Decoding process

This process is also required only at some layers only. A DATAind activates the process. The "coded" data will be read and decoded. If an error occurs the data has to be deleted and an "error signal" will be generated. If the process is stopped, decoding must be finished, and the frame must be sent to the Deblocking process to be sure it will be delivered to the entity on top. After that, the process may acknowledge the "stop signal".

7 MULTIPLEXING / SPLITTING

In section 4.5, we have described the Multiplexing / splitting entity informally. In this chapter, we will give a more detailed description of this entity. The entity has to attache multiple SAPs to one communication entity, or multiple entities to one SAP.Multiplexing or splitting requires some extra functions to be performed, e.g. re-sequencing, memory management. Some of these functions can be executed by the M/S entity, while others have to be performed by processes within the Communication entities. The M/S entity, and extensions to these processes will be discussed in the following sections.

7.1 Multiplexing

If two entities within a layer are using one underlying communication entity, they both have to be attached to this entity via an access point. For this attachment, a multiplexer will be used. In the upstream direction, de-multiplexing functions have to be performed. All primitives of both entities on top have to be transferred to the entity beneath, and vice versa. An arbiter has to decide which entity on top will get access. In order to avoid deadlock, a SAP has to be subdivided into two parts, one for each direction of primitive transfer. As nearly all control primitives require an acknowledgement, no deadlock can occur within the SAP. Some primitives invoked by the entity beneath have to be transferred to both entities on top. In that case, the underlying entity has to send special information to the control unit.

If two connections are multiplexed onto one connection, data transfer has to be multiplexed by the underlying entity. For this purpose, the Connection Control process within this lower layer has to change the window size. Extra information must be added to the DATAreqs in order to separate PDUs. The Multiplexing unit will add this information to DATAreqs, and extract information from DATAinds in order to address these primitives. If the underlying connection is reset, both entities on top must be informed of this reset. This has to be controlled by the M/S entity. If either one of the upper layer entities stops communication, the underlying connections can not be released. Only the window size in the Sequencing process has to be changed. An error, resulting in release of the underlying connection, has to be informed to both the upper layer entities.

Because we modelled the various processes within an entity as parallel processes, the most efficient way to implement a SAP is to provide each primitive type with an implementation described in fig 2.8. In that case, a SAP will consist of four blocks. For the connect-, disconnect-, reset-, and data-primitives each one block. The control unit that executes the multiplexing function, therefore also consists of four, nearly the same, blocks. One such a block within the multiplexing unit consist of two, nearly similar sub-blocks. One sub-block for primitives downstream, and one block for primitives upstream (fig 7.1). Also a set of multiplexers and demultiplexers has to be added to each block. Two communication entities on top will be attached to the control unit, which is, via attachment control of the lower layer, attached to one communication entity within the lower layer. In the next sections, we will describe the control units for each direction of primitive transfer separately.

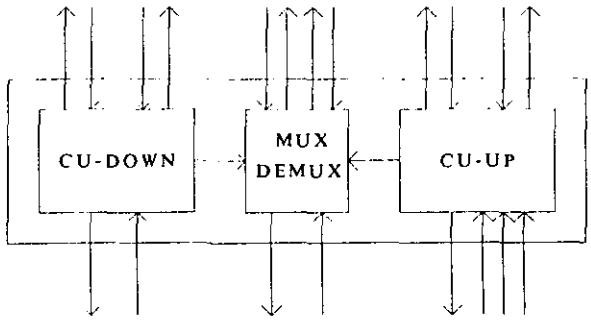

Fig. 7.1 Control unit of a single primitive

7.1.1 Control "down"

In figure 7.2, the control unit that controls the primitives downstream is shown. This unit is similar for each type of primitive. Only for data primitives, the unit has to add information to the primitives in order to provide the other side of the connection with the ability to separate the primitives. This extra function is not shown in the figure and in the state diagram of the control unit (fig 7.3). The state diagram has been extracted from the description of the control unit given in chapter 2.

In case of conflicts, the control unit uses a priority scheme to decide which entity will get access. Initially, entity 0 will get access. After a collision, priority will be interchanged. Naming of the entities will not be described in detail. Permanent naming can be used, and in case dynamic attachment to this multiplexing entity is required, an extra naming process must be implemented.

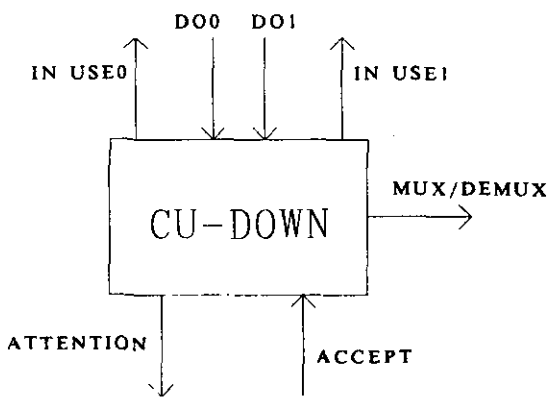

Fig. 7.2 Control unit for primitives downstream

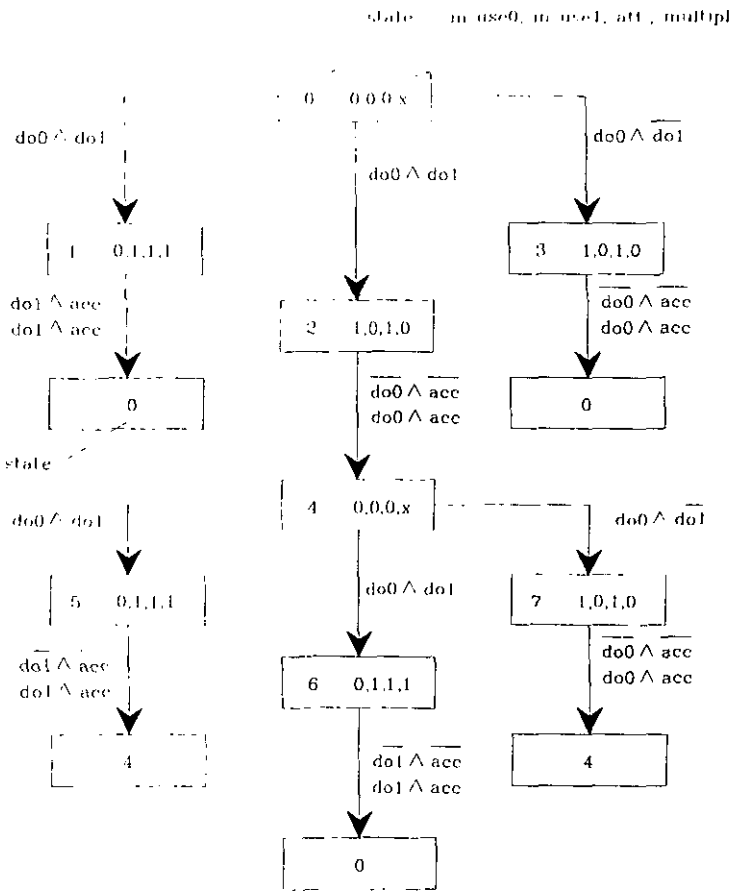

Fig. 7.3 State diagram of control unit "down"

7.1.2 Control "up"

The control unit for primitives in the upward direction is shown in figure 7.4. The CEPI input signal has to indicate to which entity the primitive has to be transferred. This CEPI can be set by a process within the communication entity or, in case of data primitives, an extra process must be added which reads the information from the primitives indicating which connection has to be operated. Depending on this information, the CEPI will be set. The "all" signal indicates that all the entities on top have to be informed of the primitive. E.g. DISCind primitives may have to be transferred to both entities. The state diagram of the control unit is given in figure 7.5. In the diagram, the value of the CEPI is added to the "do" signal. If the "all" line is set, the CEPI input is not relevant. In the diagram, this is also shown within the "do" signal.

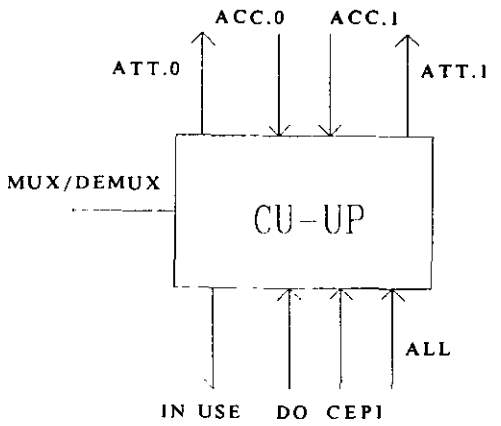

Fig. 7.4 Control unit for primitives upstream

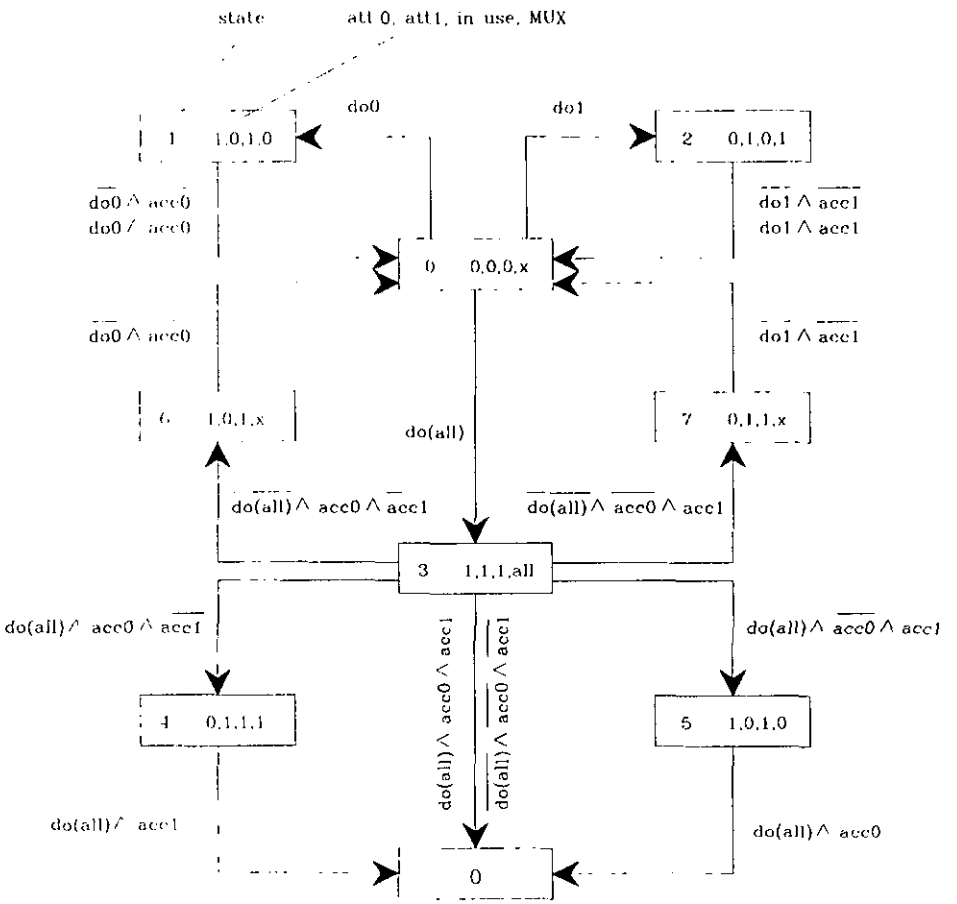

Fig. 7.5 State diagram of control unit "up"

7.2 Splitting

In section 4.5, we have discussed a splitting entity that performed re-sequencing functions for the PDUs. In this section, we will show that we can use a control unit, nearly similar as used for multiplexing, which attaches the entity to multiple SAPs. This unit will route the primitives to the right SAPs. Re-sequencing functions will not be performed by this unit. They will have to be performed by the communication entity. Process extensions necessary for this re-sequencing will be described as well.

Connection splitting can be seen as demultiplexing. Therefore, the control unit of the splitting entity can be seen as the mirrored of the control unit of the multiplexing entity. The "up" and "down" unit have to be interchanged.Some primitives, e.g. DATAreqs, have to be transferred to one of the SAPs, careless which one. For this purpose, the control unit has to be extended.

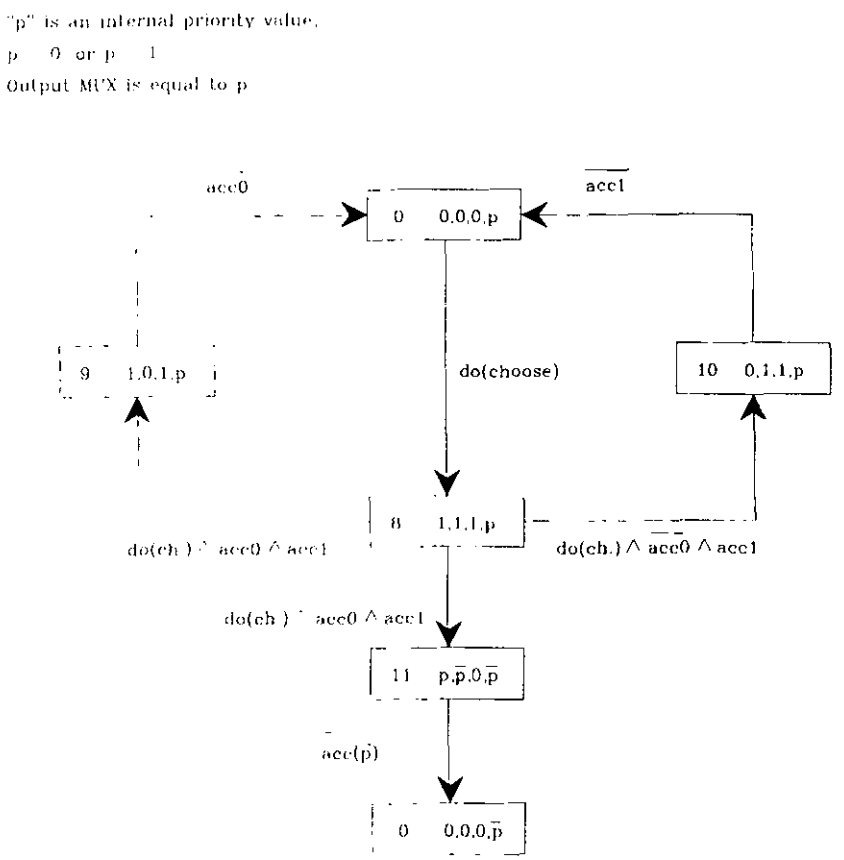

Fig. 7.6 State diagram of control unit "down" of splitting entity

An extra signal from the entity on top will be used to indicate that the primitive has to be transferred to either one of the SAPs. After this kind of request, the control unit will invoke a request on all SAPs. The first SAP accepting the request will be granted. The requests on the other SAP will be cancelled. If both SAPs accept the request at the same time, one acceptance will be granted using a priority scheme, the

other will be cancelled. If the entity on top cancels a request while it is accepted at the same time, the "attention" line will be kept high. The lower layer entity now knows that the request has been cancelled.

The architecture of the control unit "up" of the splitting entity is similar to the architecture of the multiplexing control unit "down". The "do"- and "in use"-lines will be attached to the entities beneath, and the "attention"-, "accept"-line to the entity on top. The state diagram is equal to that of figure 7.3.

The splitting control unit "down" is nearly equal to the multiplexing control unit "up". Instead of an "all" signal only, also a "choose" signal will be required. If this signal is set, the control unit may select one of the SAPs by itself. In case of a "do" command with the "choose" signal set, the state transitions of the control unit are described in figure 7.5. The rest of the state diagram is equal to the diagram in figure 7.6.

7.3 Extension to Data Transfer processes in order to provide splitting

If we use a splitting entity guarantees sequenced delivery of PDUs to the communication entity, this entity requires extra buffers, a decoding process and a sequence number comparator. A more efficient solution will be to transfer the PDUs to the communication entity without re-sequencing. In that case, the splitting entity controls access of the SAPs only, and the communication entity has to resolve from sequence errors. Some Data Transfer processes have to be extended in order to guarantee sequenced delivery of SDUs to the entity on top. The Sequencing process has to accept all frames containing sequence numbers within the window, the Deblocking / re-assembling process has to store all frames accepted, and has to transfer SDUs in the right order to the entity on top. All other processes do not have to be changed.

7.3.1 Extensions to Sequencing process

The sequencing process described in section 6.4.1 did accept frames with sequence number equal to nr only. All other frames had to be rejected. In case of re-sequencing within the Communication entity, all frames containing a sequence number within the current window have to be accepted. They have to be reordered in order to perform sequencing. After reception of a frame the process will read sequence numbers NR and NRA, and will act as described in figure 7.7. If NR or NRA (or both) are not within the window, an error has occurred. The frame has to be deleted and an error signal may be generated. Otherwise, the frame will be accepted, the value of NR modulus the window size will be transferred to the Deblocking / re-assembling process for reordering. The value of NRA-nsa indicates how many frames have been acknowledged by the peer entity. This value will be sent to the Flow control process such that the corresponding timer(s) can be stopped. The value of nr can not be updated similar to the way we described in section 6.4.1. In that case, without splitting and re-sequencing, the Deblocking process did only transfer the sequence numbers to the sequencing process if one of its buffers was free. Consequently, the sequencing process could increment the value of nr because the free buffer could be re-used immediately. If the Sequencing process accepts a frame, this frame must be stored, since the process will update parameters according to the values within this frame. However, if the Deblocking process has not yet freed the corresponding buffer, this frame can not be stored and therefore, may not be accepted. In order to avoid these conflicts, the Deblocking process has to

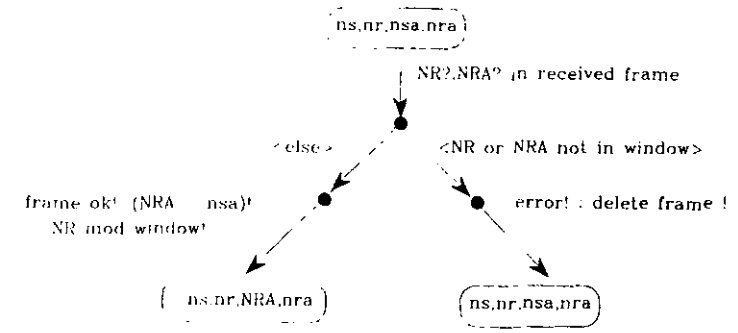

Fig. 7.7 Extended Sequencing process after reception of a frame

inform the Sequencing process of the number of buffers that can be re-used after a DATAind. Corresponding to this number, the Sequencing process can update its parameters.

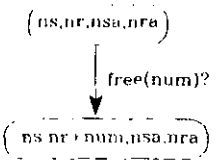

Fig. 7.8 Sequencing process updating the window

7.3.2 Extensions to Deblocking process

The Deblocking process has to store and transfer PDUs in a sequenced order to the entity on top. For this purpose, a set of registers is required equal to the window size. If a PDU has been accepted by the Sequencing process, the value of NR (MOD the window size) returned to the Deblocking process, indicates in which register the pointer to this PDU has to be stored. A "valid, invalid" bit added to each register, has to indicate whether the corresponding register contains a pointer or not.(see figure 7.9) If a register does contain a pointer already,the received PDU has to be a duplicate and may be deleted. To the set of registers, a memory element has to be appended that contains a pointer to the register which contents has to be transferred next. After transfer of this contents, the "freed number of buffers" value will be incremented. If a DATAind has been invoked by the process, the value of the register "freed number" will be transferred to the Sequencing process so the window can be updated. The corresponding registers can be re-used now. If multiple SDUs have been blocked into one PDU, multiple DATAinds will be invoked but only one register will be freed. If one SDU has been segmented over multiple PDUs, only one DATAind will be invoked, causing several

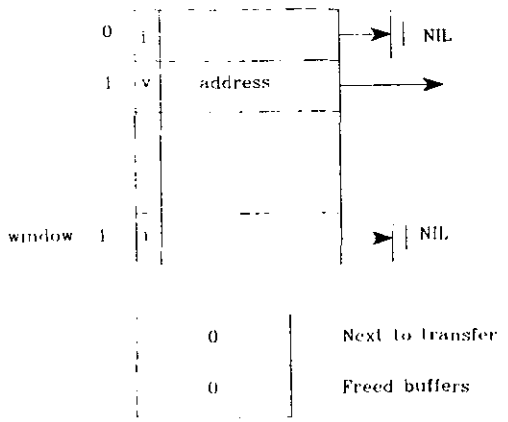

Fig. 7.9 Registers for pointers to PDUs

registers to be cleared. In figure 7.9, a PDU with sequence number 1 (MOD the window size) has been received correctly. However, the next frame to transfer has to be stored in register 0. Only after a PDU with sequence number 0 (MOD the window size) has been received and transferred, register 1 can be transferred and freed.

In this section, we discussed only an implementation of the re-ordering function of the Deblocking process. Additional, the process has to be able to receive frames from the Decoding process, read sequence numbers from these frames, and delete frames if necessary. Also, the process has to be able to link buffer descriptors or generate buffer descriptors in order to perform re-assembling and deblocking. These functions have been described in the previous chapters of this paper and we will not discuss them in this section therefore. With the processes discussed in the previous chapters, extended with the functions and entities discussed in this chapter, the Subsystem can multiplex two connections onto one connection, and split one connection into two. Multiplexing of more than two connections, or splitting to more than two connections will not change the processes essentially. Priority schemes will be more complex however, as well as naming and administration.

8 MANAGEMENT

Each communication entity within the architectural model is an independent "data processor". Therefore, a model which contains several communication entities results in a multi-processor architecture. This architecture requires complex control functions in order to provide specified services to the subsystem on top of it. The fact that we deal with special communication entities in stead of general purpose processors is not relevant to these control functions. Connections have to be controlled similar to processes on a CPU. In a general multi-processor environment, this control is performed by an operating system. Other operating systems functions are event management, synchronisation and protection. These functions have to be performed in our model too. Since memory management and I/O management are independent of the functions executed by the processor, we can also use general operating systems techniques for these problems. Thus, we may conclude that control of the communication entities can be compared to a distributed operating system. In the next section, we will hierarchically describe the functions to be performed by Layer Management and Layer Operation in relation to operating systems.

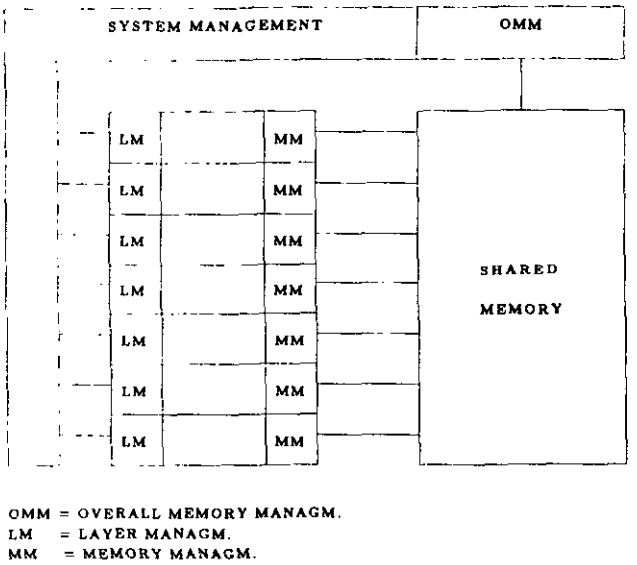

Fig. 8.1 System Management and Layer Management

8.1 Layer Management

In the ISO/OSI Reference Model, a communication system consists of seven subsystems. Systems control is performed by System Management. The most important tasks for System Management are initialisation or modification of subsystems, error control and overall memory control. (see figure 8.1) At each layer, overall control is performed by Layer Management in order to provide the specified services. For this

purpose, Layer Management can be subdivided into three main tasks which are Resource Management, Memory Management and I/O Management. In the next sections, we will discuss each of these tasks.

8.1.1 Resource Management

Resource Management allocates connections to communication entities, controls entities and performs access to entities. If a new connection has to be established, Resource Management starts a communication entity, initializes it and attaches it to an access point via I/O Management. In order to operate multiple connections by one entity, this entity has to be modified and must be attached to other SAPs if a new connection is added. Resource Management also has to resolve from errors occurring during establishment. Resource Management is an important task of Layer Management and various techniques used in operating systems can be used here too. For a discussion of these techniques we refer to literature.

8.1.2 Memory Management

The most important aspect of communication is the transfer of data. At the highest level of a communication system, lots of data may be generated for which transmission is requested. Every underlying layer adds its own information to this data and transfers the result to the lower layer. In most cases, the data as such is not transformed and therefore does not have to be transferred itself. A pointer to this data in shared memory can be used for the transfer between the layers. Architectures using this method have been discussed in several reports ([14], [15]). If we use shared memory, special care must be given in order to be sure that all entities within all subsystems are capable to access memory individually and independent of each other. When an entity has transferred a data unit to the lower layer, it has to be able to execute a new data unit while the lower layers are still executing the first unit. For this purpose, memory has to be subdivided into fixed sized blocks. Data within these blocks can be addressed via so called block descriptors (see figure 8.2). C.f. paging and page tables. Data does not have to start at the beginning of the block and does not have to fill the complete block. Special registers are included in the block descriptor for pointers and values for this purpose. If data does not fit into one block, multiple blocks can be filled and linked.

The data that will be transmitted finally must not contain block descriptors. Therefore, a control unit at the lowest layer has to remove these descriptors from the data stream. As a consequence, received data does not contain pointers. The control unit has to store this data into blocks and link the corresponding descriptors.

Since we use shared memory, special related control functions have to be performed by management. These functions are independent of the tasks performed by the processor. Therefore, multi-processor operating systems memory management functions can be used here. Some of the main functions will be discussed in the following sub-sections.

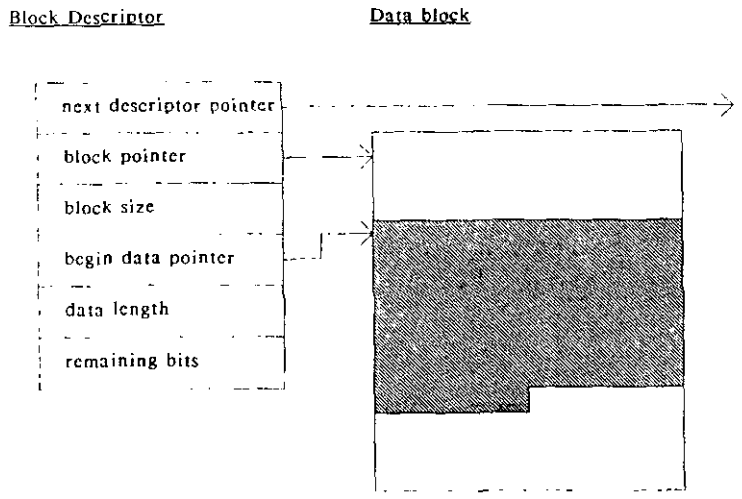

Fig. 8.2 Block descriptors and blocks

8.1.2.a Memory allocation

In order to provide parallel processing of data units, memory has been subdivided into blocks. These blocks have to be allocated to entities before they can be used. If all PDU sizes and window sizes within a subsystem are fixed, the number of blocks required can be calculated and allocated a priori by an overall memory manager (see figure 8.1). In order to store received data, also a fixed part of memory can be reserved. The global memory controller has to subdivide memory equally over the subsystems.This memory part cannot be allocated to the entities within the subsystems however. The memory blocks allocated to a subsystem are controlled by the local memory manager. This manager performs allocation of blocks to entities within the subsystem. Once blocks are allocated to an entity, they are owned by that entity, and cannot be used by another entity unless ownership has been transferred.

8.1.2.b Memory sharing

In order to transfer data between adjacent layers, we use pointer to the data in shared memory. Only these pointers will be transferred. Together with the pointer, ownership of the corresponding blocks is transferred. After that, an entity is not allowed to write into that blocks any more. Ownership is returned to the entity after the data has been transmitted or after a reset has been given to the lower layer subsystems. Memory Management controls this ownership.

At higher layers within the communication system, service primitive parameter negotiation may be required. For this purpose also shared memory can be used. Only two entities can have access to that memory part alternately. Both entities have to read and write to memory. Control of this access has been discussed in section 2.8.2.

8.1.2.c Memory protection and address mapping

These two functions may be required but they do not differ from functions used in general operating systems. Therefore, we will not discuss them in detail and refer to literature [19], [20], [21].

8.1.3 I/O Management

Input and output of a subsystem is performed via service access points. Attachment, release and naming of these access points has to be controlled by I/O management. Entities have to be attached to physical access points. These access points may have to be shared in case of multiplexing. This will also require special control. After an error, it may be necessary to detach the entity from the SAP in order to ensure re-use of these SAPs. All I/O control functions will be performed by the SAP control units, multiplexing and splitting units, and one overall I/O control unit.

8.2 Layer Operation

Layer Operation is the local communication entity's control process. It performs all local management functions which are Connection Management, I/O Management, Memory Management, Event Management, Synchronisation and Protection (see figure 8.3)

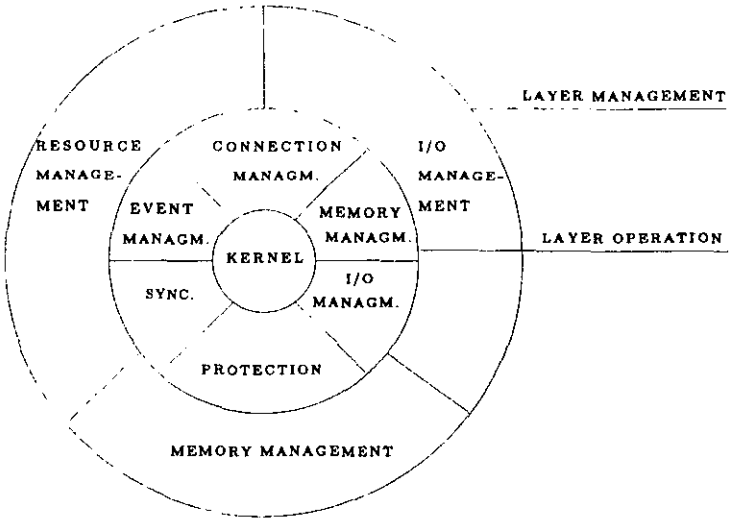

Fig. 8.3 Hierarchy of management functions

8.2.1 Connection Management

As we mentioned before, process control and connection control require similar functions. Therefore, we will discuss connection management in relation to process management of operating systems. Comparable to processes on a processor, connections have to be created, initialised, scheduled and released. During connection existence, errors have to be notified and resolved from.

A connection is started by creation of a connection control block (CCB, c.f. process control block). This CCB contains information about connection identification, type, priority and resource requirements. If a connection is suspended, the current state of the connection and the address where data has been stored have to be saved in the CCB. When the connection is restarted, this information is required to continue the protocol of that connection.

Similar to processes, connections follow the sequences of states given in figure 8.4. After creation and initialisation of a connection, this connection is ready to run. If no other connections are operated by the communication entity at the same time, transfer can be started immediately. For this purpose, the connection has to be scheduled and dispatched on the communication entity. If data transfer terminates normally, the connection can be released and the CCB can be deleted. If an error occurs, or during a reset, the connection can be suspended in the waiting state. After the entity has resolved from the error, the connection goes to the ready to run state. If the connection is ready to run, an external event can cause suspension too. Then, the connection is also transferred to the waiting state, until another event or action enables data transfer again.

Similar scheduling techniques as used in operating systems can be used for connection scheduling, e.g. FIFO, LIFO, PRIORITY, etc.. Especially for file transfer, time sliced scheduling techniques can be used. For real time communication, event driven scheduling has to be used.

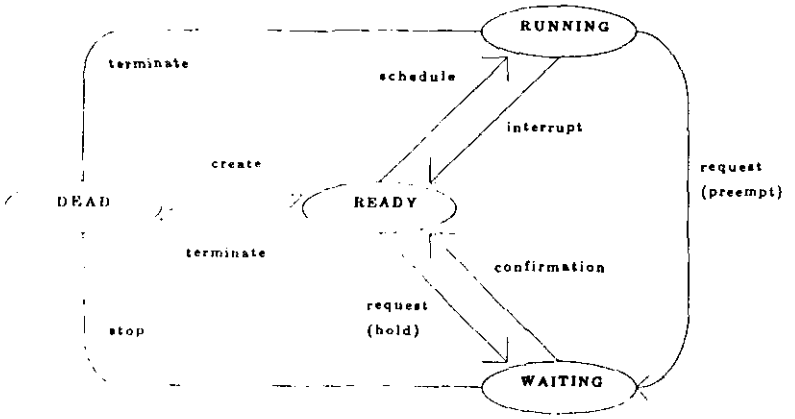

Fig. 8.4 State transitions of a connection

8.2.2 I/O Management and Memory Management

Layer Operations I/O- and Memory-management functions are similar to the related Layer Management functions. If an entity "owns" a number of blocks, control of these blocks is performed by a local Layer Operation process. I/O control will be performed by decentralised SAP control units (see chapter 2). Attachment to SAPs and switching will be controlled by Layer Management only, and not by Layer Operation.

8.2.3 Event Management and synchronisation

Event Management controls the processes and resources after special events. Events must be detected and their processing must be sequenced so those of highest priority are performed first. Therefore, Event Management is closely related to synchronisation of processes. Some processes may have to be halted or reset, e.g. re-sequencing, while other processes must be started. For example, Memory Management must be started after a reset in order to return blocks to the entity on top. Event Management tasks and synchronisation tasks are performed all over the Communication Entity, within Layer Operation as well as within the control processes within the Data Transfer process.

8.2.4 Protection

Memory protection is an important task of Memory Management in order to avoid access conflicts. This task however is not typically related to communication systems. Therefore, solutions used in general processor environments can be used here. We will not discuss these solutions in this report.

9 CONCLUSIONS

Because various, existing protocols for data communications show much similarities, in this report we have made a start to describe and implement a universal protocol. This universal protocol has to provide a maximum of functionality. The services and functions of this universal protocol have been used as a starting point for the Universal Protocol Subsystem architecture. In this report, a process model of the Universal Protocol Subsystem has been given, together with the behaviour descriptions of the processes within this model. Although some of the processes can be implemented directly, most of the processes are still described at such a high level of abstraction that direct implementation is not possible. Before that, further refinement and decomposition steps have to be taken. The high level process model can be used as a framework for implementation of almost every data communication protocol. Major decomposition decisions have been made already to ease this implementation. Also, all communications between the sub-processes that will be required has been modeled. This report can thus be used as a start point for protocol implementation. Also, further implementation of the universal protocol will be possible, starting with the presented model. This implementation may result in a system on which a number of different protocols can be executed only by changing parameters. However, before this will be realised, a lot of research has to be done.

REFERENCES

[1] Standard ISO 7498-1984. Information processing systems - Open systems interconnection - Basic reference model. Geneva: International Organization for Standardization, 1984.

[2] Zimmermann, H.
OSI reference model: The OSI model of architecture for open systems interconnection.
IEEE Trans. Commun., Vol. COM-28(1980), p. 425-432.

[3] CCITT Blue Book, Vol. VIII, Fascicle VIII.4: Data communication networks: Open Systems Interconnection (OSI) - Model and notation, service definition. Recommendations X.213 and X.214.
9th Plenary Assembly CCITT, Melbourne, 14-25 Nov. 1988.
Geneva: Int. Telecommunication Union, 1989.

[4] Tanenbaum, A.S.
Computer networks. 2nd ed.
Englewood Cliffs, N.J.: Prentice-Hall, 1988.

[5] Meijer, A. and P. Peeters
Computer network architecture.
London: Pitman, 1982.
Electrical engineering, communications,and signal processing, Vol. 1.

[6] Deasington, R.J.
X.25 explained: Protocols for packet switching networks.
Chichester: Ellis Horwood, 1985.
Ellis Horwood series in computers and their applications.

[7] Vissers, C.A. and L. Logrippo
The importance of the service concept in the design of data communication protocols.
In: Protocol specification, testing, and verification. Proc. 5th IFIP WG 6.1 Int. workshop, Toulouse-Moissac, 10-13 June 1985.
Ed. by M. Diaz.
Amsterdam: North-Holland, 1986. P. 3-17.

[8] Shields, M.W. and M.J. Wray
A CCS specification of the OSI network service.
Internal report. University of Edinburgh, 1983.
CSR-136-83

[9] Koomen, C.J.
Course notes System Technology 1. 2nd ed.
Digital Systems Group, Faculty of Electrical Engineering, Eindhoven University of Technology, 1988.

[10] Hoare, C.A.R.
Communicating sequential processes.
Englewood Cliffs, N.J.: Prentice-Hall, 1985.
Prentice-Hall international series in computer science

[11] Ward, P.T. and S.J. Mellor
Structured development for real-time systems. Vol. 1: Introduction and tools.
Englewood Cliffs, N.J.: Yourdon Press/Prentice-Hall, 1985.

[12] Hatley, D.J. and I.A. Pirbhai
Strategies for real-time system specification.
New York: Dorset House, 1987.

[13] Daanen, J.M.V.
Design of an X.25 co-processor.
M.Sc. thesis. Digital Systems Group, Faculty of Electrical Engineering, Eindhoven University of Technology, 1987.

[14] Nissink, P.L.H.M.
Design of an ISDN co-processor.
M.Sc. thesis. Ibid., 1987.

[15] Vos, H.J.M.
Interprocess communication in a multiprocessor environment: Specification of a communication controller.
M.Sc. thesis. Ibid., 1987

[16] Klip, A.
X.25 co-processor functional design level 1 and 2.
M.Sc. thesis. Ibid., 1985.

[17] Milner, R.
A calculus for communicating systems.
Berlin: Springer, 1980.
Lecture notes in computer science, Vol. 92.

[18] CCITT Blue Book, Vol. X, Fascicles X.1, X.2, X.3, X.4 and X.5: Functional Specification and Description Language (SDL). Criteria for using Formal Description Techniques (FDT's). Recommendation Z.100 and Annexes A, B, C, D, E, F.1, F.2 and F.3.
9th Plenary Assembly CCITT, Melbourne, 14-25 Nov. 1988.
Geneva: Int. Telecommunication Union, 1989.

[19] Silberschatz, A. and J.L. Peterson
Operating systems concepts.
Reading, Mass.: Addison-Wesley, 1983.
Addison-Wesley series in computer science

[20] Tanenbaum, A.S.
Operating systems: Design and implementation.
Englewood Cliffs, N.J.: Prentice-Hall, 1987.
Prentice-Hall software series

[21] Fortier, P.J.
Design of distributed operating systems.
New York: McGraw-Hill, 1986.

[22] Wesselius, R.F.
Configuration and name management.
M.Sc. thesis. Digital Systems Group, Faculty of Electrical Engineering, Eindhoven University of Technology, 1988.

GLOSSARY

BD : Block Descriptor.

CCB : Connection Control Block, c.f. Process Control Block.

CCITT : International Telegraph and Telephone Consultative Committee.

CCS : Calculus of Communicating Systems.

C.E. : Communication Entity.

CEP : Connection End-Point.

CEPI : Connection End-Point Identifier.

conf : confirmation.

CPU : Central Processor Unit.

CU : Control Unit.

EHKP4 : Einheitiges Höheres Kommunications Protokolle (Layer four).

FCS : Frame Check Sequence.

FIFO : First In First Out.

HDLC : High-level Data Link Control.

ind : indication.

ISO : International Standards Organization.

LAPB : Link Access Procedure B.

LIFO : Last In First Out.

LLC : Logical Link Control.

LM : Layer Management.

LO : Layer Operation.

M/S : Multiplexing / Splitting.

OS : Operating System.

OSI : Open Systems Interconnection.

PCI : Protocol Control Information.

PCM : Performance Configuration Management.

PDU : Protocol Data Unit.

QOS : Quality Of Service.

req : request.

resp : response.

SAP : Service Access Point.

SAPI : Service Access Point Identifier.

SDL : Specification and Description Language (CCITT).

SDLC : Synchronous Data Link Control.

SDU : Service Data Unit.

SNA : Systems Network Architecture.

UPS : Universal Protocol Subsystem.

APPENDIX A A CCS SPECIFICATION OF PRIMITIVE SEQUENCES.

In chapter 3 we have given a specification of the sequences of primitives across the interface between two layers. In this appendix we will consider two entities communicating with each other using one connection. If we model the primitive sequences in CCS, and also model a service provider as described in the CCITT recommendations [CCITT], we can calculate the combined behaviour of one entity with the provider. The resulting behaviour shows to be the mirrored behaviour of the other entity and thus, we may conclude that the sequences are defined well to be sure of "good" communication. This means that no deadlock will occur when these sequences are followed by the entities and the provider.

On the next pages, the CCS descriptions of the service user A and B will be given as well as the description of the service provider, according to the CCITT recommendation. We described a provider with a queue capacity of two data frames. A queue with more frames does not change the behaviour. We restricted ourselves to a capacity of two because a greater capacity could cause memory overflow in the program. The behaviour of the expanded process of an entity with the service provider is calculated. The program was not able to reduce this behaviour because of memory overflow. However in this behaviour we can see the mirrored behaviour of a service user.

SERVICE USERS

The behaviour of the service users A and B is described below. The sequences of primitives are as in figure 2.3.

Service user A

SA0	=	cra!:SA1	cr = connection request
	+	cia?:SA2	ci = connection indication
SA1	=	cca?:DA	cc = connection confirm
	+	dia?:SA0	crp = connection response
	+	dra!:SA0	
SA2	=	crpa!:DA	dr = disconnect request
	+	dia?:SA0	di = disconnect indication
	+	dra!:SA0	
DA	=	datra!:DA	datr = data request
	+	datia?:DA	dati = data indication
	+	rra!:RA0	
	+	ria?:RA1	rr = reset request
	+	dra!:SA0	ri = reset indication
	+	dia?:SA0	rc = reset confirm
RA0	=	rca?:DA	rrp = reset response
	+	dra!:SA0	
	+	dia?:SA0	
RA1	=	rrpa!:DA	The primitives are extended with
	+	dra!:SA0	the character a for user A and b
	+	dia?:SA0	for user B.

Service user B

```
SB0  =  crb!:SB1
     +  cib?:SB2
SB1  =  ccb?:DB
     +  dib?:SB0
     +  drb!:SB0
SB2  =  crpb!:DB
     +  dib?:SB0
     +  drb!:SB0
DB   =  datrb!:DB
     +  datib?:DB
     +  rrb!:RB0
     +  rib?:RB1
     +  drb!:SB0
     +  dib?:SB0
RB0  =  rcb?:DB
     +  drb!:SB0
     +  dib?:SB0
RB1  =  rrpb!:DB
     +  drb!:SB0
     +  dib?:SB0
```

SERVICE PROVIDER

The service provider can be seen as a pair of queues. One queue for each direction of the connection. The queues can be filled by the service users and by the service provider. The capacity of the queues is fixed.

The first seven equations below (I0,..,I6) describe the behaviour of the provider in the establishment phase. After a collision, no connection will be established. The provider can invoke a connection release by its own. The behaviour after such action is described by DC0.

```
I0   =  cra?:I1
     +  crb?:I2

I1   =  cib!:I3
     +  crb?:DC0
     +  dcra?:I0

I2   =  cia!:I4
     +  cra?:DC0
     +  dcrb?:I0

I3   =  crpb?:I5
     +  dcra?:DC1
     +  dcrb?:DC2
     +  tau:DC0
```

```
I4    =   crpa?:I6
      +   dcra?:DC1
      +   dcrb?:DC2
      +   tau:DC0

I5    =   cca!:Q0
      +   dcra?:DC1
      +   dcrb?:DC2
      +   tau:DC0

I6    =   ccb!:Q0
      +   dcra?:DC1
      +   dcrb?:DC2
      +   tau:DC0
```

After connection establishment, the provider acts like a queue for data objects. A data request at one side results in a data indication at the other side. After a reset request or a disconnect request, no more data will be accepted anymore at the corresponding side. The other side can receive data and send data up to the moment the reset object or disconnect object arrives. The behavior after these primitives is described by RQA, RQB, DQA, DQB. A provider initiated reset is described by R0 and a provider initiated disconnect is described by DC0.

```
Q0    =   datra?:Q1
      +   datrb?:Q2
      +   rra?:RQA0
      +   rrb?:RQB0
      +   dra?:DQA0
      +   drb?:DQB0
      +   tau:DC0
      +   tau:R0

Q1    =   datra?:Q3
      +   datib!:Q0
      +   datrb?:Q4
      +   rra?:RQA1
      +   rrb?:RQB1
      +   dra?:DQA1
      +   drb?:DQB1
      +   tau:DC0
      +   tau:R0

Q2    =   datra?:Q4
      +   datrb?:Q5
      +   datia!:Q0
      +   rra?:RQA2
      +   rrb?:RQB2
      +   dra?:DQA2
      +   drb?:DQB2
```

```
        +       tau:DC0
        +       tau:R0

Q3      =       datib!:Q1
        +       datrb?:Q6
        +       rra?:RQA3
        +       rrb?:RQB3
        +       dra?:DQA3
        +       drb?:DQB3
        +       tau:DC0
        +       tau:R0

Q4      =       datra?:Q6
        +       datib!:Q2
        +       datrb?:Q7
        +       datia!:Q1
        +       rra?:RQA4
        +       rrb?:RQB4
        +       dra?:DQA4
        +       drb?:DQB4
        +       tau:DC0
        +       tau:R0

Q5      =       datra?:Q7
        +       datia!:Q2
        +       rra?:RQA5
        +       rrb?:RQB5
        +       dra?:DQA5
        +       drb?:DQB5
        +       tau:DC0
        +       tau:R0

Q6      =       datib!:Q4
        +       datrb?:Q8
        +       datia!:Q3
        +       rra?:RQA6
        +       rrb?:RQB6
        +       dra?:DQA6
        +       drb?:DQB6
        +       tau:DC0
        +       tau:R0

Q7      =       datra?:Q8
        +       datib!:Q5
        +       datia!:Q4
        +       rra?:RQA7
        +       rrb?:RQB7
        +       dra?:DQA7
        +       drb?:DQB7
        +       tau:DC0
```

```
        +       tau:R0

Q8      =       datib!:Q7
        +       datia!:Q6
        +       rra?:RQA8
        +       rrb?:RQB8
        +       dra?:DQA8
        +       drb?:DQB8
        +       tau:DC0
        +       tau:R0
```

The behavior of the provider after a reset request of A is described below. Depending on the number of objects in the queue the provider will show one of the next behaviors :

```
RQA0 =  datrb?:RQA2 Behaviour after a reset request of A
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA0
        +       drb?:DC2
        +       tau:DC0

RQA1 =  datib!:RQA0
        +       datrb?:RQA4
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA1
        +       drb?:DC2
        +       tau:DC0

RQA2 =  datrb?:RQA5
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA2
        +       drb?:DC2
        +       tau:DC0

RQA3 =  datib!:RQA1
        +       datrb?:RQA6
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA3
        +       drb?:DC2
        +       tau:DC0

RQA4 =  datib!:RQA2
        +       datrb?:RQA7
        +       rib!:R10
        +       rrb?:R15
```

```
        +       dra?:DQA4
        +       drb?:DC2
        +       tau:DC0

RQA5 =  rib!:R10
        +       rrb?:R15
        +       dra?:DQA5
        +       drb?:DC2
        +       tau:DC0

RQA6 =  datib!:RQA4
        +       datrb?:RQA8
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA6
        +       drb?:DC2
        +       tau:DC0

RQA7 =  datib!:RQA5
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA7
        +       drb?:DC2
        +       tau:DC0

RQA8 =  datib!:RQA7
        +       rib!:R10
        +       rrb?:R15
        +       dra?:DQA8
        +       drb?:DC2
        +       tau:DC0
```

After a reset request of B, the provider behaves as in one of the next equations:

```
RQB0 =  datra?:RQB2
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB0
        +       dra?:DC1
        +       tau:DC0

RQB1 =  datia!:RQB0
        +       datra?:RQB4
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB1
        +       dra?:DC1
        +       tau:DC0

RQB2 =  datra?:RQB5
```

```
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB2
        +       dra?:DC1
        +       tau:DC0

RQB3 =  datia!:RQB1
        +       datra?:RQB6
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB3
        +       dra?:DC1
        +       tau:DC0

RQB4 =  datia!:RQB2
        +       datra?:RQB7
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB4
        +       dra?:DC1
        +       tau:DC0

RQB5 =  ria!:R13
        +       rra?:R15
        +       drb?:DQB5
        +       dra?:DC1
        +       tau:DC0

RQB6 =  datia!:RQB4
        +       datra?:RQB8
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB6
        +       dra?:DC1
        +       tau:DC0

RQB7 =  datia!:RQB5
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB7
        +       dra?:DC1
        +       tau:DC0

RQB8 =  datia!:RQB7
        +       ria!:R13
        +       rra?:R15
        +       drb?:DQB8
        +       dra?:DC1
        +       tau:DC0
```

Also, after a disconnect request from user A or B, the behavior of the provider depends on the number of data objects in the queues :

```
DQA0 =    datrb?:DQA1
        +  drb?:I0
        +  dib!:I0

DQA1 =    datrb?:DQA3
        +  drb?:I0
        +  dib!:I0

DQA2 =    datrb?:DQA4
        +  datib!:DQA0
        +  drb?:I0
        +  dib!:I0

DQA3 =    drb?:I0
        +  dib!:I0

DQA4 =    datrb?:DQA6
        +  datib!:DQA1
        +  drb?:I0
        +  dib!:I0

DQA5 =    datrb?:DQA7
        +  datib!:DQA2
        +  drb?:I0
        +  dib!:I0

DQA6 =    datib!:DQA3
        +  drb?:I0
        +  dib!:I0

DQA7 =    datrb?:DQA8
        +  drb?:I0
        +  dib!:I0


DQA8 =    datib!:DQA6
        +  drb?:I0
        +  dib!:I0
```

Behaviour of the provider after a disconnect request of B :

```
DQB0 =    datra?:DQB1
        +  dra?:I0
        +  dia!:I0

DQB1 =    datra?:DQB3
```

```
        +       dra?:I0
        +       dia!:I0

DQB2 =  datra?:DQB4
        +       datia!:DQB0
        +       dra?:I0
        +       dia!:I0

DQB3 =  dra?:I0
        +       dia!:I0

DQB4 =  datra?:DQB6
        +       datia!:DQB1
        +       dra?:I0
        +       dia!:I0

DQB5 =  datra?:DQB7
        +       datia!:DQB2
        +       dra?:I0
        +       dia!:I0

DQB6 =  datia!:DQB3
        +       dra?:I0
        +       dia!:I0

DQB7 =  datra?:DQB8
        +       dra?:I0
        +       dia!:I0


DQB8 =  datia!:DQB6
        +       dra?:I0
        +       dia!:I0
```

After a provider initiated disconnect, no more data is delivered to the users and the connection is released :

```
DC0  =  dia!:DC1
        +       dib!:DC2
        +       dra?:DC1
        +       drb?:DC2

DC1  =  dib!:I0
        +       drb?:I0

DC2  =  dia!:I0
        +       dra?:I0
```

After a provider initiated reset, no more data is delivered to the users too and the connection is reset, which means that the queues are empty after the reset. During the reset procedure, the connection can be released by one of the users or by the provider :

```
R0   =   ria!:R1
     +   rib!:R2
     +   rra?:R9
     +   rrb?:R12
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R1   =   rrpa?:R3
     +   rib!:R4
     +   rrb?:R13
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R2   =   ria!:R4
     +   rrpb?:R5
     +   rra?:R10
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R3   =   rib!:R6
     +   rrb?:R14
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R4   =   rrpa?:R6
     +   rrpb?:R7
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R5   =   ria!:R7
     +   rra?:R11
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0

R6   =   rrpb?:Q0
     +   dra?:DC1
     +   drb?:DC2
     +   tau:DC0
```

```
R7    =    rrpa?:Q0
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R9    =    rib!:R10
      +    rrb?:R15
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R10   =    rrpb?:R11
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R11   =    rca!:Q0
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R12   =    ria!:R13
      +    rra?:R15
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R13   =    rrpa?:R14
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R14   =    rcb!:Q0
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0

R15   =    rca!:R14
      +    rcb!:R11
      +    dra?:DC1
      +    drb?:DC2
      +    tau:DC0
```

APPENDIX B A CCS SPECIFICATION OF LAYER OPERATION

In this chapter the specifications of the different processes described in chapter six will be given, using the formal description method CCS. The commentary on these processes is given in chapter six. Only the abstract behavior equations will be given. In order to avoid memory overflow, the processes had to be described in a more abstract way than in chapter six. We therefore abstracted from internal actions which were not relevant for the interprocess communication.

Connection Control process

```
CC0   =   startconn?:CC1
      +   stopconn?:CC0S

CC1   =   startconn?:CC2
      +   stopconn?:CC1S
      +   resetpend?:CC1R

CC2   =   stopconn?:CC2S
      +   resetpend?:CC2R

CC0S  =   stoppedcon!:CC0

CC1S  =   stoppedcon!:CC0

CC2S  =   stoppedcon!:CC1

CC1R  =   nresetpend!:CC1
      +   stopconn?:CC1S

CC2R  =   nresetpend!:CC2
      +   stopconn?:CC2S
```

Connect process

```
C0    =   conprim?:C1
      +   reconnect?:CR2
      +   stopconnecting?:CSC

C1    =   tau:C2
      +   error!:CS
      +   stopconnecting?:CSC

C2    =   startconn!:C3
      +   stopconnecting?:CSA

C3    =   conprim!:C0
      +   stopconnecting?:CSA
```

```
CSC  =  stoppedc!:CS

CSA  =  stoppeda!:CS

CS   =  startconnecting?:C0

CR2  =  startconn!:CR3
     +  error!:CS
     +  stopconnecting?:CSC

CR3  =  reconnconf!:C0
     +  stopconnecting?:CSC
```

Disconnect process

```
D0   =  discprim?:D1
     +  error?:DE1
     +  reseterror?:DR0
     +  disconnect?:DD0

D1   =  stopconnecting!:D2

D2   =  stopconn!:D3

D3   =  stopreset!:D4

D4   =  stoppedc?:D5
     +  stoppeda?:D5

D5   =  stoppedcon?:D6

D6   =  discprim!:D7

D7   =  startconnecting!:D8

D8   =  startreset!:D0

DE1  =  discprim!:DE2

DE2  =  startconnecting!:D0

DR0  =  stopconnecting!:DR1

DR1  =  stopconn!:D4

DD0  =  stopconnecting!:DD1

DD1  =  stopconn!:DD2
```

```
DD2  =  stoppedc?:DD3
     +  stoppeda?:DD3

DD3  =  stoppedcon?:DD4

DD4  =  disconnected!:DE2
```

Restart process

```
R0   =  restprim?:R1
     +  stopreset?:RS

R1   =  resetpend!:R2
     +  disconnect!:R3
     +  stopreset?:RS

R2   =  nresetpend!:R4
     +  reseterror!:RS
     +  stopreset?:RS

R3   =  disconnected?:R5
     +  stopreset?:RS

R4   =  restprim!:R0
     +  stopreset?:RS

R5   =  reconnect!:R6
     +  stopreset?:RS

R6   =  reconnconf?:R4
     +  reseterror!:RS
     +  stopreset?:RS

RS   =  startreset?:R0
```

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(205) Butterweck, H.J. and J.H.F. Ritzerfeld, M.J. Werter
FINITE WORDLENGTH EFFECTS IN DIGITAL FILTERS: A review.
EUT Report 88-E-205. 1988. ISBN 90-6144-205-2

(206) Bollen, M.H.J. and G.A.P. Jacobs
EXTENSIVE TESTING OF AN ALGORITHM FOR TRAVELLING-WAVE-BASED DIRECTIONAL DETECTION AND PHASE-SELECTION BY USING TWONFIL AND EMTP.
EUT Report 88-E-206. 1988. ISBN 90-6144-206-0

(207) Schuurman, W. and M.P.H. Weenink
STABILITY OF A TAYLOR-RELAXED CYLINDRICAL PLASMA SEPARATED FROM THE WALL BY A VACUUM LAYER.
EUT Report 88-E-207. 1988. ISBN 90-6144-207-9

(208) Lucassen, F.H.R. and H.H. van de Ven
A NOTATION CONVENTION IN RIGID ROBOT MODELLING.
EUT Report 88-E-208. 1988. ISBN 90-6144-208-7

(209) Jóźwiak, L.
MINIMAL REALIZATION OF SEQUENTIAL MACHINES: The method of maximal adjacencies.
EUT Report 88-E-209. 1988. ISBN 90-6144-209-5

(210) Lucassen, F.H.R. and H.H. van de Ven
OPTIMAL BODY FIXED COORDINATE SYSTEMS IN NEWTON/EULER MODELLING.
EUT Report 88-E-210. 1988. ISBN 90-6144-210-9

(211) Boom, A.J.J. van den
H_∞-CONTROL: An exploratory study.
EUT Report 88-E-211. 1988. ISBN 90-6144-211-7

(212) Zhu Yu-Cai
ON THE ROBUST STABILITY OF MIMO LINEAR FEEDBACK SYSTEMS.
EUT Report 88-E-212. 1988. ISBN 90-6144-212-5

(213) Zhu Yu-Cai, M.H. Driessen, A.A.H. Damen and P. Eykhoff
A NEW SCHEME FOR IDENTIFICATION AND CONTROL.
EUT Report 88-E-213. 1988. ISBN 90-6144-213-3

(214) Bollen, M.H.J. and G.A.P. Jacobs
IMPLEMENTATION OF AN ALGORITHM FOR TRAVELLING-WAVE-BASED DIRECTIONAL DETECTION.
EUT Report 89-E-214. 1989. ISBN 90-6144-214-1

(215) Hoeijmakers, M.J. en J.M. Vleeshouwers
EEN MODEL VAN DE SYNCHRONE MACHINE MET GELIJKRICHTER, GESCHIKT VOOR REGELDOELEINDEN.
EUT Report 89-E-215. 1989. ISBN 90-6144-215-X

(216) Pineda de Gyvez, J.
LASER: A LAyout Sensitivity ExploreR. Report and user's manual.
EUT Report 89-E-216. 1989. ISBN 90-6144-216-8

(217) Duarte, J.L.
MINAS: An algorithm for systematic state assignment of sequential machines - computational aspects and results.
EUT Report 89-E-217. 1989. ISBN 90-6144-217-6

(218) Kamp, M.M.J.L. van de
SOFTWARE SET-UP FOR DATA PROCESSING OF DEPOLARIZATION DUE TO RAIN AND ICE CRYSTALS IN THE OLYMPUS PROJECT.
EUT Report 89-E-218. 1989. ISBN 90-6144-218-4

(219) Koster, C.J.P. and L. Stok
FROM NETWORK TO ARTWORK: Automatic schematic diagram generation.
EUT Report 89-E-219. 1989. ISBN 90-6144-219-2

(220) Willems, F.M.J.
CONVERSES FOR WRITE-UNIDIRECTIONAL MEMORIES.
EUT Report 89-E-220. 1989. ISBN 90-6144-220-6

(221) Kalasek, V.K.I. and W.M.C. van den Heuvel
L-SWITCH: A PC-program for computing transient voltages and currents during switching off three-phase inductances.
EUT Report 89-E-221. 1989. ISBN 90-6144-221-4

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(222) Jóźwiak, L.
THE FULL-DECOMPOSITION OF SEQUENTIAL MACHINES WITH THE SEPARATE REALIZATION OF THE NEXT-STATE AND OUTPUT FUNCTIONS.
EUT Report 89-E-222. 1989. ISBN 90-6144-222-2

(223) Jóźwiak, L.
THE BIT FULL-DECOMPOSITION OF SEQUENTIAL MACHINES.
EUT Report 89-E-223. 1989. ISBN 90-6144-223-0

(224) Book of abstracts of the first Benelux-Japan Workshop on Information and Communication Theory, Eindhoven, The Netherlands, 3-5 September 1989.
Ed. by Han Vinck.
EUT Report 89-E-224. 1989. ISBN 90-6144-224-9

(225) Hoeijmakers, M.J.
A POSSIBILITY TO INCORPORATE SATURATION IN THE SIMPLE, GLOBAL MODEL OF A SYNCHRONOUS MACHINE WITH RECTIFIER.
EUT Report 89-E-225. 1989. ISBN 90-6144-225-7

(226) Dahiya, R.P. and E.M. van Veldhuizen, W.R. Rutgers, L.H.Th. Rietjens
EXPERIMENTS ON INITIAL BEHAVIOUR OF CORONA GENERATED WITH ELECTRICAL PULSES SUPERIMPOSED ON DC BIAS.
EUT Report 89-E-226. 1989. ISBN 90-6144-226-5

(227) Bastings, R.H.A.
TOWARD THE DEVELOPMENT OF AN INTELLIGENT ALARM SYSTEM IN ANESTHESIA.
EUT Report 89-E-227. 1989. ISBN 90-6144-227-3

(228) Hekker, J.J.
COMPUTER ANIMATED GRAPHICS AS A TEACHING TOOL FOR THE ANESTHESIA MACHINE SIMULATOR.
EUT Report 89-E-228. 1989. ISBN 90-6144-228-1

(229) Oostrom, J.H.M. van
INTELLIGENT ALARMS IN ANESTHESIA: An implementation.
EUT Report 89-E-229. 1989. ISBN 90-6144-229-X

(230) Winter, M.R.M.
DESIGN OF A UNIVERSAL PROTOCOL SUBSYSTEM ARCHITECTURE: Specification of functions and services.
EUT Report 89-E-230. 1989. ISBN 90-6144-230-3