An Implementation of the OSI Presentation Layer

E.J.L.H. van Hout

Eindhoven, October 1992

Graduation report

Professor: prof. ir. M.P.J. Stevens
Coach: ir. M.J.M. van Weert

The faculty of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of student project- and graduation reports.
Abstract.

Incompatibility between different computer systems, due to different architectures and protocols, limits the exchange of information between those systems. To overcome this problem the International Standards Organization (ISO) defined the Open Systems Interconnection (OSI) reference model. The reference model defines seven layers, each of which performs a well-defined function.

The Digital Information Systems group of the faculty of Electrical Engineering at the Eindhoven University of Technology researches the presentation layer of the reference model. The main goal is to implement this layer in hardware to have a better speed performance than the known software implementations.

The presentation layer is concerned with preserving the meaning of the transported information. Two applications on different computers can communicate regardless of the internal data representation in each of the computer systems by using the service provided by the presentation layer. The data under transport is encoded using a common syntax, called transfer syntax, understood by both presentation layer entities. For encoding and decoding the data according to this transfer syntax, the structure of the data, called abstract syntax, has to be known to both presentation entities. The combination of an abstract syntax and a transfer syntax is called a presentation context. Data can be of different abstract syntaxes and encoded with different transfer syntaxes so, a set of presentation contexts is defined during a connection, this set is called the Defined Context Set (DCS).

The presentation layer can be split in several parts, one of which is the Presentation Protocol Machine (PPM). This thesis is concerned with the PPM, which has as major task to keep the DCS up to data. Two phases of a connection can be distinguished: the connection establishment phase and the connected phase. The main function of the connection establishment phase is to negotiate about the DCS which is going to be used during the connection. The connected phase has as main function the transportation of information which is encoded according to a presentation context of the DCS.

The modelling technique, used to create a structured design of the presentation layer, is the method according to Hatley and Pirbhai. The requirements model is introduced, which describes how a system should work in a formal way. The CASE-tool ProMod is used to realize the requirements model of the presentation layer. In this thesis the requirements model of the PPM is described. The functionality of the PPM as it is described in the recommendations is divided in two processes, the PPM process and the filter_dcs process. The PPM process is mainly concerned with the peer-to-peer communication protocols and the filter_dcs process is concerned with managing the DCS.

The PPM process is further decomposed in two major processes: the connection establishment process and the connected process. These are alternating processes, so only one of them is active at a time. The PPM process controls the filter_dcs process by sending commands. To keep the DCS up to date, the filter_dcs process has to remember the complete history of the DCS. The history of the DCS is kept in the manage_dcs store outside the filter_dcs process.
A good memory management is needed to organize the information in this store so the filter_dcs has fast access to the needed information.

The requirements model of the PPM is extended and improved. The filter_dcs process is now completely defined. The PPM process is extended and details are worked out. Especially a lot of work is done improving the error management and connection release processes. The requirements model is checked for consistency, with help of the analyze function of the CASE tool ProMod, and no syntactical errors are detected in the requirements model as it is now defined.
Contents.

Abbreviations

1 Introduction

2 The OSI Reference Model
   2.1 The layers and their function
   2.2 Communication
      2.2.1 Peer to Peer communication
      2.2.2 Communication between different layers
   2.3 Summary

3 The Presentation Layer
   3.1 Introduction
   3.2 Presentation context sets
      3.2.1 Context set definitions
      3.2.2 Context set negotiation
   3.3 The connection establishment phase
   3.4 The connected phase
      3.4.1 Information transfer
      3.4.2 Activities
      3.4.3 Synchronization and Resynchronization
      3.4.4 Token management
      3.4.5 Exception reporting
      3.4.6 Context management
      3.4.7 Connection release
   3.5 Summary

4 Modelling the Presentation Layer
   4.1 Introduction
   4.2 Structured design
      4.2.1 The requirements model
      4.2.2 The CASE tool ProMod
      4.2.3 Modelling considerations
   4.3 The requirements model of the presentation layer
      4.3.1 The context of the presentation layer
      4.3.2 The first level of decomposition
   4.4 Summary

5 The Presentation Protocol Machine
   5.1 The decomposition of the PPM
   5.2 Connection establishment
      5.2.1 The Connection PDU assembler
      5.2.2 The processes performing negotiation
      5.2.3 Addressing
| 5.2.4 Error management | 38 |
| 5.2.5 Terminate establishment | 38 |
| 5.3 Connected | 39 |
| 5.3.1 The decomposition of the connected process | 39 |
| 5.3.2 Context management | 40 |
| 5.4 Summary | 42 |

6 The Filter | 43 |
| 6.1 The filter commands | 43 |
| 6.1.1 Alter manage store commands | 43 |
| 6.1.2 Fill DCS store commands | 44 |
| 6.1.3 Data checking commands | 45 |
| 6.1.4 Context set retrieve commands | 45 |
| 6.1.5 Synchronization and activity commands | 46 |
| 6.2 The decomposition of the filter | 47 |
| 6.3 The DCS administration | 49 |
| 6.4 Summary | 51 |

7 Conclusions and Recommendations | 52 |
| 7.1 Conclusions | 52 |
| 7.2 Recommendations | 52 |

8 Literature | 54 |

Appendix A: Tables of PPDU types and service primitives | 57 |
Appendix B: ProMod structured analysis report | 60 |
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASN1</td>
<td>Abstract Syntax Notation One</td>
</tr>
<tr>
<td>BER</td>
<td>Basic Encoding Rules</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Systems Engineering</td>
</tr>
<tr>
<td>CEP</td>
<td>Connection End Point</td>
</tr>
<tr>
<td>CEPI</td>
<td>Connection End Point Identifier</td>
</tr>
<tr>
<td>CFD</td>
<td>Control Flow Diagram</td>
</tr>
<tr>
<td>CSPEC</td>
<td>Control Specification</td>
</tr>
<tr>
<td>DCS</td>
<td>Defined Context Set</td>
</tr>
<tr>
<td>DFD</td>
<td>Data Flow Diagram</td>
</tr>
<tr>
<td>FU</td>
<td>Functional Unit</td>
</tr>
<tr>
<td>ICI</td>
<td>Interface Control Information</td>
</tr>
<tr>
<td>IDU</td>
<td>Interface Data Unit</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PCI</td>
<td>Protocol Control Information</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PPDU</td>
<td>Presentation Protocol Data Unit</td>
</tr>
<tr>
<td>PPM</td>
<td>Presentation Protocol Machine</td>
</tr>
<tr>
<td>PSAP</td>
<td>Presentation Service Access Point</td>
</tr>
<tr>
<td>PSPEC</td>
<td>Process Specification</td>
</tr>
<tr>
<td>SAP</td>
<td>Service Access Point</td>
</tr>
<tr>
<td>SAPI</td>
<td>Service Access Point Identifier</td>
</tr>
<tr>
<td>SDU</td>
<td>Service Data Unit</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Information plays a very important role in the modern society. The amount of transported information constantly increases, and so is the speed at which this information transport takes place. In the past a simple letter was send by courier and it took days to arrive at its destination. Nowadays people all over the world can watch the Olympic games live on television. The invention of the computer made it possible to store and manipulate huge amounts of information, and with computer-networks this information can be transported.

However, incompatibility limits the exchange of information between computer systems. Incompatibility is a result of different computer architectures and communication protocols. This problem was widely recognized and resulted in the definition of the Open Systems Interconnection (OSI) reference model, which standardizes the communication between computers of any kind. The reference model splits the communication problem in seven parts, called layers. Each of these layers performs a well-defined function and can be implemented independent from all other layers.

Implementations already appeared on the market [Mant88], but with these products the upper layers of the model were implemented in software. Implementing layers in software has one major disadvantage, that is a poor operation speed. The Digital Information Systems Group of the faculty of Electrical Engineering at the Eindhoven University of Technology also does research to implementing the OSI reference model. The main goal is to implement the layers in hardware. Parts of the lower three layers have already been implemented and now the work is focused on the sixth layer: the presentation layer.

The presentation layer is concerned with preserving the meaning of the transported information. This means that regardless of the coding used to represent information in the sending computer, the information reaches the receiving computer in its coding used for representing information. The presentation layer can be seen as a translating layer, which consists of an encoder and decoder unit and a control unit called: Presentation Protocol Machine (PPM).

The first step in implementing the presentation layer is to develop a requirements model. Two former students started with the functional description of the presentation layer, but the complexity of the presentation layer made it impossible to complete the model in their graduation period. The model was incomplete and details were not worked out. The development of the requirements model continued and as a result the model of the encoder and decoder unit has been finished [Reut92].
This thesis will describe the improved and extended requirements model of the PPM. The first chapters introduce the OSI reference model and discuss the presentation layer. Good literature is available, so these chapters only give a brief overview and refer to literature. Chapter 4 discusses the modelling method used. The requirements model is designed, using the structured development method according to Hatley and Pirbhai, with the CASE-tool ProMod. The report generated with this tool is included in appendix B. The following two chapters describe the requirements model of the PPM. The PPM is divided in two parts so, each chapter discusses one part of the model. Finally, in chapter 7 the conclusions of the graduation work and some recommendations for further study towards an implementation are given.
Chapter 2

The OSI Reference Model

The OSI reference model was developed in 1979 by the International Standards Organization (ISO). ISO is an international organization established to promote standardization worldwide (see [Knig88] chapter 2). The reference model defines a framework for system-independent communication between computer systems. The OSI reference model is defined in recommendation [X.200].

This chapter gives a brief overview of the reference model. For a detailed discussion the reader is referred to literature [Hens88] chapters 1, 2 and 3 and [Knig88] chapter 3.

2.1 The layers and their function

The reference model defines seven layers, each of which has a well defined function to perform. The layers are chosen in such a way that similar or highly inter-related functions are collected in one layer. This approach makes it possible to define interfaces between the layers so each layer becomes a completely defined subsystem, and as a result the implementation of a particular layer is independent of all other layers. The seven layers form a stack where a higher layer uses the services of a lower layer. Figure 2.1 shows the seven layer stack. In the figure is shown that a layer only has interfaces with the one above and the one below.

![Figure 2.1 The OSI stack](image_url)
The seven layers which are shown in the figure have the following functions:

**Application layer**
Is the top layer of the OSI stack and provides the interface between the system-dependent end-user and the system-independent communication service. The layer contains service elements to support application processes such as file transfer, job management and message handling services [Beve89].

**Presentation layer**
Is concerned with preserving the meaning of the transported information. Furthermore data compression and data encryption are services provided by this layer. In the next chapter these services are described in more detail.

**Session layer**
This layer provides an organized exchange of data between users, such as simultaneous transmission, alternate transmission (token management), and checkpoint procedures (synchronization).

**Transport layer**
It splits the data from the session layer into smaller units when needed, passes these to the network layer and ensures that the pieces all arrive correctly at the other end.

**Network layer**
Controls the operations of the sub-network. It is concerned with routing packets from source to destination, relaying to allow data transmissions from one sub-network to another and congestion control.

**Data Link layer**
Is responsible for the error-free transfer of data over the network. It is also concerned with flow control to prevent a slow receiver to be overburdened with too much data from a fast sender.

**Physical layer**
Takes care of transmitting raw bits over a physical communication channel. Typical issues are output voltage and timing aspects.

### 2.2 Communication

For communication the layer function must be implemented on both computer systems, called end-systems. When such an implementation is involved in an OSI communication it is called a layer entity. A layer can thus be considered as being split across two end-systems and represented on each by a layer entity. Two entities in different end-systems but in the same layer are called peer entities.

Communication in the OSI reference model is defined as the cooperation of two peer entities. This implies that the application layer entity, for example, in one end-system communicates
with the application layer entity in the other end-system, this is called peer to peer communication. In figure 2.2 this peer to peer communication is shown as a dotted line because it is not a physical connection. The only way for the application layer entity to communicate with its peer, is to use the services of the presentation layer. The presentation layer entity on its turn wants to communicate with its peer so it has to use the services of the session layer. This procedure is repeated until the physical layer is reached. The physical layer entity has a real physical connection with its peer, so now information can be transported over a real physical link. The real information exchange is shown as a drawn line in figure 2.2.

![Diagram of OSI reference model](image)

**Figure 2.2 Communication in the OSI reference model**

### 2.2.1 Peer to Peer communication

The basic idea of peer to peer communication is that it looks like the communication takes place over the virtual path, so it is irrelevant to know how the real communication path is if one is concerned with the communication of two peer entities. The virtual path looks like a real path by using the services of the underlying layer, which is called the service provider.

The complete set of services a layer can offer to a higher layer is called the layer service definition. This set consists of several subsets called functional units (FU). It is not always necessary that all these functional units are available, some of them may not be needed for a certain connection. A higher layer, called service user, can therefore request the necessary functional units.
The services are all executed according a certain protocol specification. The messages send to the peer during execution of a service are called protocol data units (PDU). These PDUs contain control information for the peer entity and user data. This user data is received from the service user together with the request for service. The service provider does nothing with this user data, it is only transported over the virtual connection. To have a virtual connection the services of the underlying layer has to be used, so the user data together with the control information is actually sent to the underlying layer. The next paragraph will discuss this inter-layer communication.

### 2.2.2 Communication between different layers

In the OSI reference model service primitives are used to indicate the service a service user wants from a service provider. A service can be one of the following types:

- Unconfirmed
- Confirmed
- Indication

The following service primitives are used for these services:

- Request
- Indication
- Response
- Confirm

The following figures show the three types of services with the corresponding service primitives used.

![Figure 2.3 An unconfirmed service](image)

Figure 2.3 shows an unconfirmed service, where only the request and indication service primitives are used. The service user sends a request service primitive to the service provider,
which will have peer to peer communication with its peer entity. In the other end-system the peer entity issues an indication service primitive and the service is completed. An example of this kind of service could be data transfer, where the sender does not need to know if the data is received properly.

![Diagram of a confirmed service](image)

**Figure 2.4 A confirmed service**

Figure 2.4 shows a confirmed service. With this type of service all service primitives are used. With a confirmed service the service user will issue a response service primitive after reception of an indication service primitive. The service provider again will do peer to peer communication and a confirm service primitive is issued. Here an example could be a request for connection, where the answer comes with the confirm service primitive.

![Diagram of an indication service](image)

**Figure 2.5 An indication service**

Figure 2.5 shows an indication service, which is a special case of service. With this service some event in a lower layer occurred and only an indication service primitive is issued. An example of an event which could occur is a failure in the communications equipment.

These service primitives have associated parameters and user data is in most cases one of them. These parameters together form a PDU, so now the link with peer to peer communication is clear. If a layer wants to commit peer to peer communication the PDU is send to the lower layer together with a service primitive, and the lower layer becomes the service provider. Figure 2.6 gives a full overview of how a PDU is formed and of what it
The figure shows the situation for one layer (N layer) which interfaces with the layer above (N+1 layer) and the layer below (N-1 layer). The process repeats in every layer and to distinguish the same components of different layers, the name of the layer is put in front of the name of the data unit, so a PDU becomes a PPDU in the presentation layer. The assembling and disassembling of the following components is shown:

**IDU** The Interface Data Unit consists of a PDU and an ICI of the (N+1) layer. It is the total amount of information exchanged across the layer boundary.

**ICI** Interface Control Information is only used by the N layer to execute its function and is not passed to the (N-1) layer.

**SDU** The Service Data Unit is all information received from the (N+1) layer which has to be transported to the (N-1) layer. It is the PDU of the (N+1) layer.

**PCI** The Protocol Control Information is the control information for the peer entity of the N layer to perform a service function.

**PDU** The Protocol Data Unit consist of a PCI and a SDU and is the total amount of information that seems to be transported over the virtual path to the peer entity.

At the receiving end-system the process is reversed, which could be made visible by reversing
the arrows in figure 2.6.

The Interface Data Units (IDU) cross the layer boundary at specific points, so called Service Access Points (SAP). The one or more SAP's of the N layer are located between the N layer and the (N+1) layer, and are called N SAP's. Associated with each SAP is an address, called Service Access Point Identifier (SAPI). Only one (N+1) layer entity is allowed to reside above a particular N SAP and thus the address of that N SAP can be used to identify the (N+1) entity. Several N connections can be established through a particular N SAP. Each N connection is represented by an N Connection End Point (CEP), associated with that N SAP. An N CEP has an identifier unique with the N SAP, called the Connection End Point Identifier (CEPI). The N SAPI and N CEPI together uniquely identify the N connection within both the N entity and the (N+1) entity.

In the lower layers a SAPI is used to indicate a particular end-system to which the information has to be send. In the higher layers one can think of one end-system which has made several connections to other end-systems. The entity concerned with a specific connection is identified with the SAPI, while the CEPI identifies to which connection the transferred information belongs. More about communication between different layers can be found in [blac89] pp.269-283.

2.3 Summary

This chapter has given a brief description of the OSI reference model. The seven layer model has been introduced with an overview of the function of each layer. It is shown that every layer entity has a virtual connection with its peer by using the services of the layer below. The use of service primitives and associated parameters is discussed. The services are available through SAP's to which one entity is connected. This chapter was meant to make the reader familiar with the terminology used in the OSI reference model and to explain the most important issues, which will be used later on in this thesis.
Chapter 3
The Presentation Layer

This chapter gives a description of the presentation layer with special interest for the presentation protocol machine, because it is the main subject of this thesis. The presentation layer is defined in the standards [X.216] and [X.226]. For a full discussion the reader is referred to literature [Rose90] chapter 9 and [Reut92] chapters 3 and 5.

3.1 Introduction

The presentation layer is, as stated before, concerned with preserving the meaning of the transported information. Furthermore, data compression and data encryption are services provided by this layer. A particular example of different ways to represent information is the use of ASCII or EBCDIC for character definitions. If two end-systems want to communicate and one of them uses ASCII and the other EBCDIC, the presentation layer has to do conversion. This conversion can be done in three different ways:

- The sender converts its information to the format of the receiver.
- The receiver converts the received information to its own format.
- The sender converts the information to a standard format and the receiver converts the received information in the standard format to its own used format.

The first two options are fast because only one conversion has to be done, but the end-system which does the conversion has to know the format of the other end-system. If many end-systems are connected through a network a lot of conversion routines are needed per end-system if its wants to communicate with all the others. The last option requires two conversions of transported information, but now end-systems only have to know the standard format to be able to communicate with all the others. ISO has chosen for the last option and information in transit is said to have a common syntax. In each presentation entity conversion of data will take place, even if two end-systems with the same data representation are communicating. More about data representation can be found in [Beve88] and [Tay87].

From the former discussion it seems that only one common syntax is used, but this is not the case. Several common syntaxes can be used during a connection and the set of used common syntaxes can even change during a connection. The common syntax which will be used depends upon the information to be transported and the set of common syntaxes supported by both entities. During connection establishment the peer entities have to negotiate about the set of common syntaxes which will be used. Once a connection is established this set of common syntaxes can change, so again negotiation is necessary. Negotiating about the common syntax is one of the tasks of the presentation protocol machine (PPM). The PPM can be seen as the control unit of a presentation entity. Figure 3.1 shows the most simple model of a presentation entity. It consists of an encoder/decoder unit, the PPM and two buffers to hold the data to and from the boundary layers.
3.2 Presentation Context Sets

3.2.1 Context set definitions

In the preceding paragraph it was mentioned that the used common syntax depends on the information to be transported. This implies that the presentation layer has to know the data structure of the information to be transported. The description of this data structure is known as the abstract syntax. With this abstract syntax a name is associated which is called the abstract syntax name. OSI defines a notation to describe the abstract syntax. This notation is called abstract syntax notation 1 (ASN.1). A full description of ASN.1 can be found in the standard [X.208] and literature [Rose90] chapter 8.

The presentation layer is able to select an appropriate common syntax, called transfer syntax, for each abstract syntax. One transfer syntax is already defined by OSI and called the basic encoding rules (BER). In future more transfer syntaxes could be defined, implementing for example data compression and encryption. A full description of BER can be found in the standard [X.209] and literature [Rose90] chapter 8.

The abstract syntax together with the appropriate transfer syntax form a presentation context. A transfer syntax can be used with more than one abstract syntax and an abstract syntax can be encoded by several transfer syntaxes. To identify uniquely these combinations a presentation context identifier is used. Figure 3.2 shows an example of a presentation context.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>ABSTRACT SYNTAX NAME</th>
<th>TRANSFER SYNTAX NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>STUDENT</td>
<td>BER</td>
</tr>
</tbody>
</table>

Figure 3.2 An example of a presentation context
During a connection a set of presentation contexts is used. This set is called the **defined context set** (DCS). Information which has to be encoded is first checked to which presentation context it belongs. Then the information is encoded according to the specified transfer syntax. At the receiving end-system, information is again checked to which presentation context it belongs and decoded. One special presentation context exists which is not part of the DCS, it is called the **default context**. If the DCS is empty, the default context remains and the data, if the structure is according to the abstract syntax, will be encoded according to the specified transfer syntax. If the data can not be encoded according to the DCS or the default context, an error will occur. Managing the DCS and the default context is the task of the PPM.

From the former discussion it is clear that both end-systems must have the same DCS, and default context. It is not enough to let the peer entity know which presentation contexts are going to be used, it might have no support for certain presentation contexts. The only way to form a DCS and default context supported by both presentation entities is to negotiate about them.

### 3.2.2 Context Set Negotiation

During connection establishment context set negotiation is used to form a DCS and default context. Once a connection is established the DCS can be altered using context set negotiation. During context set negotiation peer to peer communication will take place. In the connection establishment phase the CP, CPA and CPR PPDU’s are used. In the connected phase the AC and ACA PPDU’s are used. Appendix A gives an overview of all PPDU’s and service primitives used by the PPM.

**Default context negotiation**

The default context is formed in the connection establishment phase and can not be altered during the connected phase. If the application layer requests a presentation layer connection it issues a P_connect.request service primitive and one of the associated optional parameters is the **default context name**\(^1\). The PPM selects an appropriate transfer syntax and adds its name to the default context, which now consists of an abstract syntax name and a transfer syntax name. The default context name becomes part of the CP PPDU, which is send by the PPM (initiating PPM) to its peer (responding PPM).

The responding PPM checks if the default context is supported. If no support is available a CPR PPDU is send back and no connection will be established. If the default context is supported a P_connect.indication service primitive is send to the peer application entity. The peer application entity checks if it supports the abstract syntax of the default context and the result of this check is send back in the P_connect.response service primitive. If the application entity rejects the default context a CPR PPDU is send back and no connection will be established. If the default context is accepted the establishment of the connection depends on

---

\(^1\) If this parameter is not present, it is assumed that the presentation layer has some prior agreement about the default context and negotiation will not take place.
other factors, but if it is not accepted a connection will never be established.

**Defined context set negotiation**

The DCS negotiation differs in a few ways from the default context negotiation. First the DCS consists of a set of presentation contexts so a **presentation context definition list** instead of a default context name and a **presentation context definition result list** instead of the default result are used. Second, the initiating PPM adds not one but all appropriate transfer syntaxes to an abstract syntax and the responding PPM has to select one. The DCS negotiation is discussed in two phases, phase 1 is the request from initiator to responder and phase 2 is the acknowledge from responder to initiator.

![Diagram of DCS negotiation, phase 1](image)

**Figure 3.3** DCS negotiation, phase 1

Figure 3.3 shows the DCS negotiation phase 1. In this figure a presentation context identifier is abbreviated to 'ID', abstract syntax to 'AS' and transfer syntax to 'TR'. The application entity requests a presentation layer connection by sending a _P_connect.request_ service primitive. One of the optional parameters is the presentation context definition list\(^2\). The initiating PPM adds all appropriate transfer syntaxes to each abstract syntax and sends the resulting list as a parameter of the CP PPDU to the responding PPM. The responding PPM checks the list for not supported presentation contexts. A presentation context is not supported if either the abstract syntax or none of the proposed transfer syntaxes is supported. The results

---

\(^2\) If no list is present, only the default context can be used for encoding data.
of this check are stored in the presentation context definition result list. In this list 'accepted' is abbreviated to 'AC' and 'Provider rejection' to 'PR'. Together with the presentation context definition list the result list is send as a parameter of the P_connect.indication service primitive to the responding application entity. At this point phase 2 starts, which is shown in figure 3.4.

Figure 3.4 DCS negotiation, phase 2

The responding application entity checks all items in the presentation context definition list which are marked as accepted in the presentation context definition result list. If the responding application entity does not support an abstract syntax, the corresponding item in the presentation context definition result list is marked with 'User Rejection' (UR). The altered presentation context definition result list is, as a parameter of the P_connect.response service primitive, send back to the responding presentation entity.

The responding presentation entity now selects one transfer syntax for each accepted presentation context and stores them in the presentation context definition result list. At this point the responding presentation entity knows the resulting DCS. The presentation context definition result list is, as a parameter of the CPA or CPR PPDU, send to the initiating presentation entity. The initiating presentation entity now knows the resulting DCS and sends a P_connect.confirm service primitive to the application entity. The presentation context definition result list is send as a parameter of the service primitive to let the application entity know which presentation contexts can be used.

The example discussed in this paragraph is the situation found at connection establishment. Once a connection has been established and the presentation layer is said to be connected
other service primitives and PPDU's are used, but the principle stays the same. During the connected phase the P_alter_context service primitive and AC and ACA PPDU's are used.

3.3 The Connection Establishment Phase

The connection establishment phase provides a service to establish a connection between two presentation entities. The presentation service primitive used is the P_connect primitive and for peer to peer communication the CP, CPA and CPR PPDU’s are used. The CP PPDU is used to request a connection and the answer comes in the form of a CPA or CPR PPDU. A CPA PPDU is a positive answer to the request for connection, while a CPR PPDU rejects a connection. During the connection establishment phase the peer entities negotiate about the following issues, and a connection is only established if the negotiation of all of them is successful.

**Presentation Protocol Version**

The presentation layer acts according to a protocol which is executed by the PPM. The initiating PPM provides a list of protocol versions it is capable of supporting and sends this list in a CP PPDU to its peer. The peer selects one protocol of this list and sends it back in the CPA or CPR PPDU. If none of the proposed protocols is supported by the peer it will always send a CPR PPDU. The protocol described in the recommendations [X.216] and [X.226] is defined as protocol version 1.

**Presentation Functional Units**

The services of the presentation layer are divided in three subsets, called Functional Units (FU’s). These functional units are: the Kernel, the Context Management (CM) and the Context Restoration (CR) functional unit. The Kernel FU provides the presentation layer with the basic functions and is therefore always selected. The context management FU contains the services to alter the DCS during the connected phase, its use is optional. The context restoration FU contains the services to restore the DCS to the contents defined earlier during the connection, it is optional and can only be selected if the context management FU is selected. The initiating PPM sends a list of selected FU’s in the CP PPDU and the responding PPM will send this list back in the CPA or CPR PPDU with only the supported FU’s.

**Default Context and Defined Context Set**

These issues are discussed in the previous paragraph.

The PPDU’s used in the connection establishment phase can optionally contain user data. This user data comes from the application layer and therefore it has to be encoded. The data contained in the CP PPDU must be encoded without notice if the peer entity supports the used presentation context. To increase the chance that the peer can decode the data, it can be encoded more than once using different proposed transfer syntaxes. When none of the used transfer syntaxes is supported by the peer, the data can not be decoded and the peer will send a CPR PPDU.

If negotiation is successful and the data, if present, can be decoded, the connection can be
established and the presentation layer will be in the connected phase.

3.4 The Connected Phase

After a connection has successfully been established the presentation layer is said to be in the connected phase. During the connected phase several services are available, but the most important is of course information transfer. The following paragraphs discuss these services and special attention is focused on the use and changes of the DCS.

3.4.1 Information transfer

Information can be transferred using almost any presentation service. The information from the application layer is transferred in the user data parameter, which is an optional parameter included in almost every presentation service primitive. Four presentation services exists, specially meant for transporting information.

Normal data transfer
This is an unconfirmed service where the data is transferred from the presentation service user to its peer. The data is encoded according to the current DCS. If the DCS is empty the data is encoded according to the default context.

Typed data transfer
This service is the same as the former one, with the exception that the data transfer is not subject to data token restrictions. The use of data tokens is explained in paragraph 3.4.4. The data is encoded according the current DCS. If the DCS is empty the data is encoded according the default context.

Expedited data transfer
An unconfirmed service, which guarantees that the data arrives at the destination before any subsequently issued primitive. This implies that the order in which the data is send is the same as it is received at the other end-system. The data is always encoded according the default context.

Capability data transfer
This is the only confirmed data transfer service. It is invoked if control information has to be transferred between the peer application entities. The data is encoded according the current DCS. If the DCS is empty the data is encoded according the default context.

How the data transferred with other presentation services is encoded depends on the service and the result of the actions taken by that service. An example is the alter context service, where the data may not be encoded according presentation contexts which are accepted for deletion.
3.4.2 Activities

Activity management is a service provided by the session layer. The application layer can only use the services of the session layer by invoking the presentation layer, so the presentation layer has to offer this service to the application layer. The presentation layer itself is not concerned with this service, unless the context restoration FU is selected.

During a session connection different logical pieces of work can be distinguished, called activities. The complete session may comprise a number of activities, but only one activity is allowed at a time. An activity can be started, interrupted, resumed, discarded and ended. If an activity is interrupted another activity can start and possibly alter the DCS. If the interrupted activity is resumed the DCS may not be the same as it was at the time the activity was interrupted. The presentation layer will restore the DCS associated with the resumed activity if the context restoration FU is selected. If this FU is not selected the DCS will not be restored. Figure 3.5 gives an example of activity management in case the context restoration FU is selected.

![Figure 3.5 Activity Management](image)

The figure shows activity 1 which is interrupted as the DCS was set to the contents depicted as DCS 2. The DCS is restored to this contents as activity 1 is resumed.

3.4.3 Synchronization and Resynchronization

Synchronization is a service provided by the session layer and the presentation layer only acts if the context restoration FU is selected. When transmitting data, synchronization points can
be inserted to divide the datastream into a number of units. There are two types of synchronization points:

- Major synchronization points
- Minor synchronization points

Major synchronization points are used to structure the exchange of data into a series of dialogue units. The characteristic of a dialogue unit is that all communication within it is completely separated from all communication before it and after it. A major synchronization point is set between these dialogue units. Minor synchronization points are used to structure the data exchange within a dialogue unit. Both types of synchronization points are identified by a synchronization point serial number. The presentation layer does not alter the DCS when minor or major synchronization points are set. It only has to remember the contents of the DCS associated with a synchronization point if the context restoration FU is selected.

The synchronization service is only useful in combination with the resynchronization service. The resynchronization service restores the connection to a specific synchronization point. The synchronization point serial number is used to identify the point to where the connection has to be restored. If the context FU is selected, the presentation layer has to restore the DCS to the one associated with the synchronization point. Figure 3.6 gives an example of the synchronization and resynchronization service.

![Diagram](image)

The figure shows a very simple situation where the DCS is restored to a specified synchronization point. The behaviour of the service depends not only on the selected FU's but also on the type of resynchronization service. The type shown in figure 3.6 is the set type, but two other types exist: the restart type and the abandon type. The restart type restores the connection to the synchronization point after the last acknowledged major
synchronization point. The abandon type sets the synchronization point serial number to an unused value. How the DCS is altered with these resynchronization types depends on a number of factors. In some cases the DCS is set according to a list, called presentation context definition list, which is send as a parameter of the RS or RSA PPDU. The exact description of how the DCS is set with this service can be found in recommendation [X.226].

3.4.4 Token Management

Token management is a service provided by the session layer. The presentation layer is only used by the application layer to invoke this service. Four tokens are defined, each dynamically assigned to one session service user at a time to permit certain services to be invoked. As the presentation layer only forms a bridge for this service, the session service user is an application layer entity.

If a token is assigned to an application layer entity, this entity is called the owner of that particular token. The tokens with the services controlled by them are:

Data Token
   Controls all Data transfer services, except the Typed Data Transfer service.

Release Token
   An orderly release may only be initiated by the owner of the token.

Synchronization-minor Token
   Minor synchronization or resynchronization services may only be initiated by the owner of the token.

Major/activity Token
   Major synchronization or activity services may only be initiated by the owner of the token.

For a full discussion of the token management services the reader is referred to recommendation [X.215].

3.4.5 Exception Reporting

The exception reporting is again a service provided by the session layer. Two types of exception reporting exist:

Provider exception reporting service
   It permits session service users to be notified if a service can not be executed due to protocol errors or malfunctions in the session layer.

User exception reporting service.
   It permits a session service user to report an error condition to its peer.

---

3 The RS and RSA PPDU are used for peer to peer communication during the resynchronization service.
For a full discussion of the exception reporting services the reader is referred to recommendation [X.215].

3.4.6 Context Management

Context management is used to alter the DCS during a connection. The presentation service primitive used is the P\_alter\_context primitive. For peer to peer communication the AC and ACA PPDU's are used. The DCS can be altered by adding new presentation contexts to the DCS, or deleting existing presentation contexts from the DCS. The presentation contexts proposed for addition are stored in the presentation context addition list and the presentation contexts proposed for deletion are stored in the Presentation context deletion list. Negotiation about the proposed additions and deletions is done as discussed in paragraph 3.2.2. Two lists are used for proposing additions and deletions so, two lists with results are are send back to the initiator, see figure 3.4. These result lists are the presentation context addition result list and the presentation context deletion result list. After negotiation the DCS is updated with the accepted additions and deletions.

3.4.7 Connection Release

A connection can be released by three different services:

The presentation release service
This is an orderly confirmed release of the connection. The service is provided by the session layer. The presentation connection is released as a result of the release of the session connection. The connection is released after all data in transit has been delivered so, no data will be lost.

The presentation user abort service
This is a destructive release of the connection. Any information still in transit may be lost if this service is invoked. A application entity can invoke this service to force the release of a presentation connection at any time and have the peer informed about this termination. The service can transport user data which is encoded according to the DCS of the initiator. The service can be invoked at any time so, the DCS of the responder may not be the same as the DCS of the initiator. Therefore, a presentation context identifier list\(^4\) is send to inform the responder of the presentation contexts used to encode the data. The user data parameter is used by the application entity to inform its peer about the reason for the release of the connection.

The presentation provider abort service
This service is also a destructive way to release a connection. This service is initiated by the presentation service provider and is therefore an indication service as explained

\(^4\) The DCS of the initiator and responder can only be different during connection establishment, or during the connected phase if the context management FU is selected. If this is not the case or no user data is present, this list is will not be send.

20
in paragraph 2.2.2. The presentation service provider can invoke this service to indicate the release of a connection due to an error condition.

3.5 Summary

This chapter discussed the function and use of the presentation layer. The meaning and use of the Defined Context Set has been explained. Attention was specially focused on the Presentation Protocol Machine. The major task of the PPM is to keep the DCS up to data. Several services the PPM provides are explained and the use or alterations of the DCS are specially mentioned. The negotiation of the DCS is explained in detail to give the reader a good understanding of this important service of the PPM. Two phases of a connection are introduced: the connection establishment phase and the connected phase. The main function of the connection establishment phase is to negotiate about several issues. The connected phase has as main function the transportation of information.
Chapter 4

Modelling the Presentation Layer

4.1 Introduction

The specifications of the presentation layer are described in the recommendations [X.216] and [X.226]. These recommendations are written in a very vague way so it takes a lot of time to understand them. The reader also has to have knowledge of the reference model [X.200], the session layer [X.215], ASN.1 [X.208] and the Basic Encoding Rules [X.209]. The presentation layer is very complex so designing an architecture directly from the recommendations is a rather hopeless task.

The problem of implementing ever more complex systems was recognized and many different models were devised to overcome the gap between the specification of a system and its implementation [Lomb87]. Nowadays, a number of methods and tools are available to aid in every phase of system development.

This chapter will first discuss the chosen modelling technique and the CASE\(^5\) tool used to realize the model. Furthermore, the requirements model of the presentation layer will be introduced. The parts of the requirements model this thesis is concerned with will be further explained.

4.2 Structured Design

The chosen modelling technique, used for structured design, comes from D.J. Hatley and I.A. Pirbhai. This technique is described in the book "Strategies for Real-Time System Specification", see [Hatl87]. The method consists of two models, the requirements model and the architecture model. The requirements model describes what the system must do, without any concern about the final implementation. The architecture model describes how the system will do it. The architecture model therefore is concerned with implementation matters like physical entities and data channels.

4.2.1 The Requirements model

The requirements model is based on the use of data flow diagrams (DFD) and control flow diagrams (CFD). These diagrams show data or control flows and processes which act on these flows. The system is decomposed into a hierarchical structure by decomposing the processes into new flow diagrams containing subprocesses.

---

\(^5\) Computer Aided System Engineering: The full or partial automated design of one or more phases of system development [Vini91].
The top level of this hierarchical structure is formed by the context diagram, which is a special sort of flow diagram. The data context diagram is the top level diagram of the data flow diagrams and the control context diagram is the top level diagram of the control flow diagrams. The data and control context diagrams depict the communication of the system with its environment in terms of data and control flows respectively. Figure 4.1 shows an example of both types of context diagrams. The total system is depicted as one process which communicates with its environment. The environment consists of external entities, so called terminators.

![Diagram](image)

**Figure 4.1** Example of a data and control context diagram

The single top level process, defining the complete system, is decomposed in a number of subprocesses. These subprocesses together constitute the first level of decomposition, which is described in terms of a DFD and a CFD. The flows entering or leaving the single top level process on the context diagrams must also enter or leave the flow diagrams of the first level of decomposition. New data and control flows appear on the DFD and CFD diagrams, which depict the flow of data and control between the subprocesses. The DFD and CFD diagrams can contain other objects besides processes. The objects diagrams may contain are given in the following overview:

**Context Diagram:**
- One process
- Terminators

**Data flow diagram:**
- Processes
- Data stores

**Control flow diagram:**
- Processes
- Control stores
- Control bars

23
A store is used to store data or control signals for later use. A data flow can transport data to a data store where it is held until it is overwritten with new data, the same is true with control flows and a control store. Control bars appear only on control flow diagrams and are used to depict the flow of control to and from a control specification (CSPEC). Control specifications represent the finite state machine behaviour of the system. The CSPEC describes how output control flows are generated from the input control flows. However, the main reason for a CSPEC is activating and deactivating processes. Processes can be (de)activated with process activators within the CSPEC. Process activators are not depicted on data flow or control flow diagrams.

The processes found at the lowest level of decomposition are called functional primitives and their behaviour is described in a process specification (PSPEC). The principle is that at some point there is little room left for ambiguity or confusion, and a narrative definition can be used to describe the process. Figure 4.2 shows the overall structure of the requirements model.

The processes are all assumed to be infinitely fast and triggered if all necessary data is

---

6 Not to be confused with service primitives.
available. Therefore, the requirements model describes an idealized system.

4.2.2 The CASE tool ProMod

ProMod is a CASE tool which has been used to realize the requirements model. The reason for choosing ProMod is, that it supports the Hatley and Pirbhai method for the most part. Promod supports three phases of system development:

Requirements Analysis and Definition
The aim of this phase is to create a structured model of the system which contains all technical requirements expressed in a consistent way.

System Design
This phase is to develop an architectural model, which describes how the system will meet the requirements defined in the first phase.

Program Design
This phase is to create the specifications in pseudocode for each function defined in the architectural model.

For realizing the requirements model only the requirements analysis and definition phase of ProMod has been used. This phase does not fully support the method according Hatley and Pirbhai. The discrepancies between ProMod and the Hatley and Pirbhai modelling technique are:

- Timing Specifications are not supported at all in ProMod.
- ProMod requires a PSPEC for every process even if it is not a functional primitive.
- Bidirectional flows can not merge or split.
- The definition of flows in the data dictionary is limited.

The last two restrictions are involved with modelling issues not explained in this thesis, for a full discussion of these issues the reader is referred to [HatI87].

A major advantage of ProMod is the fact that it checks consistency of the model. The model can be analyzed and ProMod will signal for example undefined flows. The designer is able to locate syntactical errors and remove them from the model. A syntactical correct design does of course not guarantee that the system is correctly specified. More about ProMod can be found in the user manual [Prom89], and in [Hurk92] on page 45 a list of program bugs can be found.

4.2.3 Modelling considerations

From the former paragraph it is clear that the requirements model made with ProMod can not be exactly according the method of Hatley and Pirbhai. In some cases a deviation of the Hatley and Pirbhai method is consciously made by reason of clearness. This paragraph is meant to explain how certain modelling aspects should be interpreted.
Process activation

In our case, all processes are defined active, unless their activation and deactivation is explicit stated in a CSPEC. A process is (de)activated with a process (de)activator, which is a statement in the CSPEC. The statement for activation is 'ACTIVATE <process name>' and the statement for deactivation is 'DEACTIVATE <process name>'. If a process is (de)activated, all its subprocesses are (de)activated.

A process which executes a set of actions not repeated within the process, will stop after doing the last action. The process only restarts if it is activated again. Therefore, such a process always has a process activator.

Communication between processes

Data or control information travelling between processes is depicted as flows. A flow is not, like in a real system, a channel which transports the information. It merely depicts that information is transported in some way. This implies that the receiving process can read the information any time as long as the sending process is not deactivated or new information is send.

Two processes can only communicate if both are active at the same time or if a store is used to hold the data or control flows after the sending process is deactivated. A case where a store has to be used is if two processes are active one at a time. An example of two alternating processes can be found in the requirements model of the PPM, which is decomposed in 'Connection Establishment' and 'Connected'. If the 'Connection establishment' process is active the 'Connected' process is not and vice versa. All information which has to be transported between these processes has to be stored.

According to Hatley and Pirbhai, processes are assumed to be infinitely fast so, if a process does not have to wait for information (all flows towards the process are defined) it takes no time to execute. A data flow provided by such a process can be used immediately after activating the process. In some cases this can lead to confusion. In these cases, although its not used in the Hatley and Pirbhai method, a status control flow is used to indicate to the CSPEC that the process is ready and the data can be used, see figure 4.3.

![Figure 4.3 The use of a status control flow](image-url)
Control Specifications

The control specifications represent the finite state machine behaviour so, the specification consists of one or more states. Dependent on the input control flows the CSPEC generates output control flows and possibly a transition to another state is made. If a process is activated its CSPEC has to start in some state, this will be the first state found in the CSPEC, which, in our model, is mostly called 'STA_idle' or 'STA_normal'.

In our case, the CSPECS are written in a pseudo programming language. Only a few statements and constructions are used, which will be explained with the help of an example, see figure 4.4.

```
STA_idle
   ACTIVATE 'Process 1'
   GOTO STA_wait

STA_wait
   IF >ready = 'ok'
       <read = 'yes'
       DEACTIVATE 'Process 1'
       GOTO STA_terminate
   ELSEIF >ready = 'error'
       <read = 'no'
       DEACTIVATE 'Process 1'
       GOTO STA_idle
   ENDIF

STA_terminate
   // Do nothing and
   // wait for deactivation
```

FIGURE 4.4 An example of a CSPEC

The example of figure 4.4 could be the CSPEC describing the behaviour of the system shown in figure 4.3. The first state is 'STA_idle' and on activation of the system it will be in this state. This state only activates process 1 and the system will make a state transition to state 'STA_wait'. In this state two conditions are checked, if input (indicated with a > sign) 'ready' has the value 'ok' or if the input 'ready' has the value 'error'. If the 'ready' input has some other value, nothing happens and the system will wait for one of the conditions to become true.

If the input 'ready' has the value 'ok', the output (indicated with a < sign) 'read' is set to 'yes' and after that, process 1 will be deactivated. The deactivation of process 1 can not be done before the checking of the 'ready' input, because in that case the input would be undefined. Finally, a state transition to state 'STA_terminate' is made. The state 'STA_terminate' has no statements so it does nothing, and the only way of leaving this state
is by deactivating the system and then activating the system again.

The observant reader will have noticed from figure 4.3 that process 2 uses data provided by process 1 and process 1 is deactivated after process 2 is told to read the data. As processes are assumed indefinite fast it is not clear what happens first, reading the data or deactivating process 1. In this example it is clear what is meant: that the data is first read before process 1 is deactivated. In some cases it is not so clear and therefore, a status control flow is used to prevent for this sort of situations.

4.3 The Requirements model of the presentation layer

This paragraph will discuss the first levels of the requirements model of the presentation layer. These levels are: the context diagram, the first functional decomposition, and the PPM. The context diagram and the first functional decomposition are involved with the complete presentation layer. A separation is made between the part of the requirements model this thesis is specially concerned with, and the part discussed in [Reut92].

4.3.1 The context of the presentation layer

The context diagram of the presentation layer is depicted in figure 4.5.

![Figure 4.5 The presentation layer context diagram](image)

The presentation layer interfaces with the application and session layer, so these layers are the terminators on the context diagram. The control flows on the control context diagram transport the service primitives. The p_ici (presentation interface control information) control flow carries the presentation service primitives. These service primitives are used for
communication between the application layer and the presentation layer. The s_ici (session interface control information) control flow carries the session service primitives for communication between the presentation and session layer.

The parameters associated with the service primitives are transported in the corresponding data flows, depicted on the data context diagram. The data flows are the presentation service data unit (p_sdu) and the session service data unit (s_sdu). The p_sdu transports the parameters associated with presentation service primitives and the s_sdu the parameters associated with the session service primitives.

A complete list of presentation and session service primitives can be found in appendix A. The exact description of the data and control flows can be found in the requirements model in appendix B.

4.3.2 The first level of decomposition

The single presentation layer process in figure 4.5 is decomposed in a data flow diagram and an associated control flow diagram. For a discussion of this first level of decomposition, the data flow diagram of Figure 4.6 can best be used.

![Diagram of the presentation layer](image)

**Figure 4.6** Data flow diagram of the presentation layer
Figure 4.6 shows that the single presentation layer process on the context diagram is decomposed in six subprocesses:

- Psdu_buffer
- Ssdu_buffer
- Direction_arbiter
- Encoder_decoder
- PPM
- Filter_dcs

The data flows 'p_sdu_e' and 's_sdu_e' carry the same information as the data flows 'p_sdu' and 's_sdu' in figure 4.5. In figure 4.5 the 'p_sdu' and 's_sdu' data flows where depicted twice as an unidirectional flow, in figure 4.6 the 'p_sdu_e' and 's_sdu_e' data flows are depicted once as bidirectional flows. The 'e' in the 'p_sdu_e' and 's_sdu_e' data flows stands for 'external' and indicates that these flows are flowing from and to the environment of the presentation layer.

One additional data flow appears in figure 4.6, the 'update_micro_programs' data flow. This flow carries the micro-programs to the 'encoder_decoder' process. For reason of simplicity, this flow is not depicted in the context diagram of figure 4.5. The renaming of the 'p_sdu_e' and 's_sdu_e' data flows is done for the same reason. The processes on figure 4.6 will be described below.

The psdu and ssdu buffer processes

The interface of the presentation layer with its adjacent layers is formed through the psdu_buffer and the ssdu_buffer processes. These buffers accept service primitives and their associated parameters. The service primitives are transported in control flows, which are not visible in the depicted DFD, and the parameters are transported in the 'sdu' data flows (p_sdu_e, p_sdu, s_sdu and s_sdu_e). Besides buffering the data, data checking and error control are performed in these processes. If a service primitive is not recognized or the associated parameters are incorrect, an abort service primitive is send to the PPM. For a full discussion of the both buffer processes, the reader is referred to [Reut92] chapter 7 and appendix A.

The direction arbiter process

The requirements model of the presentation layer describes an uni-directional system, this is according to the CCITT recommendations. This means that a presentation layer entity is able to process only one service primitive at a time. If the application layer issues a presentation service primitive and the session layer issues a session service primitive to the same presentation layer entity at the same time, one of these service primitives together with its parameters has to be stored for later processing. The previously mentioned buffer processes store the information. The direction arbiter process controls which of the received service primitives will be processed.

If a buffer receives a service primitive it stores the service primitive and the associated parameters and requests the usage of the presentation layer entity from the direction arbiter.
The direction arbiter will signal the buffer to forward the primitive and the associated parameters if the presentation layer entity is ready to process the request for service. The direction arbiter is fully explained in [Reut92] chapter 7 and appendix A.

**The encoder_decoder process**

This process performs the actual encoding and decoding of data. The data send by the application layer (user data) is encoded according to the DCS or the default context so the process has to read the contents of the defined_cs and default_cs stores. The control information added by the presentation PPM for peer-to-peer communication will always (in the connected as well as in the connection establishment phase) be encoded using the Basic Encoding Rules (BER).

An other important function of the encoder_decoder process is to fill the supp_cs store with the supported abstract syntaxes and transfer syntaxes. The encoder_decoder process can only encode or decode data if it is according to a presentation context consisting of an abstract syntax and transfer syntax which are stored in the supp_cs store. This store is filled before the presentation layer tries to set up a connection.

The encoder_decoder process is very complex and for a full discussion the reader is again referred to [Reut92] chapter 7 and appendix A.

**The PPM and filter_dcs processes**

These two processes together perform the function which is in recommendation [X.226] described as the function to be performed by the PPM. The filter_dcs process is in fact the part of the PPM concerned with managing the DCS. Representing this functionality of the PPM in an individual process is recommended in [Cane87] and [Garb88].

The PPM process provides the presentation entity with the peer-to-peer communication protocols and controls the filter_dcs process. In chapter 5 the PPM process will be further decomposed.

The filter_dcs process keeps the defined_cs store up to data with the DCS. In chapter 3 it was mentioned that the DCS can change during a connection due to context alterations, resynchronization and activity switching. The filter_dcs process has to remember the DCS at certain synchronization points and for certain activities, to be able to restore the DCS to the contents defined earlier. For keeping track of the history of the DCS, the manage_dcs store is used. This store and the filter_dcs process will be discussed further in chapter 6.

All processes depicted in figure 4.6 are now described so the interaction of the processes can be discussed. In the following discussion the processes are mentioned by name only, so the term 'process' is omitted.

If the application layer sends a presentation service primitive with its associated parameters, the psdu_buffer will store the service primitive together with the parameters. The psdu_buffer will place a request to the direction_arbiter. If the requests is granted the service primitive
together with the parameters are send to the PPM. If the requested service has some impact on the DCS, the PPM will signal the filter_dcs to take the necessary actions. The PPM will construct a PPDU for peer-to-peer communication or, if the requested service is a session service where the presentation layer only acts as a bridge, it will generate a session service primitive with associated parameters. The PPDU or service primitive together with the parameters are send to the encoder_decoder. The PPM signals the encoder_decoder through the filter_dcs that the data can be encoded. The encoder_decoder first checks if the user data is according to the DCS or default context and an error is generated if not. If no error occurs the data is encoded. If the PPM has send a PPDU, the encoder_decoder will generate a session service primitive with associated parameters consisting of the encoded parameters of the PPDU. The encoded data is send to the ssdu_buffer, which will send it to the session layer.

If the session layer sends a session service primitive with its associated parameters, the ssdu_buffer will store the service primitive together with the parameters. A requests is placed by the ssdu_buffer to the direction_arbiter, and if it is granted the service primitive together with the parameters are send to the encoder_decoder. The encoder_decoder first decodes the control information parameters encoded using the BER. These parameters are send to the PPM which then knows what action has to be taken. In some cases the DCS has to be altered before the user data can be decoded. If this is the case the PPM will signal the filter_dcs to take the necessary actions. Next, the PPM will signal the encoder_decoder to decode the user data. The encoder_decoder first checks if the user data is according to the DCS or the default context and an error is generated if not. If the data is according to the DCS or default context it is decoded and send to the PPM. The PPM will send a presentation service primitive together with the associated parameters to the psdu_buffer, which will send it to the application layer.

A full description of the processes and flows can be found in appendix B, where also the corresponding control flow diagram can be found.

4.4 Summary

This chapter discussed the chosen modelling technique, which is the method according to Hatley and Pirbhai. The requirements model has been introduced, which describes how a system should work in a formal way. The CASE-tool ProMod has been discussed, which is used to realize the requirements model of the presentation layer. This tool made it necessary to have some deviations from the method specified by Hatley and Pirbhai. These deviations, which sometimes were consciously made to clarify the model, were explained.

The requirements model of the presentation layer has been introduced. The context diagram and the first level of decomposition were described. The functionality of the PPM as it is described in the recommendations is divided in two processes, the PPM process and the filter_dcs process. The PPM process is mainly concerned with the peer-to-peer communication protocols and the filter_dcs process is concerned with managing the DCS. Both processes will be further discussed in the following chapters.
Chapter 5

The Presentation Protocol Machine

In recommendation [X.226] the function of the PPM is described with the help of state tables. These state tables are made up of predicates, actions and states. The actions and predicates concerning the DCS are dealt with by the filter_dcs process, which will be discussed in the next chapter. This chapter describes the decomposition of the PPM process. The state tables according to which the PPM process is designed can be found in [X.226].

5.1 The decomposition of the PPM

The decomposition of the PPM process is depicted in figure 5.1.

Figure 5.1 Data flow diagram of the PPM

In figure 5.1 four processes appear. Two of these processes are always active, these are the fill_error_type and check_user_data processes. These processes have a simple task which will be explained below.

Fill_error_type

The process only copies the contents of the error_type flow to the error_type store. The
reason for not connecting the data flow directly to the store is that ProMod does not support this option.

If somewhere in the presentation layer entity an error occurs, a P_abort.indication service primitive is generated by the PPM. One of the associated parameters of this service primitive is 'provider reason', which contains the reason for the abort. The contents of this parameter are read from the error_type store. An error can occur in the PPM process during the connection establishment or the connected phase, or in the encoder_decoder process. Therefore, three processes (one at a time) are able to fill the error_type store, as depicted in figure 5.1. The P_abort.indication service primitive is generated by the PPM. As the PPM can be in the connection establishment or the connected phase the error_type store can be read by the processes implementing both phases.

Check_user_data

This process checks if the user data parameter is present with a presentation service primitive (issued by the application layer) or session service primitive (issued by the session layer). Checking for the user data parameter associated with a presentation service primitive is simply done by checking the p_sdu flow for this parameter. Checking if this parameter is send with a session service primitive can not be done by the PPM itself. The encoder_decoder will only send the user data parameter to the PPM after it has been decoded, but the PPM has to know if user data is present before it will be decoded. Therefore, the encoder_decoder will signal the check_user_data process if user data is present. The PPM has to know if user data is present before the data is encoded or decoded because in some cases the DCS will change if user data is present and the user data has to be encoded or decoded according to the new DCS.

The most important processes depicted in figure 5.1 are: connection_establishment and connected. Only one of these two processes is active at a time. As soon as a connection has been established the connection_establishment process is deactivated and the connected process is activated. If a connection is released the connected process is deactivated and the connection_establishment process is activated. Decomposing the PPM this way is obvious, because the state tables in recommendation [X.226] define a special state table for the connection establishment phase. In the following paragraphs these two processes will be further discussed.

5.2 Connection establishment

This process is only concerned with establishing a presentation connection and it is activated as soon as the presentation layer is not in the connected phase. The process stays in an idle state until it receives a request for setting up a connection. During the connection establishment phase only a limited set of service primitives and PPDU's are used. The following table gives an overview. The PPDU's, used for peer-to-peer communication, are

---

7 In the psdu and ssdu buffers also an error can occur, but these processes will generate an abort themselves, without using the error_type store.
shown after the service primitive, which they are associated with.

**Table 5.1: Service primitives and PPDU’s used during connection establishment**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>PPDU</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_connect</td>
<td>CP</td>
<td>Presentation connection establishment</td>
</tr>
<tr>
<td></td>
<td>CPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPR</td>
<td></td>
</tr>
<tr>
<td>P_P_Abort</td>
<td>ARP</td>
<td>Abnormal release, Provider initiated</td>
</tr>
<tr>
<td>P_U_Abort</td>
<td>ARU</td>
<td>Abnormal release, user initiated</td>
</tr>
<tr>
<td>S_P_Abort</td>
<td>-</td>
<td>Abnormal release of session layer</td>
</tr>
</tbody>
</table>

The decomposition of the connection establishment process is depicted in figure 5.2. In this figure seven processes are defined, which will be discussed in the following paragraphs.

![Data flow diagram of connection establishment](image)

**Figure 5.2** Data flow diagram of connection establishment

Figure 5.2 only shows the data flow diagram so control flows and control bars are not visible. In the following paragraphs sometimes control flows are mentioned. For the control flow diagrams and the full definition of processes and flows the reader is referred to appendix B.
5.2.1 The connection pdu assembler

From the data flow diagram of figure 5.2 it can be seen that this process has many data flows to the other processes. The process receives and sends all the parameters associated with the service primitives or PPDUs represented in table 5.1, except for the abort services which are dealt with in the terminate_establishment process. The other processes can retrieve the necessary parameters from this process or send the parameters for a PDU or presentation service primitive to this process.

The process reads the 'p_sdu' or 's_sdu' flow if a presentation service primitive respectively a PDU arrives. After the parameters are read the other processes can retrieve them. If a presentation service primitive or a PDU has to be send, the CSPEC of the connection establishment process has to notify the connection_pdu_assembler of this. The CSPEC sends the type of PDU or presentation service primitive and signals the connection_pdu_assembler to send the associated parameters in the 's_sdu' respectively the 'p_sdu' flow.

The connection_pdu_assembler process is one of a complete set of processes all concerned with the receiving and sending of PDU or service primitive associated parameters. All these processes act in the same way, only the types of PDU's and service primitives and therefore the parameters they process are different.

5.2.2 The processes performing negotiation

The following processes perform some sort of negotiation during the connection establishment phase: context_definition, requirements and mode_protocol.

context_definition

This process negotiates about the DCS and the default context as described in paragraph 3.2.2. The data flows in figure 5.2 exactly depict the various parameters, which are used for default context and DCS negotiation. The supp_cs store is read for supported abstract syntaxes and transfer syntaxes. The resulting default context is stored in the default_cs store depicted in figure 4.6 and the resulting DCS is send to the filter_dcs process.

requirements

This process negotiates about the functional units as described in paragraph 3.3. The process checks if the requested presentation functional units are supported. The presentation functional units it is able to support, are stored in the fu_domain store depicted in figure 4.6. Furthermore, it checks if the functional units requested for the session layer support the presentation requirements, if not it adds the necessary session functional units to the requested session requirements.

mode_protocol

This process performs negotiation about the protocol which will be used by the PPM and it
handles the setting of the mode in which the PPM will operate. The presentation protocol version negotiation is explained in paragraph 3.3. This process will store the chosen protocol version through the 'protocol_ vers' data flow in the protocol_ vers store depicted in figure 5.1.

The mode in which the PPM will operate is determined by the initiating application layer entity and is one of 'normal mode' or 'X.410 mode'. If the PPM operates in the 'X.410 mode' no DCS, default context or functional units will be negotiated, all user data is assumed to be of the same presentation context. This mode is defined for supporting the earlier defined CCITT recommendation X.410. The mode_protocol process will store the chosen mode in the same named store depicted in figure 4.6. The used mode has a major influence on the behaviour of the encoder_decoder process, for a detailed discussion the reader is referred to [Reut92] chapter 7.

5.2.3 Addressing

In paragraph 2.2.2 it was already mentioned that one end-system can have several connections with other end-systems and addressing is used to identify one of these connections. The addressing process handles all addressing tasks concerned with the session SAP’s and the presentation SAP’s. Two data flows transport the necessary information to and from the process, these are the 'address' and 'selector' data flows.

When a request for a presentation connection is received from the application layer entity, two parameters are used for addressing: the calling-presentation-address and the called-presentation address, see figure 5.3.

![Diagram of Addressing Process]

**Figure 5.3** The function of the addressing process

A presentation-address consists of an address identifying the SSAP (session-address) and a set of addresses identifying the PSAP’s (presentation-selector). The presentation-address identifies both the SSAP and the set of PSAP’s used for this connection. The calling-
presentation-address identifies the SAP's in the initiating end-system, this can be seen as the address of the sender. The called-presentation-address identifies the SAP's in the responding end-system, which can be seen as the addressee. The presentation service primitive contains the presentation-addresses, whereas the PPDU's contain the presentation selectors and the session addresses. Therefore, the addressing process has to divide the presentation-addresses into the session-addresses and the presentation-selectors if a presentation service primitive is received, and it has to construct the presentation-addresses from the session-addresses and presentation-selectors if a PPDU is received. Figure 5.3 depicts the dividing and constructing of these parameters.

Together with the response service primitive, the responding-presentation-address is send. This presentation-address identifies the SAP's that should be used to re-establish a presentation connection if the requested connection is rejected because the SAP's identified by the called-presentation-address could not be used. For a detailed discussion of the addressing mechanism, the reader is referred to [ISO.7498].

5.2.4 Error management

In the connected phase an error always leads to sending abort service primitives and the release of the connection. In the connection establishment phase a connection is not established yet so a second possibility exists, which is rejecting the request for a connection by sending a CPR PPDU to the initiating PPM.

In the connection establishment phase several errors can occur, some of them will lead to the sending of abort service primitives and others will lead to rejecting the requested presentation connection. The error_management process decides if an abort or a rejection will take place. The process also fills the error_type store with the correct error message in case of an abort and provides the 'provider_reason' parameter for the CPR PPDU in case of a rejection of the presentation connection.

5.2.5 Terminate establishment

The terminate_establishment process is concerned with all abort services during the connection establishment phase. The process handles service primitives and PPDU's and if an error occurs somewhere in the presentation layer which leads to an abort this process will initiate an abort procedure. The service primitives and associated PPDU's used for peer-to-peer communication can be found in table 5.1.

An error leading to an abort can also occur in the connection_pdu_assembler process, discussed in paragraph 5.2.1. Therefore, the process receives and sends the parameters associated the abort services itself, so in the further decomposition of this process, which can be found in appendix B, a pdu_assembler process as described in paragraph 5.2.1 appears.

An abort service initiated by a presentation user can transport user data which has to be encoded according to a presentation context known by the initiator as well as the responder.
The problem which occurs is that the DCS of the initiator and responder might not be the same, because negotiation about the DCS is not finished yet. In paragraph 3.4.7 this problem and, as a solution to it, the use of a presentation context identifier list are discussed. The terminate_establishment process of the initiating PPM has to send this list as a parameter of the ARU PPDU, therefore the data flow 'filter' is used to retrieve this list from the filter_dcs process. The terminate_establishment process of the responding PPM has to update the DCS according to this list, but it only includes the supported presentation contexts in the DCS. The process uses the 'supp_cs' data flow to check the presentation context identifier list for not supported presentation contexts. Only the supported presentation contexts are send to the filter_dcs process, through the 'filter' data flow, to update the DCS.

5.3 Connected

Once a connection has been established, the connection establishment process will be deactivated and the connected process activated. During the connected phase all service primitives and PPDU’s represented in the tables of appendix A can be used, except for the connection establishment service primitives and associated PPDU’s.

5.3.1 The decomposition of the connected process

Figure 5.4 depicts the data flow diagram of the decomposed connected process. The associated control flow diagram and the further decomposition can be found in appendix B.
The connected process has a lot more functionality than the connection establishment process, but decomposing the connected process is a lot easier. In recommendation [X.226] several state tables are defined, each defining one service in the connected phase. By decomposing the connected process in processes exactly covering these services, each process is defined by one of these state tables. All processes are defined active in the connected phase but only act if their service is needed. The presentation entity performs one service at a time, so only one process at a time is performing some action. As can be seen from the data flow diagram of figure 5.4 (and the control flow diagram in appendix B) the processes function almost independent from each other, there is no flow of data or control between the processes.

The fact that there is no flow of data between the processes implies that each process has to handle its own service primitives and associated parameters and PPDU’s for peer-to-peer communication. In the further decomposition of each of these processes, which can be found in appendix B, a pdu_assembler process, as described in paragraph 5.2.1, appears.

In figure 5.4 the context_phase store is depicted in a central position and is in fact used by all processes. This store represents the phase in which the PPM operates, which can be the data transfer phase (t0 phase) or an alter context phase (ac0, ac1 or ac2 phase). The meaning of these phases will be explained in the following paragraph. The phase in which the PPM operates has much effect on the functioning of the processes. The state tables define the behaviour of the processes for each of these four phases, which are represented in the state tables as states.

The different services available in the connected phase have been discussed in paragraph 3.4 and as each of the processes implements one of these services, no further description of the tasks of the processes is needed here. For the decomposition of each of the processes the reader is referred to appendix B.

5.3.2 Context management

The context_management process acts as described in paragraph 3.4.6, but because of the fact that the process has much impact on the behaviour of the other processes, some special attention is focused on it.

The connected phase can be split up in a data transfer phase and three alter context phases. The PPM operates in the data transfer phase (t0 phase) until one of the end systems issues a request for altering the DCS. If the PPM of the initiating end system receives the request, the context_management process will handle the request. The process will start negotiation by sending an AC PPDU to its peer with the proposed DCS alterations and the PPM no longer operates in the data transfer phase. The PPM now awaits an ACA PPDU from its peer and the phase it operates in is the ac0 phase. In the ac0 phase two things can happen: the PPM receives the expected ACA PPDU and the negotiation ends so the PPM operates again in the data transfer phase, or the PPM receives an AC PPDU. The PPM can receive an AC PPDU from the peer after sending one itself, if both end systems issued a request for altering the DCS at the same time.
If only one end system initiated DCS negotiation the initiating PPM will operate for a while in the ac0 phase until an ACA PPDU is received, which will bring the PPM back in the t0 phase. The responding PPM will operate in the ac1 phase in the time between sending a P_Alter_context.indication service primitive to the application entity and receiving its response in the P_Alter_context.response service primitive.

If both end systems initiated DCS negotiation the situation becomes more complex and a third alter context phase will become active. In the ac2 phase the PPM awaits an ACA PPDU from the peer, containing the response to its own "local" proposed alterations, or a P_Alter_context.response service primitive from the application entity, with the response to the "remote" proposed DCS alterations of the peer. Figure 5.5 depicts two simultaneous DCS alterations. In the figure the four phases are shown and the service primitives concerned with the same DCS negotiation are depicted in the same font.

![Figure 5.5 Two simultaneous DCS alterations](image)

The figure depicts only one possible situation which can occur if two DCS alterations are in progress. The sequence of phases in which the PPM operates changes if, for example, in the ac2 phase the ACA PPDU is received before the P_Alter_context.response service primitive. If two simultaneous DCS alterations occur, the context management process will handle them independently.
5.4 Summary

In this chapter the PPM, as it is described in the recommendations, has been divided in two processes: the PPM and the filter_dcs process. The reason for this is that the PPM process can now be designed according to the state tables defined in the recommendation [X.226], without being concerned with extra overhead. The PPM process is further decomposed in two major processes: the connection establishment and the connected process. These are alternating processes, so only one of them is active at a time. Both processes are further decomposed and discussed. The next chapter will discuss the filter_dcs process.
Chapter 6
The Filter

The filter_dcs process performs a part of the functionality of the PPM as defined in recommendation [X.226]. The process is concerned with managing the DCS and is controlled by the PPM process discussed in the previous chapter. The first paragraph will discuss the so-called filter commands. The process itself and its decomposition processes will be discussed in the second paragraph.

6.1 The filter commands

The PPM process controls the filter_dcs process by sending commands. The complete set of commands is divided in five groups of related commands. In the following paragraphs each group of commands will be described. The commands are send to the filter_dcs in the control flow 'command' and the associated parameters are send in the data flow 'filter'. These flows can be found in the control flow diagram respectively the data flow diagram of the filter_dcs process depicted in appendix B. In the description of the commands the parameters transported by the 'filter' data flow are also mentioned.

In some commands the words 'local' and 'remote' appear. Something is defined 'local' if it results from an action initiated at the same side of the connection, and 'remote' if the peer is responsible for it.

6.1.1 Alter manage store commands

The filter_dcs process uses the manage_dcs store, depicted in figure 4.6, for storing the complete history of the DCS. How the history is stored will be discussed in paragraph 6.3. The commands in this paragraph have in common that they alter the contents of the manage_dcs store.

ADD_DEF_LIST_TO_MANAGE_STORE
The presentation context definition list is stored in manage_cs and defined as the connection establishment DCS. This command is only used during connection establishment. Filter contains the modified presentation context definition list

CONSOLIDATE_LOCAL_ACCEPTED_CONTEXTS
Mark all approved local additions as active and mark all approved local deletions as deleted in the current DCS. A context addition or deletion is called approved if the contents of the corresponding item in the result list is 'acceptance'. Next, the defined_cs store is filled with the resulting context set. Filter contains the presentation context addition result list and the presentation context deletion result list.
CONSOLIDATE_REMOTE_ACCEPTED_CONTEXTS
Mark all approved remote additions as active and mark all approved remote deletions as deleted in the current DCS. A context addition or deletion is called approved if the contents of the corresponding item in the result list is ‘acceptance’. Next, the defined_cs store is filled with the resulting context set. Filter contains the presentation context addition result list and the presentation context deletion result list.

INITIALIZE_FILTER
Clears all stores inside the filter and clears the stores manage_cs and defined_cs.

SET_DCS_TO_ID_LIST
The current DCS is set according to the presentation context identifier list. The new DCS is stored in manage_dcs and in defined_cs. Filter contains the presentation context identifier list.

SET_DCS_TO_INTERACTIVE
The current DCS is set to the inter-activity DCS. The new DCS is stored in manage_dcs and in defined_cs.

UPDATE_FILTER_WITH_RESULT_LIST
All contexts which are marked as rejected in the presentation context definition result list will be deleted from the context set contained in the manage_dcs store. From the resulting contexts, the transfer syntax will be set to the one accepted in the result list. The remaining context set is still marked as the connection establishment DCS. This command is only used during connection establishment. Filter contains the presentation context definition result list.

UPDATE_LOCAL_PROPOSED_CONTEXTS
Add the presentation context addition list to the store manage_cs and mark them as local proposed contexts. Mark the contexts which are identified by the presentation context deletion list as local proposed for deletion. Filter contains the presentation context addition list and the presentation context deletion list.

UPDATE_REMOTE_PROPOSED_CONTEXTS
Add the presentation context addition list to the store manage_cs and mark them as remote proposed contexts. Mark the contexts which are identified by the presentation context deletion list as remote proposed for deletion. Filter contains the presentation context addition list and the presentation context deletion list.

6.1.2 Fill DCS store commands
The defined_cs store, depicted in figure 4.6, is used by the filter_dcs process to store the currently defined DCS. The commands discussed in this paragraph fill the defined_cs store with a set of presentation contexts forming the currently defined context set.

FILL_DCS_WITH_CON_EST_CONTEXTS
Fill the store defined_cs with the DCS as was agreed during connection establishment.

**FILL_DCS_WITH_RESULT_LIST**
The contexts in the manage_dcs store which are not marked as rejected in the presentation context definition result list are stored in the defined_cs store. This command is only used during connection establishment. *Filter contains the presentation context definition result list.*

### 6.1.3 Data checking commands

**CHECK_SYNTAX_AGAINST_FILTER**
Check if each presentation data value is from presentation contexts contained in the presentation context identifier list. This command does not alter the DCS. *Filter contains the presentation context identifier list.*

**CHK_DATA**
Check if each presentation data value is from presentation contexts of the DCS, or from the default context if the DCS is empty.

**P06**
Check if each presentation data value is from presentation contexts of the DCS not proposed for deletion from the DCS by the peer PPM.

**P07**
Check if each presentation data value is from presentation contexts of the DCS not proposed for deletion from the DCS by the local PPM.

**P13**
Check if each presentation data value is from the default context set.

**P24**
Check if each presentation data value is from presentation contexts of the DCS, or from presentation contexts proposed for addition to the DCS by the local PPM, or from the default context if either the DCS is empty or all presentation contexts of the DCS were proposed for deletion by the local PPM.

**P25**
Check if each presentation data value is from presentation contexts of the DCS not proposed for deletion by the peer PPM or from presentation contexts proposed for addition to the DCS by the local PPM.

### 6.1.4 Context set retrieve commands

**GET_DCS_ID_LIST**
Retrieves the presentation context identifier list of the current DCS. *Filter contains the*
presentation context identifier list.

6.1.5 Synchronization and activity commands

ASSOCIATE_SYNC
Add a synchronization point to the administration of the current DCS. Filter contains the synchronization point serial number.

CHECK_ACTIVITY_ASSOCIATED
Check if a DCS is associated with the pair of old activity identifier and synchronization point serial number. Filter contains the old activity identifier and the synchronization point serial number.

DEASSOCIATE_SYNC_WITH_ACTIVITY
Eliminate any associations between the synchronization point serial number and the DCS for the current activity or, if no activity is in effect, eliminate any associations between synchronization point serial numbers of the last activity and their DCS.

ELIMINATE_AND_ASSOCIATE_SYNC
Eliminate all synchronization points set earlier and replace them with one new synchronization point serial number. Filter contains the synchronization point serial number.

IS_SYNC_NOT_ASSOCIATED
Check if either no syncpoint identifier is associated with a DCS or the resynchronization identifier is not associated with a DCS and is greater than the lowest synchronization identifier which has an associated DCS. Filter contains the synchronization point serial number.

NEW_ACTIVITY
Set up an administration for a new activity. All new synchronization points will be linked to this activity and all context set alterations act on this activity. Filter contains the activity identifier.

REMEMBER_DCS_AS_INTERACTIVE
The current DCS is marked to be the inter activity DCS.

RESUME_ACTIVITY
The activity identified by the old activity identifier is reactivated and the identifier is replaced with the activity identifier. The DCS is restored to the one associated with the synchronization point serial number. Next, the new DCS is stored in defined_cs. Filter contains: old activity identifier, activity identifier and synchronization point serial number.

RESYNCHRONIZE_SYNC_POINT
The DCS is restored to the synchronization point serial number, if this number is
known to the filter. If the synchronization point serial number is less than the numbers known to the filter, the DCS is restored to the one at connection establishment. In all other cases the DCS remains unchanged and syntax_status returns 'resync_fail'. The new DCS is stored in manage_dcs and defined_cs. Filter contains the synchronization point serial number.

6.2 The decomposition of the filter

The filter_dcs process is not explicit defined in state tables, as the PPM process is. The filter_dcs process represents the actions and predicates stated in these state tables. The decomposition of the process is therefore not so obvious as it was with the connected process discussed in the previous chapter. The basic idea behind the decomposition of the process found in figure 6.1 is that three processes are concerned with adding all DCS alterations to the history of the DCS kept in the manage_dcs store and one process fills the defined_cs store with the currently defined DCS. The currently defined DCS is pointed to with help of three stores inside the filter_dcs process.

The three stores depicted in figure 6.1 are used to remember the currently defined DCS. The operation_mode store indicates if the DCS is:

- The one defined at connection establishment
- The inter-activity DCS
- A DCS associated with an activity

Figure 6.1 Data flow diagram of the filter
The activity_number store contains the number of the current or last activity and the sync_number store contains the synchronization point serial number. The combination of these three stores offers enough information to identify exactly one DCS as it is defined in the manage_dcs store. These stores are always used if a process has to retrieve the current DCS.

In figure 6.1 four processes are depicted, which will be explained below:

**Get_set_cs**

This process is used for reading or sending presentation contexts lists from or to the 'filter' flow. The presentation contexts lists which are handled are the:

- Presentation context definition list
- Presentation context definition result list
- Presentation context addition list
- Presentation context deletion list
- Presentation context addition result list
- Presentation context deletion result list
- Presentation context identifier list

All these lists, except for the last one, can only be read from the 'filter' flow and are used to update the manage_dcs store. The presentation context identifier list is used for several purposes. On receipt of the 'check_syntax_against_filter' command the presentation context identifier list is used to directly fill the defined_cs store, therefore the flow 'ident_list' is used. The 'get_dcs_id_list' command is used to retrieve the current DCS in the form of this list and in this case the list is send in the 'filter' flow. The process knows what the current defined DCS is by reading the three stores depicted in figure 6.1.

**Synchronization_manager**

The synchronization_manager process is concerned with associating minor and major synchronization point serial numbers to the current DCS, checking if a DCS is associated with a synchronization point serial number contained in the 'filter' flow and resynchronization, which implies that the DCS is restored to the one associated with the synchronization point serial number contained in the 'filter' flow.

**Activity_manager**

The activity_manager process is concerned with setting up an administration for a new activity, checking and possibly restoring the DCS to the one associated with an activity number, defining the current DCS as the interactivity DCS, and deassociating all synchronization point serial numbers with the current or last activity.

---

8 How a DCS is stored in the manage_dcs store is explained in the following paragraph.
fill_dcs_store

This process fills the defined_cs store with a DCS. The DCS can be the one defined by the presentation context identifier list, an empty DCS, or the DCS identified by the three stores depicted in figure 6.1. In the latter case the process has, depending on the filter command, to regard that sometimes not all the presentation contexts contained in the identified DCS must be stored in the defined_cs store. An example of such a command is 'P06', where the currently defined DCS without the presentation context proposed for deletion by the peer PPM has to be stored in the defined_cs store.

The filter commands discussed in paragraph 6.1 are in most cases not handled by only one process. The CSPEC of the filter_dcs process describes the actions each process has to take and the sequence in which these actions should occur. An example of a filter command where two processes are concerned with is 'resume_activity'. The activity_manager process will alter the contents of the three stores depicted in figure 6.1 and the fill_dcs_store process will fill the defined_cs store with the DCS identified by the new contents of these stores.

For the complete decomposition of the filter_dcs the reader is referred to the requirements model in appendix B. The next chapter will discuss the contents of the manage_dcs store.

6.3 The DCS administration

In the former paragraphs is was already mentioned that the filter_dcs process uses the manage_dcs store for remembering the history of the DCS. The data transfer during a presentation connection is organized in units with help of activity and synchronization management. The DCS administration therefore, has to remember the DCS as it was during the transfer of these units. An example of how the manage_dcs store could be organized to administrate the history of the DCS is shown in figures 6.2 and 6.3.

![Figure 6.2 The DCS administration](image)

Figure 6.2 shows the first step in a possible organization of DCS management. All presentation contexts and activities used during a connection will be remembered. Within an activity all used presentation contexts during that activity will be remembered by storing the
presentation context identifiers to the administration of that activity\(^9\) (figure 6.2 shows the identification of presentation contexts for activity 2). As all used presentation contexts within an activity are remembered, a distinction has to be made between the currently used and the ones not in use.

Figure 6.3 The DCS administration in more detail

Figure 6.3 shows the DCS administration for activity 1 in more detail, the fields are filled in with devised values for means of illustration. As can be seen, the activity number, presentation context identifier, and a field for indicating if a presentation context is currently active are available in the DCS administration of an activity. Two fields are used for indication if the presentation context is proposed for addition or deletion, one for local and one for remote propositions. One field is used for the list of synchronization point serial numbers. If a synchronization point is set, the synchronization point serial number is added to each list belonging to active presentation contexts. Restoring the DCS to the one defined at a certain synchronization point can be easily done by making the presentation contexts active which have the synchronization point serial number stored in their list of synchronization point serial numbers.

Figure 6.3 also shows a fictitious set of presentation contexts. The presentation context identifier, the abstract syntax name and the transfer syntax name are shown. One presentation

\(^9\) One special administration exists, which is the 'no activity' administration. Except for the activity number, it is the same as an activity administration.
context consists of an abstract syntax name and a list of transfer syntax names, this is the case if the presentation context is still negotiated about. The presentation contexts belonging to the inter-activity DCS and the ones belonging to the DCS defined at connection establishment are indicated with two extra fields. As the inter-activity as well as the connection establishment DCS are defined on a global basis, they can be indicated in the set of presentation contexts.

6.4 Summary

This chapter discussed the filter_dcs process, which performs the part of the functionality of the PPM concerned with managing the DCS. The PPM process controls the filter_dcs process by sending commands. The set of commands is divided in five groups of related commands, which were discussed in the first paragraph. The filter_dcs process is decomposed in four processes which were discussed in the second paragraph. The history of the DCS is kept in the manage_dcs store outside the filter_dcs process. A possible organization of the contents of this store has been discussed in the third paragraph.
Chapter 7

Conclusions and Recommendations

7.1 Conclusions

The requirements model of the presentation layer has been extended and improved. The filter.dcs process is now completely defined and it is decomposed in its subprocesses. Furthermore, a possible memory management organization for the DCS administration has been introduced. The PPM process has been extended and details are worked out. Especially a lot of work has been done improving the error management and connection release processes. All possible errors (including the ones due to a hardware failure) are handled. If user data can not be decoded by the receiving presentation entity this is handled as an error. Therefore, data send during the connection establishment phase seems to have a great probability of generating an error, because the initiating presentation entity is not aware of the supported presentation contexts of the receiving presentation entity. In fact, the uncertainty about the supported presentation contexts is not as high as at seems. An application, in a normal case, will not randomly try to communicate with other applications, but tries to connect to the same application on an other end-system [Rama88], so the supported abstract syntaxes will be known to the initiating application and the probability of an error will be small.

The requirements model has been checked for consistency, with help of the analyze function of the CASE tool ProMod, and no syntactical errors were detected in the requirements model as it is now defined.

7.2 Recommendations

The fact that no syntactical errors were detected in the requirements model does not guarantee that the model gives a correct specification of the presentation layer. Therefore, it is advisable to check the requirements model for a correct specification before the next step towards an implementation is made. The Hatley and Pirbhai method does not discuss this matter and the CASE tool ProMod offers no possibility to simulate the model so, the first question which has to be answered is how the model will be checked.

The second step towards an implementation is creating the architecture model of the presentation layer. This model defines modules, each which can be implemented in hardware or software. The choice if a module is implemented in hardware or software is not an easy one. An implementation of the presentation layer has to be very flexible, which makes the software solution preferable to a hardware solution, but for reason of performance a hardware solution will be in favour.

The presentation entity is concerned with communication (with its peer) and data translation. The data translation is liable to be the critical factor in communication performance between
application processes [Teru90]. Therefore, the most profit can be made by making the data translation process fast. The encoder/decoder process is concerned with the data translation so this process is probably the best candidate to be implemented in hardware (at least for a substantial part), see [Reut92] chapter 7.

When making the choice if a module is implementated in hardware or in software, the question always will be: what is more important, speed or flexibility? When considering speed requirements, one should also think of: how many times will this specific action be taken during a connection? An example of such a considerations is the following:

Upon receipt of an ARU PPDU, the DCS has first to be set to the contents of the presentation context identifier list contained in the same ARU PPDU, before the user data can be decoded. As the filter_dcs process has to handle this list and set the DCS accordingly, this process should be fast too, to be able to decode the data within a certain time limit. But, the connection will be released upon receipt of an ARU PPDU so, no PPDU’s with user data will follow after the ARU PPDU. This specific service will only appear once during a connection so making this service fast will hardly improve the performance of the presentation entity.

When working on a real implementation several problems will arise which are not defined in the recommendations, such as giving an implementation of a SAP. In [Svob89] some of these implementation matters are discussed and it might be of interest to take to notice the results of this paper.
Chapter 8

Literature

[Beve89] Bever, M.
OSI Application Layer-Entwicklungsstand und Tendenzen.
IBM European Networking Center, Heidelberg, West Germany, 1989
Technical report no:43.8901

[Beve88] Bever, M. and M. Zimmerman
Data Translation in Heterogeneous Computer Networks.
IBM European Networking Center, Heidelberg, West Germany, 1988
Technical report no: 43.8806

[Blac89] Black, U.
Data Networks: Concepts, Theory and Practice.
New Jersey: Prentice Hall Int., 1989

[Canes87] Caneschi, F. and E. Merelli
Standardizing the presentation layer. Why and what ?, Proc. Seventh
Int. Conf. on Distributed Computing Systems, Berlin, 21-25 Sept. 1987,

[Garb88] Garbe, K.
Die Darstellungsschicht im OSI-Modell- Funktionen und Probleme ihrer
Implementierung.
Informatik-Zentrum des Hochschulwesens Technische Universität
Dresden, 1988

[Hatl87] Hatley, D. and I. Pirbhai

[Hens88] Henshall, J. and A. Shaw
OSI Explained: End-to-end computer communication standards.
Chichester: Ellis Horwood, 1988
Computer Communications and Networking.

[Hurk92] Hurkmans, M.
A requirements model of the OSI presentation Layer.
Faculty of Electrical Engineering, Digital Information Systems Group,
Eindhoven University of Technology, February 1992

[Knig88] Knightson, K. and T. Knowles, J. Larmouth
Standards For Open Systems Interconnection.

[Lomb87] Lombardo, A.
Designing presentation service by Lotus, Proc. Second Fall Joint
Exploring Technology, Today and Tomorrow, 1987

[Mant88] Mantelman, L.
Upper layers: From bizarre to bazaar, Data Communications, Jan. 1988
vol.1, no.1, pp.110-128.

[Prom89] ProMod Computer Aided Software Engineering.
User Manual, Version V1.7 A,
GEI, Gesellschaft für Electronische Information Verarbeitung,
December 1989.

[Rama88] Raman, L.G.
OSI upper layer protocol requirements for TMN operations, Proc. Seventh
Conf. on Computer Communications, New Orleans, 27-31 March 1988,
pp. 181-185, 1988 IEEE INFOCOM: The conference on computer
communications, 1988

[Reut92] Reuterink, J.A.
An architecture of the presentation layer of OSI.
Faculty of Electrical Engineering, Digital Information Systems Group,

[Rose90] Rose, M.
The Open Book: A practical perspective on OSI.
New Jersey: Prentice Hall Int., 1990

[Svob89] Svobodova, L.
Implementing OSI Systems, IEEE journal on selected areas in

[Tayl87] Taylor, G.J.
Presentation of user data in the presentation layer, Proc. Int. Conf.
International Open Systems 87, Vol. 1, London, March 1987,

[Teru90] Teruji Shiroshita
A Data Processing Performance Model for the OSI Application Layer

[Vini91] Ving, T. and S. Fischer
CASE, een overzicht van de markt.
[ISO.7898] Information Processing Systems - Open Systems Interconnection -
Basic Reference Model, Part 3: Naming and Addressing, International

[X.200] Information Processing Systems - Open Systems Interconnection -

[X.208] Information Processing Systems - Open Systems Interconnection -
Specification of Abstract Syntax Notation One (ASN.1), International

[X.209] Information Processing Systems - Open Systems Interconnection -
Specification of Basic Encoding Rules for ASN.1, International

[X.215] Information Processing Systems - Open Systems Interconnection - Basic
Connection Oriented Session Service Definition, International Standards
Organization, 1984, ISO 8326.

[X.216] Information Processing Systems - Open Systems Interconnection -
Connection Oriented Presentation Service Definition, International

[X.226] Information Processing Systems - Open Systems Interconnection -
Connection Oriented Presentation Protocol Definition, International
Appendix A: Tables of service primitives and PPDU types

Table A1: Presentation Service Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_Connect</td>
<td>C</td>
<td>Connection establishment</td>
</tr>
<tr>
<td>P_Release</td>
<td>C</td>
<td>Connection release</td>
</tr>
<tr>
<td>P_U_Abort</td>
<td>N</td>
<td>User-initiated abort</td>
</tr>
<tr>
<td>P_P_Abort</td>
<td>P</td>
<td>Provider-initiated abort</td>
</tr>
<tr>
<td>P_Alter_Context</td>
<td>C</td>
<td>Context addition and deletion</td>
</tr>
<tr>
<td>P_Data</td>
<td>N</td>
<td>Normal data transfer</td>
</tr>
<tr>
<td>P_Expedited_Data</td>
<td>N</td>
<td>Expedited data transfer</td>
</tr>
<tr>
<td>P_TYPED_Data</td>
<td>N</td>
<td>Typed data transfer</td>
</tr>
<tr>
<td>P_Capability_Data</td>
<td>C</td>
<td>Control information transfer</td>
</tr>
<tr>
<td>P_Token_Give</td>
<td>N</td>
<td>Give the token to the peer</td>
</tr>
<tr>
<td>P_Token_Please</td>
<td>N</td>
<td>Request the token from the peer</td>
</tr>
<tr>
<td>P_Control_Give</td>
<td>N</td>
<td>Give all tokens to the peer</td>
</tr>
<tr>
<td>P_Sync_Minor</td>
<td>O</td>
<td>Add a synchronization point</td>
</tr>
<tr>
<td>P_Sync_Major</td>
<td>C</td>
<td>Replace all sync. points with one</td>
</tr>
<tr>
<td>P_Resynchronize</td>
<td>C</td>
<td>Go back to previous sync. point</td>
</tr>
<tr>
<td>P_U_Exception_Report</td>
<td>C</td>
<td>Report of an user exception</td>
</tr>
<tr>
<td>P_P_Exception_Report</td>
<td>P</td>
<td>Report of a provider exception</td>
</tr>
<tr>
<td>P_Activity_Start</td>
<td>N</td>
<td>Start an activity</td>
</tr>
<tr>
<td>P_Activity.Resume</td>
<td>N</td>
<td>Restart an interrupted activity</td>
</tr>
<tr>
<td>P_Activity_End</td>
<td>C</td>
<td>End an activity</td>
</tr>
<tr>
<td>P_Activity_Interrupt</td>
<td>C</td>
<td>Interrupt an activity</td>
</tr>
<tr>
<td>P_Activity_Discard</td>
<td>C</td>
<td>Abandon an activity</td>
</tr>
</tbody>
</table>

Type of presentation service primitive:

- **C** Confirmed service
- **N** Non confirmed service
- **O** Optional confirmed service
- **P** Provide initiated service
<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>Connect Presentation</td>
</tr>
<tr>
<td>CPA</td>
<td>Connect Presentation Accept</td>
</tr>
<tr>
<td>CPR</td>
<td>Connect Presentation Reject</td>
</tr>
<tr>
<td>AC</td>
<td>Alter Context</td>
</tr>
<tr>
<td>ACA</td>
<td>Alter Context Acknowledge</td>
</tr>
<tr>
<td>ARU</td>
<td>Abnormal Release, User initiated</td>
</tr>
<tr>
<td>ARP</td>
<td>Abnormal Release, Provider initiated</td>
</tr>
<tr>
<td>TD</td>
<td>Transfer Data</td>
</tr>
<tr>
<td>TE</td>
<td>Transfer Expedited data</td>
</tr>
<tr>
<td>TTD</td>
<td>Transfer Typed Data</td>
</tr>
<tr>
<td>TC</td>
<td>Transfer Capability data</td>
</tr>
<tr>
<td>TCC</td>
<td>Transfer Capability confirm</td>
</tr>
<tr>
<td>RS</td>
<td>Resynchronize</td>
</tr>
<tr>
<td>RSA</td>
<td>Resynchronize Acknowledge</td>
</tr>
</tbody>
</table>

Table A2: PPDU types
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_Connect</td>
<td>C</td>
<td>Connection establishment</td>
</tr>
<tr>
<td>S_Release</td>
<td>C</td>
<td>Connection release</td>
</tr>
<tr>
<td>S_U_Abort</td>
<td>N</td>
<td>User-initiated abort</td>
</tr>
<tr>
<td>S_P_Abort</td>
<td>P</td>
<td>Provider-initiated abort</td>
</tr>
<tr>
<td>S_Data</td>
<td>N</td>
<td>Normal data transfer</td>
</tr>
<tr>
<td>S_Expeditied_Data</td>
<td>N</td>
<td>Expedited data transfer</td>
</tr>
<tr>
<td>S_Typed_Data</td>
<td>N</td>
<td>Typed data transfer</td>
</tr>
<tr>
<td>S_Capability_Data</td>
<td>C</td>
<td>Control information transfer</td>
</tr>
<tr>
<td>S_Token_Give</td>
<td>N</td>
<td>Give the token to the peer</td>
</tr>
<tr>
<td>S_Token_Please</td>
<td>N</td>
<td>Request the token from the peer</td>
</tr>
<tr>
<td>S_Control_Give</td>
<td>N</td>
<td>Give all tokens to the peer</td>
</tr>
<tr>
<td>S_Sync_Minor</td>
<td>O</td>
<td>Add a synchronization point</td>
</tr>
<tr>
<td>S_Sync_Major</td>
<td>C</td>
<td>Replace all sync. points with one</td>
</tr>
<tr>
<td>S_Resynchronize</td>
<td>C</td>
<td>Go back to previous sync. point</td>
</tr>
<tr>
<td>S_U_Exception_Report</td>
<td>N</td>
<td>Report of an user exception</td>
</tr>
<tr>
<td>S_P_Exception_Report</td>
<td>P</td>
<td>Report of a provider exception</td>
</tr>
<tr>
<td>S_Activity_Start</td>
<td>N</td>
<td>Start an activity</td>
</tr>
<tr>
<td>S_Activity_Resume</td>
<td>N</td>
<td>Restart an interrupted activity</td>
</tr>
<tr>
<td>S_Activity_End</td>
<td>C</td>
<td>End an activity</td>
</tr>
<tr>
<td>S_Activity_Interrupt</td>
<td>C</td>
<td>Interrupt an activity</td>
</tr>
<tr>
<td>S_Activity_Discard</td>
<td>C</td>
<td>Abandon an activity</td>
</tr>
</tbody>
</table>

Type of presentation service primitive:

C  Confirmed service
N  Non confirmed service
O  Optional confirmed service
P  Provide initiated service