Eindhoven University of Technology

MASTER

An HDLC byte processor specification in SDL : towards an implementation in IDaSS

Arts, M.W.L.

Award date:
1994

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Master’s Thesis:

An HDLC byte processor specification in SDL - towards an implementation in IDaSS

M.W.L. Arts

Coach : Dr. ir. A.C. Verschueren
Supervisor : Prof. ir. M.P.J. Stevens
Period : September 1993 - August 1994

The Faculty of Electrical Engineering of Eindhoven University of Technology does not accept any responsibility regarding the contents of Master’s Theses.
Abstract

This report deals with part of the research done to implement a microprocessor that conforms to High-level Data Link Control (HDLC) procedures as described in the ISO International Standards 3309, 4335, and 7809. The study involves a rather theoretical description of OSI data link layer issues, like the definition of data link services (described in CCITT Recommendation X.212) and the relevant functions of data link protocols. The investigation is continued by a specification in the CCITT Specification and Description Language (SDL) of the microcontroller’s required behaviour. This behaviour constitutes the HDLC basic repertoire of commands and responses and the specification resides on HDLC frame level. Besides the exact control procedures, also the interfaces to the hierarchically lower and higher system layers are defined. At last, some suggestions for implementation in hardware are given, especially targetted on the use of the Interactive Design and Simulation System (IDaSS).
'Nothing that actually occurs is of the smallest importance.'

Oscar Wilde
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ix</td>
<td>ix</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Aim of the research</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>The OSI reference model and the data link layer</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>Functions of the data link layer</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Services of the data link layer</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>General data link service definitions</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Definition of the OSI connection-mode data link service</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Overview of the data link service</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Model of the connection-mode data link service</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Sequence of CODLS primitives</td>
<td>10</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Summary of the sequence of CODLS primitives at one DLC endpoint</td>
<td>15</td>
</tr>
<tr>
<td>2.2.5</td>
<td>CODLS primitives and their parameters</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Protocols of the data link layer</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of data link protocols</td>
<td>21</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Introduction</td>
<td>21</td>
</tr>
<tr>
<td>3.1.2</td>
<td>The historical development of data link layer functions</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>HDLC procedures</td>
<td>24</td>
</tr>
<tr>
<td>3.2.1</td>
<td>HDLC standards</td>
<td>24</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The HDLC frame structure</td>
<td>25</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Consolidation of HDLC classes of procedures</td>
<td>26</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Elements of HDLC procedures</td>
<td>30</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Description of the HDLC procedures in TWS unbalanced operation (UNC and UAC) using the basic repertoire of commands and responses</td>
<td>34</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Description of the HDLC procedures in TWS balanced operation (BAC) using the basic repertoire of commands and responses</td>
<td>39</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Time-out function considerations</td>
<td>42</td>
</tr>
<tr>
<td>3.2.8</td>
<td>Limitations of HDLC</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>Formal specification of the HDLC controller</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>The use of formal description techniques</td>
<td>45</td>
</tr>
<tr>
<td>4.1.1</td>
<td>The use of formality in the design process</td>
<td>45</td>
</tr>
<tr>
<td>4.1.2</td>
<td>The CCITT Specification and Description Language (SDL)</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Specification of the HDLC byte controller in SDL</td>
<td>48</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Specification strategy</td>
<td>48</td>
</tr>
<tr>
<td>4.2.2 Structure representation; the OSI-SDL relationship</td>
<td>49</td>
</tr>
<tr>
<td>4.2.3 Behaviour representation</td>
<td>53</td>
</tr>
<tr>
<td>4.2.4 Adding quantitative information to the specification</td>
<td>59</td>
</tr>
<tr>
<td>4.2.5 The use of timers in the specifications</td>
<td>61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Implementation suggestions and conclusions</td>
<td>63</td>
</tr>
<tr>
<td>5.1 Towards an implementation in IDaSS</td>
<td>63</td>
</tr>
<tr>
<td>5.1.1 The SDLC/HDLC controller in IDaSS</td>
<td>63</td>
</tr>
<tr>
<td>5.1.2 Possible implementation problems</td>
<td>64</td>
</tr>
<tr>
<td>5.2 General conclusions</td>
<td>66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A The OSI reference model</td>
<td>71</td>
</tr>
<tr>
<td>A.1 The beginning of OSI</td>
<td>71</td>
</tr>
<tr>
<td>A.2 The OSI reference model, services, and protocols</td>
<td>72</td>
</tr>
<tr>
<td>A.2.1 The OSI architecture</td>
<td>72</td>
</tr>
<tr>
<td>A.2.2 The OSI service specifications</td>
<td>73</td>
</tr>
<tr>
<td>A.2.3 The OSI protocol specifications</td>
<td>73</td>
</tr>
<tr>
<td>A.3 Elements of the OSI reference model</td>
<td>74</td>
</tr>
<tr>
<td>A.3.1 The seven-layer model</td>
<td>74</td>
</tr>
<tr>
<td>A.3.2 Services and service access points</td>
<td>75</td>
</tr>
<tr>
<td>A.3.3 Functions and protocols</td>
<td>76</td>
</tr>
<tr>
<td>A.3.4 Naming and addressing</td>
<td>76</td>
</tr>
<tr>
<td>A.3.5 Connections</td>
<td>77</td>
</tr>
<tr>
<td>A.3.6 Data transfer on a connection</td>
<td>77</td>
</tr>
<tr>
<td>A.3.7 The relation of OSI to other models</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B A summary of the CCITT Specification and Description Language (SDL)</td>
<td>79</td>
</tr>
<tr>
<td>B.1 Introduction</td>
<td>79</td>
</tr>
<tr>
<td>B.2 Basic SDL concepts</td>
<td>80</td>
</tr>
<tr>
<td>B.2.1 SDL building blocks</td>
<td>80</td>
</tr>
<tr>
<td>B.2.2 Processes</td>
<td>81</td>
</tr>
<tr>
<td>B.2.3 Procedures</td>
<td>83</td>
</tr>
<tr>
<td>B.2.4 Data handling</td>
<td>84</td>
</tr>
<tr>
<td>B.3 Structuring and refining SDL systems</td>
<td>84</td>
</tr>
<tr>
<td>B.3.1 Partitioning</td>
<td>85</td>
</tr>
<tr>
<td>B.3.2 Refinement</td>
<td>85</td>
</tr>
<tr>
<td>B.4 Additional concepts</td>
<td>86</td>
</tr>
<tr>
<td>B.4.1 Services</td>
<td>86</td>
</tr>
<tr>
<td>B.4.2 Other additional concepts</td>
<td>86</td>
</tr>
<tr>
<td>B.5 A summary of SDL/GR constructs</td>
<td>87</td>
</tr>
</tbody>
</table>
C SDL specifications of the HDLC byte controller

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System specification</td>
<td>92</td>
</tr>
<tr>
<td>System specification - signal and variable declarations</td>
<td>93</td>
</tr>
<tr>
<td>System specification - block decompositions</td>
<td>94</td>
</tr>
<tr>
<td>Primary station - block specification and declarations</td>
<td>95</td>
</tr>
<tr>
<td>Primary station - data queue management specification (textual)</td>
<td>96</td>
</tr>
<tr>
<td>Primary station - process Setup</td>
<td>97</td>
</tr>
<tr>
<td>Primary station - process TX</td>
<td>98</td>
</tr>
<tr>
<td>Primary station - process RX</td>
<td>105</td>
</tr>
<tr>
<td>Primary station - process SR</td>
<td>112</td>
</tr>
<tr>
<td>Secondary station - block specification and declarations</td>
<td>116</td>
</tr>
<tr>
<td>Secondary station - data queue management specification (textual)</td>
<td>117</td>
</tr>
<tr>
<td>Secondary station - process TX</td>
<td>118</td>
</tr>
<tr>
<td>Secondary station - process RX</td>
<td>124</td>
</tr>
<tr>
<td>Secondary station - process SR</td>
<td>133</td>
</tr>
<tr>
<td>Combined station - block specification and declarations</td>
<td>136</td>
</tr>
<tr>
<td>Combined station - data queue management specification (textual)</td>
<td>137</td>
</tr>
<tr>
<td>Combined station - process TX</td>
<td>138</td>
</tr>
<tr>
<td>Combined station - process RX</td>
<td>150</td>
</tr>
<tr>
<td>Combined station - process SR</td>
<td>166</td>
</tr>
<tr>
<td>General macro definitions</td>
<td>171</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Relationship between the DLS primitives and the data link protocol. ........ 6
2.2 The X.212 queue model of a data link connection. ............................. 8
2.3 a) Successful DLC establishment; b) DLC collision. ............................ 11
2.4 a) DLS user invoked DLC release; b) Simultaneous DLS user invoked DLC release. ................................................................. 11
2.5 a) DLS provider invoked DLC release; b) Simultaneous DLS user and DLS provider invoked DLC release. ................................. 12
2.6 a) DLS user rejection of a DLC establishment attempt; b) DLS provider rejection of a DLC establishment attempt. .............................. 13
2.7 Both primitives are destroyed in the queue. ......................................... 14
2.8 a) DISCind arrives before CONNresp is sent; b) DISCind arrives after CONNresp is sent. ................................................................. 14
2.9 Normal data transfer. ................................................................. 15
2.10 a) DLS user invoked reset; b) Simultaneous DLS user invoked reset. 15
2.11 a) DLS provider invoked reset; b) Simultaneous DLS user and DLS provider invoked reset. ................................................................. 16
2.12 State transition diagram for sequences of CODLS primitives at one DLC endpoint. ................................................................. 17
2.13 Summary of CODLS primitives and parameters. .................................. 18

3.1 Classification of the many data link protocols and protocol types. .......... 22
3.2 The HDLC basic frame structure. .................................................... 25
3.3 Unbalanced data link configuration. .................................................. 27
3.4 Balanced data link configuration. ...................................................... 27
3.5 HDLC classes of procedures. .......................................................... 29
3.6 Information transfer format of control field bits. .................................. 30
3.7 Supervisory format of control field bits. ............................................ 31
3.8 Unnumbered format of control field bits. ........................................... 31
3.9 Format of the FRMR frame information field. ...................................... 38

4.1 The specification of an OSI service in SDL. ........................................ 49
4.2 The specification of an OSI protocol in SDL. ....................................... 50
4.3 SDL specification of the local behaviour of an OSI data link layer entity. 51
4.4 Appointment of states to each communication phase. ............................ 54
4.5 State transition diagram for the statical functional behaviour of the HDLC byte controller. ................................................................. 56
A.1 The OSI reference model, services, and protocols: successively more detailed and more constraining specifications. 73
A.2 The seven-layer OSI architecture. 74
A.3 Entities, service access points (SAPs), protocols, connections, and connection endpoints (CEPs). 76
A.4 The logical relationship between data units in adjacent layers. 78
B.1 An SDL system diagram. 81
B.2 An SDL block diagram. 82
B.3 An SDL block substructure. 85
Preface

'To demonstrate things that are self-evident using things that are not evident at all is typical of those who are not able to distinguish between what is understandable as such and what is not. The consequence is that these people talk about empty names, with no possible outcome.'

Aristotle (Phys. II, 1, 193a)

High-level Data Link Control (HDLC) procedures are amongst the oldest recognized protocols residing at the OSI data link layer. Many of today's more sophisticated bit-oriented data link protocols are derived from HDLC, or they merely represent a specific subset of it, like X.25 LAPB and LAPD. Most of the elementary functions for reliable data communication can be found in these protocols and various widely used modes of operation can be accommodated by them.

For some years, research has been going on at the Digital Information Systems Section of the Eindhoven University of Technology to implement a microcontroller conforming to HDLC procedures. However, probably the most important issue – apart from the rather complicated hardware design problems – of specifying the exact behaviour of such a controller to conform to the various standards had yet been untouched. This report is meant to solve these problems and make room for an efficient and suitable implementation by means of such a specification.

In Chapter 1 a further introduction into the subject is given and a more general framework of the topic is created. Essential functions of the OSI data link layer will be treated in order to explain the context in which all data link level control procedures should be placed.

The concept of data link services is introduced in Chapter 2. There, it is also elucidated how this concept can be used to model the communication over a data link connection at a higher level of abstraction. This communication, which is defined in terms of data link service primitives, will show to be crucial in describing the interaction between the OSI data link layer and its hierarchically higher layers.
In Chapter 3 the attention is drawn to the data link layer protocols level. After a short historical view of data link control procedures, the relevant ISO International Standards regarding HDLC are discussed. Here, most of the HDLC procedures are described (textually) to a rather high level of detail. Of course, this will serve the understanding of the more formal specifications that follow in Chapter 4.

In that Chapter, the exact reasons for specification are given as well as the strategy used to gain the wanted results in the CCITT Specification and Description Language (SDL).

Conclusions and suggestions towards an implementation are given in Chapter 5. Without taking into account too much of the implementation details it is made plausible how the SDL specification can lead to a correct hardware design in IDaSS, the local digital circuit's design environment. At last, some possible implementation complications are handed to facilitate further research.

Unfortunately there wasn't enough time left for me to actually start with the implementation; the specification part has proved to be a very comprehensive and time-consuming job. To gain complete control over the 'compulsory' standards was probably the toughest part, but correctly interpreting them into something “that is understandable as such” was a breathtaking adventure.

I am convinced of the fact that my efforts will eventually lead to the desired results, since the project is supervised by a real expert: my coach Ad Verschueren. He assigned me this challenging topic of research and was always there to answer all my questions, even the stupid ones! His patience, continuous interest, expertise, and flexibility made this last year of my study to a very enjoyable period.

For both the valuable and non-valuable discussions we’ve had, I'd like to thank my fellow-students at the Digital Information Systems Section, including: Marco de Bakker, Wilfred Bosch, Marco van Hassel, Ton van Hekezen, Patrick Heuts, Peter Huis in 't Veld, Leon van de Laar, Robin Stenfert, Jos Verhaegh, Richard Vermeij, and Bart-Jan Wattel. Also, the staff members of the tenth floor are appreciated for giving aid and comfort, in particular: Prof.ir. Mario Stevens for his pragmatism and special interest shown during my research and presentations, and Rian van Gaalen for her good humour.

Last, but not least, I would like to express my special thanks to my parents for their never-ending support, and my partner for life, Inge, for just being there and accepting all those lost weekends.

Marco Arts
Eindhoven, the Netherlands
August, 1994
Chapter 1

Introduction

1.1 Aim of the research

IDaSS (Interactive Design and Simulation System) is an interactive environment for creating digital circuits; it is designed and currently updated at the Digital Information Systems Section of the Eindhoven University of Technology. It can serve the design of a variety of more or less complex data processing hardware or other kinds of synchronous machines on both very large scale (VLSI) or ultra large scale (ULSI) integration level (see [Vers 90]).

The possibilities of IDaSS are fruitfully employed in lots of design projects and many research has been done to develop a library of designs. One of the current projects is the creation of a microcontroller that conforms to the High-level Data Link Control (HDLC) procedures that are defined in several ISO International Standards, residing at the data link layer of the OSI reference model. Pepijn Lavrijssen started the first attempt in 1992 to implement such a processor in IDaSS. His design was modified at various points by dr.ir. Ad Verschueren which has resulted in a hardware prototype that is focussed specifically to support HDLC or HDLC-like protocols. As will be seen later (Chapter 5), this piece of hardware fits easily to be programmed on HDLC frame level to perform such control procedures.

The missing part in the overall project is, however, the question of how exactly the microcontroller should be programmed in order to perform the required behaviour of HDLC. Therefore, it is necessary to know all details of both HDLC data link level procedures and the definition of the controller’s interfaces with the surrounding physical and network layer as supported by the OSI reference model. The general goal of this research is, hence, to specify exactly and correctly the system’s required behaviour to ease further implementation attempts. Though, before pointing out precisely how this behaviour should be specified, a more general framework of the research object will be given first.

1.2 The OSI reference model and the data link layer

In Appendix A of this report the concept of Open Systems Interconnection (OSI) and its supporting reference model is briefly discussed. It is seen there, that the aim of OSI is to remove any technical impedement from communication between systems, even though they may be of quite different origins. OSI is not concerned with the description of the internal operation of a single system; it is concerned only with the exchange of information at the points of interconnection between systems.
In organizing the description of intersystem communication, various techniques are applied to produce a modular structure. The major subdivisions, within which all OSI standards are to be interpreted, are set out in the OSI reference model (see [IS7498]). The modular approach as defined in this model provides for a comprehensible structure and subdivides the work into individual pieces of manageable size. These can then be subject to independent development and maintenance, which is particularly important in establishing standards appropriate to a rapidly developing technology, where innovation will require addition and updating of the individual parts to be performed without destabilizing the whole set.

It is important to recognize that the structured approach of the OSI reference model divides the interconnection protocols in a particular way by defining series of layers of functions. Each boundary in the model represents a demarkation between groups of functions; these functions are given visibility by different aspects of the protocol rules. A service represents the definition of such a demarkation, and, thus, it can be seen as a boundary between functions. Hence, a service is defined at the boundary between functions. A crucial distinction is to be made here between services and protocols; a series of layers can be generated, each adding value to the function of some 'lower' service in order to provide some 'higher' service. The rules which coordinate the functions thus isolated within two (or more) communicating systems then form a protocol between the systems.

In the remainder of this report we will focus on the data link layer (DLL), comprising layer 2 of the OSI reference model. More precisely, our main object of study will be the data link services and protocols and the way in which they contribute to achieving the fundamental objectives, i.e., the fundamental functions of data link control.

1.3 Functions of the data link layer

In terms of the architectural concepts of OSI, the data link layer builds on, or, as one could say, adds value, to the services provided by the underlying physical layer (PhL). The combined capability is then offered upward to the network layer (NL).

The primary objective of the data link layer is to assure the reliable transfer of user data over a data link. In achieving this objective, a data link entity must cope with both the requirements of the communications medium, residing in the physical layer, and the requirements of the user, as existing hierarchically above the data link layer. Together, these sometimes conflicting requirements – as will be shown – define the functions which must be accomplished by the data link layer. These functions can be said to be more or less common to all data link protocols. The way in which these functions are accomplished varies with type and sophistication of the actual protocol being used, as will be extensively discussed in Chapter 3.

According to [Cona 83], the set of required functions is formed by

- the initialization function which deals with the establishment of an active data link connection over an already existing physical path. The physical path may be built on one or more physical circuits. The acquisition of the path and the movement of the bits over the path are the responsibility of the underlying physical layer process and will not be subject of discussion here. Initialization usually involves the exchange of supervisory sequences establishing readiness to receive or transmit and, if necessary, identification of the parties;
1.3 Functions of the data link layer

- an identification function is necessary to identify a particular receiver or sender among the many that may be present on a multipoint facility or among the huge number that may be accessible through a switched network. Data link layer identification is usually accomplished through an exchange of, or the a priori assignment of, data link addresses. The identification function may also require the exchange of parameters describing the capability of the communicating stations;

- the underlying physical layer provides a stream of synchronized bits. A function of the data link layer is to determine where in this stream of bits the intelligence being transferred lies. The synchronization function accomplishes this by providing functions to acquire, maintain, and, if necessary, re-establish character or bit synchronization, i.e., bringing the receiver's decoding mechanisms into alignment with the transmitter's encoding mechanisms;

- a framing mechanism is necessary to divide the user data into segments suitable for transmission through the data link. Extremely long blocks of information are unlikely to survive transmission through a noisy medium without error. Very short blocks, on the other hand, may be inefficient. Framing can aid the synchronization process and provide the ability to identify when data should or may not be present. It also provides convenient segments on which to apply error detection processes;

- the transparency function is one which permits the data link control to be totally 'transparent' to the format or structure of the user data. Transparency permits the user to send information in any code set, in any length, and in any format with the assurance that data link mechanisms will not interpret any user data as link control information. Data link protocols vary widely in their capability and techniques used to provide transparency;

- receivers need to be able to regulate the flow of information into their systems in order to prevent them from being overwhelmed by incoming data where the input rate exceeds the station's capacity to accept and process the data. Flow control functions accomplish this regulation;

- error control functions provide for the detection of errors induced by the transmission medium, the acknowledgment of correctly received segments, and requests for retransmission of segments containing errors. Cyclic redundancy checks are the most commonly used error detection techniques. Some data link protocols also employ a sequence control function, which numbers and verifies individual segments of data, to guarantee the detection of missing segments. Sliding window protocols form a good example of providing this functionality;

- exception condition detection and recovery include the functions required to detect and recover from abnormal occurrences, such as loss of response, illegal or invalid sequences, severed links, and the many other exception conditions that can occur to information moving over the data link. Time-out processes are a common method of detecting such occurrences;

- following the transfer of the user data, the data link, which was logically established by the initialization function, can be terminated. Termination functions involve tidying
up the data link by assuring that all data sent have been received and then 'grace-
fully' clearing the logical connection. Data link termination does not necessarily involve
disconnection of the physical path.

The listed functions taken together comprise the set of services that the data link layer makes
available across the boundary with the network layer. A set or subset of these services, called
a data link layer entity, is then invoked by the network layer to support a particular instance
of intersystem communication. The services of the data link layer will be discussed in the
next Chapter.
Chapter 2

Services of the data link layer

2.1 General data link service definitions

In terms of OSI, the fundamental purpose of the data link layer is to provide a set of services to the network layer. These services, called the data link services (DLSs), are those associated with the inherent limitations of the interconnecting medium. They represent the composite of the services of the underlying physical layer and the value-added services of the data link layer. In this sense, the data link layer, in addition to the physical layer, is called the data link service provider, or DLS provider for short, with respect to the hierarchically higher layers. Those parts of the system above the data link service boundary are collectively called the data link service user\(^1\), or DLS user for short.

The DLS provider will, in general, be distributed across a number of distinct pieces of equipment. However, the way in which this subdivision takes place is not of concern to the DLS user. The DLS user may view the DLS provider as a distributed abstract machine, the operation of which provides the specified service. Another important aspect in the definition of a DLS is that it establishes the properties of a type of communication, i.e., the relationship, in general, between any entity playing the role of DLS user and the DLS provider. However, any communication will be an example of this ideal relationship, and so it is an instance of the type defined by the DLS. Communicating systems may be involved at any time in a number of such instances, which may be related parts of some larger activity, or may be independent (see [Lini 83]).

Regarding the type of data transmission, a so called connection-mode data transmission is assumed here. This implies transmission of DLS data units only within the context of a data link connection, or DLC for short, that has been previously established (see also Appendix A). Therefore, the data link services that are dealt with in this Chapter can also be referred to as connection-mode data link services, or CODLSs for short. Figure 2.1 shows that a DLS provider – more particularly, a DLL in the end system – must provide a translation between

1. the primitives and parameters of the OSI CODLS, and
2. data link protocol data units, or DLPDUs for short.

The primitives and parameters of the OSI CODLS will be discussed in the remainder of this Chapter. The communication in terms of data link protocol data units, or rather, the

\(^1\)In the following, the data link service user will refer to the network layer only.
6 Services of the data link layer

Figure 2.1: Relationship between the DLS primitives and the data link protocol.

associated data link control procedures will be explained in Chapter 3, while leaving the aforementioned translation problem for Chapter 4.

2.2 Definition of the OSI connection-mode data link service

The OSI CODLS is defined in CCITT Recommendation X.212 [X.212] and in the ISO International Standard 8886 [IS8886], which are, as far as the CODLS is concerned, identical. The description following below can be seen as a précis of CCITT Recommendation X.212.

2.2.1 Overview of the data link service

The previous Section and the last part of Chapter 1 showed that the DLS provider makes invisible to the DLS users the way in which supporting communications resources are utilized to achieve transparent and reliable transfer of data between these DLS users. Also, several functions have been mentioned for which the DLS provider is responsible.

Among these functions the most important are

- *independence of the underlying physical layer*; the DLS provider relieves DLS users from all concerns regarding which configuration is available (e.g. a point-to-point connection) or which physical facilities are used (e.g. half-duplex transmission);

- *transparency of transferred information*; the DLS provider provides for the transparent transfer of DLS user data. It does not restrict the content, format or coding of the information, nor does it ever need to interpret its structure or meaning;

- *reliable transfer of data*; the DLS provider relieves the DLS user from loss, insertion, corruption or, if requested, misordering of data which may occur. In some cases of
unrecoverable errors in the DLL, duplication or loss of data link service data units (DLSDUs) may occur

- **quality of service selection;** the DLS provider makes available to DLS users a means to request and to agree upon a quality of service (QOS) for the transfer of data. QOS is specified by means of QOS parameters representing characteristics such as throughput, transit delay, accuracy and reliability;

- **addressing;** the DLS provider allows the DLS user to identify itself and to specify the data link service access points (DLSAPs) to which a DLS is to be established whenever more than two DLSAPs are supported by the DLS provider. Data link addresses have only local significance within a specific data link configuration over a single transmission medium (point-to-point or multipoint physical connections) or a group of parallel transmission media (multilink function). Therefore, it is not appropriate to define a global addressing structure.

2.2.2 Model of the connection-mode data link service

**Types of service primitives**

Data link services are invoked or indicated by the use of a set of service primitives exchanged across the layer boundary with the network layer. In other words, information is passed between the DLS user and the DLS provider by service primitives (which may contain parameters) at the DLSAPs. It should be remembered that the primitives described below are totally abstract in nature, i.e., they are used to illustrate and describe the DLS and are not an implementation specification. The four standard types of OSI primitives used to interact between the data link layer and the network layer are:

- **request**, which is issued by the DLS user to invoke a specific link layer procedure;

- **indication**, which is issued by the DLS provider to advise the DLS user that a service has been invoked by either the peer DLS user in a connected system or by the DLS provider in the local system;

- **response**, which is issued by the DLS user to complete a procedure previously indicated;

- **confirm**, which is issued by the DLS provider to complete a previously requested procedure.

**Queue model of the data link connection**

In [X.212] a queue model of a DLC is used to aid the understanding of the end-to-end service features perceived by DLS users, without attempting to specify or constrain DLS implementations. This queue model, which is illustrated in Figure 2.2, represents the operation of a

---

2 Detection of duplicate or lost DLSDUs is often performed by DLS users.

3 If a DLS user needs to distinguish among several DLCs at the same DLSAP, then a local connection endpoint identification (CEI) mechanism must be provided. Such an implicit identification is not described in this report.

4 These four types of primitives usually follow the specific name of a service being used and are then abbreviated to req, ind, resp, and conf, respectively.
DLC in the abstract by a pair of queues linking the two DLSAPs. There is one queue for each direction of information flow. Each queue represents a flow control function in one direction of transfer. The ability of a DLS user to add objects to a queue will be determined by the behaviour of the other DLS queue. Objects are entered or removed from the queue as a result of interactions at the two DLSAPs. The pair of queues is considered to be available for each potential DLC.

The following objects may be placed in a queue by a DLS user:

- a *connect object*, representing a DL-CONNECT primitive (or CONN for short) and its parameters;
- a *data object*, representing a DL-DATA primitive (DATA) and its parameters;
- a *reset object*, representing a DL-RESET primitive (RSET) and its parameters; and
- a *disconnect object*, representing a DL-DISCONNECT primitive (DISC) and its parameters.

The following objects may be placed in a queue by the DLS provider:

- a reset object, representing a RSET primitive and its parameters;
- a *synchronization mark object*; and
- a disconnect object, representing a DISC primitive and its parameters.

The queues are defined to have the following general properties. A queue is empty before a connect object has been entered and can be returned to this state, with loss of its contents, by the DLS provider; objects are entered into a queue by the sending DLS user, subject to

---

Figure 2.2: The X.212 queue model of a data link connection.

---

5Service primitive parameters are discussed in Section 2.2.5.
control by the DLS provider. Objects may also be entered by the DLS provider. Objects are removed from the queue, under the control of the receiving DLS user. Furthermore, objects are normally removed in the same order that they were entered. However, objects may be destructed by other objects, as described below. Finally, it should be added that a queue has a limited capacity, but this capacity is not necessarily either fixed or determinable.

As can be seen by the objects that can be placed in the queue by a DLS user, there are four types of services (within the three phases of communication, namely DLC establishment, data transfer, and DLC release) that should be supported. How these types of services are modelled is described below.

1. DLC establishment - A pair of queues is associated with a DLC between two DLSAPs when the DLS provider receives a CONNreq primitive at one of the DLSAPs, and a connect object is entered into one of the queues. From the standpoint of the DLS users of the DLC, the queues remain associated with the DLC until a disconnect object representing a DISC primitive is either entered or removed from the queue.

DLS user A, who initiates a DLC establishment by entering a connect object representing a CONNreq primitive into the queue from DLS user A to DLS user B, is not allowed to enter any other object, other than a disconnect object, into the queue until after the connect object representing the CONNconf primitive has been removed from the DLS user B to DLS user A queue. In the queue from DLS user B to DLS user A objects can be entered only after DLS user B has entered a connect object representing a CONNresp primitive.

2. Normal data transfer - Flow control on the DLC is represented in this queue model by the management of the queue capacity, allowing objects to be added to the queues. The addition of an object may prevent addition of a further object.

Once objects are in the queue, the DLS provider may manipulate pairs of adjacent objects, resulting in deletion. An object may be deleted if, and only if, the object which follows it is defined to be destructive with respect to that object. If necessary, the last object in the queue will be deleted to allow a destructive object to be entered — they may therefore always be added to the queue. Disconnect objects are defined to be destructive with respect to all other objects. Reset objects are defined to be destructive with respect to all other objects except connect and disconnect objects.

Whether the DLS provider performs actions resulting in deletion or not will depend upon the behaviour of the DLS users and the agreed QOS for the DLC. In general, if a DLS user does not remove objects from a queue, the DLS provider shall, after some unspecified period of time, perform all the permitted deletions.

3. Reset - In order to accurately model the reset service a synchronization mark object is required. The synchronization mark object exhibits the following properties:

- it cannot be removed from the queue by a DLS user;
- a queue appears empty to a DLS user when a synchronization mark object is the next object in the queue;
- a synchronization mark object can be destroyed by a disconnect object;
• when a reset object is immediately preceded by a synchronization mark object, both the reset object and the synchronization mark object are deleted from the queue.

The initiation of a reset procedure can be modelled by the two queues as follows. The initiation of a reset procedure by the DLS provider is represented by the introduction into each queue of a reset object followed by a synchronization mark object. A reset procedure initiated by a DLS user is represented by the addition, by the DLS provider, of a reset object into the queue from the reset initiator to the peer DLS user and the insertion of a reset object followed by a synchronization mark object into the other queue.

Unless destroyed by a disconnect object, a synchronization mark object remains in the queue until the next object following in the queue is a reset object. Both the synchronization mark object and the following reset object are then deleted by the DLS provider.

4. **DLC release** - The insertion into a queue of a disconnect object, which may occur at any time, represents the initiation of a DLC release procedure. The release procedure may be destructive with respect to other objects in the two queues and eventually results in the emptying of the queues and the disassociation of the queues with the DLC.

The insertion of a disconnect object may also represent the rejection of a DLC establishment attempt or the failure to complete DLC establishment. In such cases, if a connect object representing a CONNreq primitive is deleted by a disconnect object, then the disconnect object is also deleted. The disconnect object is not deleted when it deletes any other object, including the case where it deletes a connect object representing a CONNresp primitive.

### 2.2.3 Sequence of CODLS primitives

From the queue model it is possible to extract some important constraints on the sequence in which the CODLS primitives may occur. The constraints determine the order in which primitives occur, but do not fully specify when they may occur. Other constraints, such as flow control of data, will affect the ability of a DLS user or a DLS provider to issue a primitive at any particular time.

A primitive issued at one DLC endpoint will, in general, have consequences at the other DLC endpoint. The relations of each type at one DLC endpoint to primitives at the other DLC endpoint are defined below. It should be noted, however, that a DISCreq or DISCind primitive may terminate any of the other sequences before completion, as discussed in the queue model definition of Section 2.2.2.

**Sequence of CODLS primitives during the connection establishment phase**

The connection establishment service primitives can be used to establish a DLC. The sequence of primitives in a successful connection establishment is defined by the time-sequence diagram in Figure 2.3a. The DLS user initiating the DLC establishment issues a CONNreq

---

6 This will show to have an important impact on the implementation of a DLL entity.

7 The time-sequence diagrams in Figures 2.3 to 2.11 assume the issuance of a service primitive as an instantaneous event. This is an ideal situation, though it should have no implementation consequences.
2.2 Definition of the OSI connection-mode data link service

primitive, which is indicated at the other DLS user by a CONNind primitive. If the other DLS user agrees on the DLC establishment, it issues a CONNresp primitive. The DLC establishment is successfully completed if the initiating DLS user is confirmed on the agreement by a CONNconf primitive.

If the DLS users simultaneously initiate a DLC establishment – assuming they have equal rights to do so – they both issue a CONNreq primitive at their associated DLSAP. Simultaneous issuing of CONNreq primitives at the two DLSAPs results in one DLC, as is shown in Figure 2.3b.

These DLC establishment procedures may fail either due to the inability of the DLS provider to establish a DLC or due to the unwillingness of the called DLS user to accept a CONNind primitive, which is described below.

Sequence of CODLS primitives during the connection release phase

The connection release service primitives are used to release a DLC. Initiation of the release service element is permitted at any time regardless of the current phase of the DLC. Once a release service has been initiated, the DLC will be disconnected. A DISCreq cannot be rejected. Furthermore, the DLS does not guarantee delivery of any DLSDU associated with the DLC once the release phase is entered.

Figure 2.3: a) Successful DLC establishment; b) DLC collision.

Figure 2.4: a) DLS user invoked DLC release; b) Simultaneous DLS user invoked DLC release.
The sequence of CODLS primitives depends on the origins of the DLC release action. The release may be initiated by any of the following:

- either or both of the DLS users, to release an established DLC. Initiation of the DLC release action can be originating from one DLS user, with a request from that DLS user leading to an indication to the other, as is illustrated in Figure 2.4a.

The DLC release can, generally, also be initiated by both DLS users, with a request from each of the DLS users (see Figure 2.4b);

- the DLS provider can release an established DLC. All failures to maintain a DLC are indicated in this way. Both DLS users are indicated that the DLC is released. This is shown in Figure 2.5a.

![Diagram](image1)

Figure 2.5: a) DLS provider invoked DLC release; b) Simultaneous DLS user and DLS provider invoked DLC release.

It is, however, also possible that one DLS user and the DLS provider independently initiate the DLC release, with a request from the originating DLS user and an indication to the other (see Figure 2.5b);

- a DLS user may reject a DLC establishment (indicated with CONNind) attempt by using a DISCreq. This sequence of events is defined in the time-sequence diagram in Figure 2.6a;

- if the DLS provider is unable to establish a DLC, it indicates this to the requester by a DISCind primitive, as is defined in Figure 2.6b;

- if the DLS user, having previously sent a CONNreq primitive and not received a CONNconf or DISCind primitive, wishes to abort the DLC establishment attempts, it shall issue a DISCreq. The resulting sequence of primitives is dependent upon the relative timing of the primitives involved and the transit delay of the DLS provider as shown in the time-sequence diagrams in Figures 2.7 and 2.8. No information can be implied by detecting which of these alternatives occur.
2.2 Definition of the OSI connection-mode data link service

Sequence of CODLs primitives during the data transfer phase

Normal data transfer service

The data transfer service primitives provide for an exchange of user data (DLSDUs), in either direction or in both directions simultaneously, on a DLC. The DLS preserves both the sequence and the boundaries of the DLSDUs.

The operation of the DLS in transferring DLSDUs can be modelled as a queue of unknown size within the DLS provider (see Section 2.2.2). The ability of a DLS user to issue a DATAreq or of the DLS provider to issue a DATAind primitive depends on the behaviour of the receiving DLS user and the resulting state of the queue. It is already shown earlier that this is given in by the fact that the DLS is described in terms of allowed sequences of service primitives (and their parameters), rather than when a primitive could or should be issued.

The sequence of primitives in a successful data transfer is defined in the time-sequence diagram in Figure 2.9. The sequence of primitives shown in Figure 2.9 may remain uncompleted if a DISC or a RSET primitive occurs, as is shown in the previous and the following Section, respectively.

Reset service

The reset service may be used:

- by the DLS user, to resynchronize the use of the DLC; or
- by the DLS provider, to report detected loss of data unrecoverable within the DLS. All loss of data which does not involve loss of the DLC is reported in this way.

Invocation of the reset service will unblock the flow of DLSDUs in case of congestion of the DLC; it will cause the DLS provider to discard DLSDUs, and to notify user or users that did not invoke a reset service that a reset service has occurred. The service will be completed in a finite time, irrespective of the acceptance of DLSDUs. Any DLSDUs not delivered to the DLS users before completion of the service will be discarded by the DLS provider.

The interaction between each DLS user and the DLS provider shall be either one of the following exchanges of primitives: a RSETreq from the DLS user, followed by a RSETconf from the DLS provider, or, a RSETind from the DLS provider, followed by a RSETresp from

![Image](image_url)

**Figure 2.6:** a) DLS user rejection of a DLC establishment attempt; b) DLS provider rejection of a DLC establishment attempt.
the DLS user. The RSETreq acts as a synchronization mark in the stream of DLSDUs that are transmitted by the issuing DLS user; the RSETind, likewise, acts as a synchronization mark in the stream of DLSDUs by the peer DLS user. Similarly, the RSETresp acts as a synchronization mark in the stream of DLSDUs transmitted by the responding DLS user, while the RSETconf acts as a synchronization mark in the stream of DLSDUs that are received by the DLS user which originally issued the RSETreq. No DLSDU transmitted by the DLS user before the synchronization mark in that transmitted stream will be delivered to the other DLS user after the synchronization mark in that received stream. The DLS provider will discard all DLSDUs, submitted before the issuing of the RSETreq, that have not been delivered to the peer DLS user when the DLS provider issues the RSETind. Also, the DLS provider will discard all DLSDUs, submitted before the issuing of the RSET primitive that have not been delivered to the initiator of the reset when the DLS provider issues the RSETconf. Furthermore, no DLSDU transmitted by a DLS user after the synchronization mark in that transmitted stream will be delivered to the other DLS user before the synchronization mark in that received stream.

The complete sequence of primitives depends upon the origin of the reset action and the occurrence of conflicting origins. Thus, the reset service can be defined as follows:

1. the invocation by one DLS user leads to a RSETreq from that DLS user, followed by a RSETconf from the DLS provider to that DLS user. The interaction with the peer DLS user starts with a RSETind from the DLS provider, followed by a RSETresp from

Figure 2.7: Both primitives are destroyed in the queue.

![Diagram](image.png)

Figure 2.8: a) DISCind arrives before CONNresp is sent; b) DISCind arrives after CONNresp is sent.
2.2 Definition of the OSI connection-mode data link service

Figure 2.9: Normal data transfer.

the DLS user. This is shown in Figure 2.10a;

Figure 2.10: a) DLS user invoked reset; b) Simultaneous DLS user invoked reset.

2. if both DLS users invoke the reset service simultaneously, i.e., they both issue a RSETreq, then the DLS provider will issue a RSETconf to both of them (see Figure 2.10b);

3. if the reset service is invoked by the DLS provider, then both DLS users are informed by a RSETind from the DLS provider, followed, upon acceptance of the initiated reset service, by a RSETresp from both the DLS users. This is illustrated in Figure 2.11a;

4. it is also possible that one DLS user and the DLS provider invoke a reset service simultaneously. Then, the DLS user will, following an issued RSETreq, be informed by a RSETconf from the DLS provider. The peer DLS user will, upon acceptance of the initiated reset service, respond to the DLS provider with a RSETresp primitive (see Figure 2.11b);

The above sequences of primitives may, of course, remain uncompleted if a DISC primitive occurs.

2.2.4 Summary of the sequence of CODLS primitives at one DLC endpoint

The possible overall sequences of CODLS primitives at a DLC endpoint can be deduced from the time-sequence diagrams given above and are summarized in the state transition diagram of Figure 2.12.
In this diagram

- DISC stands for either the request or indication form of the primitive in all cases;
- the labelling of the states DLS User Initiated Reset Pending and DLS Provider Initiated Reset Pending indicate the party that started the local interaction;
- the Idle state reflects the absence of a DLC. It is the initial and final state of any sequence, and once it has been re-entered, the DLC is released;
- the use of this state transition diagram to describe the allowable sequences of CODLS primitives does not impose any requirements or constraints on the internal organization of any implementations\(^8\) of the service, as will be shown in Chapter 4.

### 2.2.5 CODLS primitives and their parameters

Most of the CODLS primitives, as extensively discussed in the previous Sections, go with certain parameters. The parameters conveyed by a service primitive depend upon the type of service and on the phase of intersystem communication. Figure 2.13 summarizes the CODLS primitives and their parameters.

**Parameters of the DLC establishment service primitives**

As can be seen from Figure 2.13, the *address parameters* are used only in the DLC establishment phase and, hence, are implied (or irrelevant) in all other phases. The parameters which take addresses as values all refer to DLSAP addresses\(^9\).

The *called address* parameter conveys an address identifying the DLSAP to which the DLC is to be established. The *calling address* parameter conveys the address of the DLSAP from which the DLC has been requested. The *responding address* parameter conveys the address of the DLSAP to which the DLC has been established.

---

\(^8\)Of course, and this will prove later on, this state transition diagram can (partially) be used for the implementation of the service.

\(^9\)If the configuration allows any of these addresses to be known by the DL entity on an *a priori* basis, then these DLSAP address(es) need not explicitly be conveyed in the protocol.
The term 'quality of service' (QOS) refers to certain characteristics of a DLC as observed between the connection endpoints (CEPs). QOS describes aspects of a DLC that are attributable solely to the DLS provider. Once a DLC is established, the DLS users at the two endpoints have the same knowledge and understanding of what the QOS over their interconnecting DLC is.

QOS is determined in terms of QOS parameters, as conveyed during the DLC establishment phase. These parameters give DLS users a method of specifying their needs and give the DLS provider a basis for appropriate protocol selection. The CODLS QOS parameters can be divided into those parameters which may be selected on a per-connection basis during the DLC establishment phase, and those parameters which are not selected during DLC establishment, but whose values are known by other methods.

QOS parameters that may be selected during DLC establishment are:

- **throughput**, which is defined as the total number of DLSDU bits successfully transferred
by a DATAreq/DATAind primitive sequence divided by the input/output time for that sequence;

- **protection**, which is the extent to which a DLS provider attempts to prevent unauthorized monitoring or manipulation of DLS user originated information. Three levels of protection are defined (see [X.212, p. 189]), but these are not discussed here;

- **priority**, concerning the relationship between DLCs. This parameter specifies the relative importance of a DLC with respect to the order in which DLCs are to have their QOS degraded – if necessary –, and the order in which DLCs are to be released to recover resources – if necessary. Priority is specified by a minimum and a maximum within a given range. This parameter only has meaning in the context of some management entity able to judge relative importance.

QOS parameters for which there is no selection during DLC establishment are:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Service</th>
<th>Primitive</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLC establishment</td>
<td>DLC establishment</td>
<td>CONNreq</td>
<td>(Called address, calling address, QOS parameter set)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONNind</td>
<td>(Called address, calling address, QOS parameter set)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONNresp</td>
<td>(Responding address, QOS parameter set)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONNconf</td>
<td>(Responding address, QOS parameter set)</td>
</tr>
<tr>
<td></td>
<td>Normal data</td>
<td>DATAreq</td>
<td>(DLS user data)</td>
</tr>
<tr>
<td></td>
<td>transfer</td>
<td>DATAind</td>
<td>(DLS user data)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSETreq</td>
<td>(Reason)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSETind</td>
<td>(Originator, reason)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSETresp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSETconf</td>
<td></td>
</tr>
<tr>
<td>DLC release</td>
<td>DLC release</td>
<td>DISCreq</td>
<td>(Reason)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISCind</td>
<td>(Originator, reason)</td>
</tr>
</tbody>
</table>

Figure 2.13: Summary of CODLS primitives and parameters.
2.2 Definition of the OSI connection-mode data link service

- **transit delay**, which is the elapsed time between DATAreq primitives and the corresponding DATAind primitives. Elapsed time values are calculated only on DLSDUs that are successfully transferred;

- **residual error rate**, which is the ratio of total incorrect, lost and duplicate DLSDUs to total DLSDUs transferred across the DLS boundary during a measurement period;

- **resilience**, specifying the probability of a DLS provider initiated DLC release or a DLS provider initiated reset during a specified time interval on an established DLC.

There are, of course, many reasons for assigning a certain QOS parameter to be implemented (or not) and maybe even more ways of actually implementing it\(^\text{10}\). The use of QOS parameter selection is not required when only one level of QOS is offered by the DLS provider. This will be the general case in the remainder of this report, especially in Chapter 4. This means that these parameters are generally determined by *a priori* knowledge and agreement or are *implicitly* conveyed by the DLS provider.

**Parameters of the data transfer service primitives**

**Normal data transfer**

The *DLS user data* parameter allows the transmission of DLS user data between DLS users, without modification by the DLS provider.

The DLS user may transmit any *integral number of octets* greater than zero up to a limit determined by the DLS provider. The value of this limit is made available to the DLS user by the use of management facilities or *a priori* knowledge. Only the latter case is considered in this report.

The DLS user data is, of course, identical for each pair of DATAreq and DATAind primitives.

**Reset**

The *originator* parameter, as conveyed by the RSETind primitive, indicates the source of the reset. Its value indicates

- either the DLS user, the DLS provider, or that the originator is unknown.

The *reason* parameters, as conveyed by the RSETreq and the RSETind primitives, give information indicating the cause of the reset. The value conveyed in this parameter will be as follows:

- when the originator parameter indicates a DLS provider generated reset, the value is 'data link flow control congestion' or 'data link error';

- when the originator parameter indicates a DLS user initiated reset, the value is 'user resynchronization'; and

- when the originator parameter indicates an unknown originator, the value is 'reason unspecified'. This allows the parameters to be implied when they cannot be explicitly conveyed in the data link protocol.

\(^{10}\)Recommendation X.212 does not mention when, if, or how a certain QOS parameter should be implemented.
Parameters of the connection release service primitives

The originator parameter indicates the source of the release. Its value indicates either the DLS user, the DLS provider, or that the originator is unknown.

The reason parameter gives information about the cause of the release. The value conveyed in this parameter will be as follows:

- when the originator parameter indicates a DLS provider generated release, the value is one of:
  1. 'disconnection - permanent condition';
  2. 'disconnection - transient condition';
  3. 'connection rejection - DLSAP address unknown';
  4. 'connection rejection - DLSAP unreachable/permanent condition';
  5. 'connection rejection - DLSAP unreachable/transient condition';
  6. 'connection rejection - QOS not available/permanent condition';
  7. 'connection rejection - QOS not available/transient condition'; or
  8. 'reason unspecified';

- when the originator parameter indicates DLS user initiated release, the value is one of
  1. 'disconnection - normal condition';
  2. 'disconnection - abnormal condition';
  3. 'connection rejection - permanent condition';
  4. 'connection rejection - transient condition'; or
  5. 'reason unspecified'; and

- when the originator parameter indicates an unknown originator, the value of the reason parameter is 'reason unspecified'. This allows the parameters to be implied when they cannot be explicitly conveyed in the data link protocol.

Some general remarks on the use of service primitive parameters

Most of the service primitive parameters ‘defined’ above are still under study regarding their ability to convey more specific diagnostic and management information. Hence, it is not completely defined which values in which context the parameters listed in Figure 2.13 should contain. It should be noted, however, that the service primitive parameters, like the service primitives themselves, provide a means of describing the behaviour of intersystem communications at the DLS user level.

If the service primitive parameters are placed in this context, it will become clear that many of them can generally be implied by the DLS users or are known on an a priori basis, as is repeatedly stressed in this Section and will deliberately be pointed out in parts of the next Chapters.
Chapter 3

Protocols of the data link layer

3.1 Overview of data link protocols

3.1.1 Introduction

A rather extensive overview of the CODLS is given in the previous Chapter, conforming a higher level of abstraction in the OSI scheme. To present a more concrete and definite description of data link control, one should 'dig into' the lowest level of abstraction in the overall OSI architecture, i.e., the OSI data link protocols (see also Appendix A). With reference to Figure 2.1, a data link protocol can be said to constitute the horizontal communication – in terms of DLPDUs – between two DLL entities, which will be the subject of study in this Chapter. From the OSI reference model one can learn that this is only a virtual communication, in the sense that it merely conforms to the OSI reference model as a description of peer-to-peer communication. In reality, the DLPDUs are passed down to or forwarded up from the physical layer (PhL), for which they serve as PhL-SDUs, as is shown for the general case in Figure A.4.

In the remainder of this report, the interface between the data link layer and the physical layer, i.e., the definition of the physical layer service primitives and their parameters, will be considered as given. Moreover, the description of the composite of the data link layer and the physical layer as the DLS provider will be entirely in terms of DLPDUs, relying completely on the physical layer to co-operate in offering the higher layers the entire set of data link services.

3.1.2 The historical development of data link layer functions

Section 1.3 showed that the primary objective of the data link layer is to assure the reliable transfer of user data over a data link. There, it was also pointed out that, in achieving this fundamental objective of data link control, a DLL entity should cope with both the requirements of the communications medium (not described here) and the requirements of the user, more in particular the DLS user. Also, a set of required functions was defined, that can be said to be common to all data link protocols and the accomplishment of which varies with type and sophistication of the actual protocol being used. The set of required functions was defined to be: data link setup and termination, identification, flow control aided with segmenting and delimiting functions, providing synchronization and data transparency, error and abnormal condition recovery, sequence control, and some more general link management
functions.

In the remainder of this Section, we will see – from an historical point of view – how data link control procedures\(^1\) have evolved during the years and how they have grown to protocols embedding possibly all the required functionality described above. As an introduction, the development of the various types of data link protocols will be illustrated first. The widely used high-level data link control (HDLC) procedures, being the final protocols of interest here, are capable of completely providing the well-defined set of rules, essential to successful and efficient operation, i.e., to meet the requirements mentioned in Section 1.3.

**The early freewheeling data link protocols**

Data link control procedures are among the oldest recognized communication protocols. They have evolved continuously from the early asynchronous 'start-stop' batch-oriented protocols, through the widely implemented character-oriented protocols, to the increasingly popular bit-oriented protocols, providing for more interactive types of operation. A short classification of the many types of protocols is given in Figure 3.1.

![Figure 3.1: Classification of the many data link protocols and protocol types.](image)

In the early 1970's it became evident that the various existing character-oriented data link control procedures that had served so well in many applications (e.g. the ARPANET) were not well suited for the newer interactive applications being pursued. Technology had provided more reliable transmission facilities, more intelligent and cost-effective computers and terminals, and new frontiers for their use in almost every segment of the business, industry, government, and academic environments. Extending or modifying the existing protocols to satisfy these needs was found to be generally inadequate. The character-oriented control procedures were basically half-duplex in nature and batch-oriented in operation. They were tied to the transmission code being used, and, moreover, they generally had a rather poor throughput performance.

\(^1\)In this report, the terms 'protocol' and 'control procedure' are used as equivalents.
All in all, it was time for a new approach to data link control. An approach that would correct and improve the identified shortcomings present in the existing protocols, while providing the features and services that this new environment demanded. The bit-oriented data link control procedures provide a satisfactory solution to this problem for synchronous data communications needs.

**Bit-oriented data link control procedures**

Synchronous data link control procedures do not surround each character with start and stop bits, but place a preamble and postamble bit pattern around the user data. These bit patterns are usually called *flags*. They notify the receiver that user data are arriving and that the last user data character has arrived. To solve the problem of code-dependency, which was found in the character-oriented data link protocols, byte-count-oriented protocols were developed in the 1970's. Their principal advantage over character-oriented data link protocols is their more effective means of handling user data transparency: they simply insert a count field at the transmitting station. This field specifies the length of the user data field. As a result, the receiver need not examine the user data field contents, but can only count the incoming characters as specified by the count field.

Bit-oriented data link protocols were developed during the same period of time and they have grown to the most prevalent data link protocols, providing far more flexibility than the character-oriented protocols. According to the tutorial paper of Carlson [Carl 80], the following capabilities of early bit-oriented data link control procedures were identified as being essential:

- code-independent operation (transparency);
- adaptability to various applications, configurations, and uses in a consistent manner;
- both two-way alternate (half-duplex) and two-way simultaneous (full-duplex) data transfer capability;
- high efficiency (throughput); and
- high reliability.

Instead of counting the bytes sent or received, bit-oriented protocols use a more sophisticated framing mechanism, and, thus, make room for the essential segmenting and delimiting functions. The beginning and end of every separate frame is marked by an opening flag and a closing flag, respectively. This provides for the ability to determine where the intelligence being transferred lies. It permits the user to send information in any code set, in any length, and in any format with the assurance that data link mechanisms will not 'trip over it'. It also provides for the maintenance of synchronization processes.

Within the flags, there is also room for a _control field_. The control field (usually octet-aligned) contains information regarding the type of frame being sent or received; whether it is an initialization or termination frame or an information frame can be deduced from the control field format. Furthermore, the control field usually contains one or more frame _sequence numbers_, offering sequence control capabilities. The inclusion of _receive and send counters_ — sometimes called 'piggybacking' — permits flexible flow control with _sliding windows_.

---

2This is explained in detail in [Tane 88, p. 232 ff.].
known as continuous automatic request for repeat (ARQ). Also, an address field can be added to aid identification functions, and a frame checking sequence (FCS), which usually contains the result of a cyclic redundancy check (CRC) to provide for error control. To complete the functionality, most of the bit-oriented data link control procedures are able to use timers for the early detection of lost frames.

The HDLC family

The detection of and research on these highly preferable properties was very important and it was fully recognized by IBM at first. Their original bit-oriented data link protocol was called SDLC, synchronous data link control. Before publicly announcing it, IBM took its protocol to the ISO and ANSI standards committees. This resulted in the ISO standard HDLC, high-level data link control\(^3\), and the ANSI standard ADCCP, advanced data communications control procedure. Most of the major vendors participated in this new ‘standards market’ and confused many with their similar acronyms; BDLC, UDLC, and NCRDLC (Burroughs’, Univac’s, and NCR’s versions, respectively) were all largely equivalent to the ISO HDLC standard. Though, the vendor standards included limited lists of options, while HDLC and ADCCP offered numerous options, chosen so essentially everyone could be satisfied\(^4\). LAPB (in the CCITT X.25 Recommendation) and LAPD (as proposed for the D channel on ISDN networks) are basically subsets of HDLC, with specified options and minor additions, as will be shown later. Furthermore, IEEE 802 data link control protocols for local area networks are closely related to HDLC.

It is seen that many of today’s applied data link protocols are deduced from and are mostly a subset of the well-developed ISO HDLC procedures. In the remainder of this Chapter, the HDLC procedures will be discussed in detail.

3.2 HDLC procedures

3.2.1 HDLC standards

HDLC procedures are defined in a set of ISO International Standards. The International Standards 4335 and 7809 are the most important ones as far as the method of operation of HDLC procedures is concerned. Related and more advanced HDLC topics are also standardized which completes the ISO documentation on this subject as follows:

- [IS3309], describing the HDLC frame structure;
- [IS4335], defining the HDLC elements of procedures;
- [IS7478], for the use of multilink procedures;
- [IS7776], which describes the X.25 LAPB-compatible DTE data link procedures (a subset of HDLC);

\(^3\)It is important to note here that HDLC was developed and in practice even before the OSI reference model was standardized. Actually, the definition of the OSI data link layer services and protocols is mainly given in by the early HDLC standards. Ipsuo facto, HDLC had to fit the OSI reference model! However, it will be seen (page 44) that HDLC is still being criticized for a badly defined layering with respect to the underlying physical layer; HDLC accounts for a part of the physical layer’s functionality.

\(^4\)As [Tane 88, p. 254] states: “The nice thing about standards is that you have so many to choose from. Furthermore, if you do not like any of them, you can just wait for next year’s model.”.
3.2 HDLC procedures

- [IS7809], which consolidates the HDLC classes of procedures;
- [IS8471], for address resolution/negotiation in switched environments; and
- [IS8885] defining the XID frame information field content and format.

In the following, only the HDLC methods of operation will be discussed, i.e., a functional description on HDLC frame level will be given. In this, essentially the definitions found in the ISO International Standards 4335 and 7809 will be followed, to a large extent even literally\(^5\). Furthermore, some assumptions will be made as to the optional functions to be implemented and the classes of procedures being employed.

3.2.2 The HDLC frame structure

The HDLC frame structure is defined in [IS3309] and prescribes the relative positions of the various components of the basic frame as well as the bit combination for the frame delimiting sequence (the flag). Since this description is essentially on bit-level and does not treat any major procedural issues, it will only be briefly discussed here for a better understanding.

The basic frame structure

In HDLC, all transmissions are in frames, and each frame consists of the fields shown in Figure 3.2. In this figure

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>8 bits</td>
<td>8 bits</td>
<td>x bits</td>
<td>16 or 32 bits</td>
<td>01111110</td>
</tr>
</tbody>
</table>

Figure 3.2: The HDLC basic frame structure.

- **Flag** is the flag sequence. All frames shall start and end with the flag sequence. All data stations which are attached to the data link shall continuously hunt for this sequence. Thus, the flag is used for synchronization. Furthermore, a single flag may be used as both the closing flag for one frame and the opening flag for the next frame;

- **Address** is the data station address field. In command frames, the address shall identify the data station(s) for which the command is intended. In response frames, the address shall identify the data station from which the response originated;

- **Control** is the control field. The control field indicates the type of commands or responses, and contains sequence numbers, where appropriate. The control field shall be used:
  
  - to convey a command to the addressed data station(s) to perform a particular operation, or

\(^5\)This is done in order to prevent the reader from getting confused by any possible ambiguities in the definitions and description of HDLC procedures. Also, this assures that the protocol description given is as complete as possible, which will show to be very valuable in the next Chapter.
to convey a response to such a command from the addressed data station;

- **Information** is the information field. Information may be any sequence of bits. In most cases it will be linked to a convenient character structure, but, if required, it may be an unspecified number of bits and unrelated to a character structure. In the remainder of this report, the information field – when present – will be assumed to contain an integer number of octets (bytes), which is also advised in Recommendation X.212 ([X.212]) with respect to the format of the user data fields in DLS primitives;

- **FCS** is the frame checking sequence field. Two frame checking sequences (FCSs) are specified in [IS3309]; a 16-bit FCS and a 32-bit FCS. The 16-bit FCS is normally used. The 32-bit FCS will be optional and is for use by prior agreement in those cases that need a higher degree of protection than can be provided by the 16-bit FCS.

To provide transparency, the transmitter of a frame shall examine the frame content between the two flag sequences including the address, control, and FCS fields and shall insert a 0 bit after all sequences of 5 contiguous 1 bits (including the last 5 bits of the FCS) to ensure that a flag sequence is not simulated. The receiver shall examine the frame content and shall discard any 0 bit which directly follows 5 contiguous 1 bits. This technique is also known as 'zero-bit insertion'.

**Address and control field extensions**

Address and control fields may be extended by one or more octects. This is optional in HDLC and will not be discussed here.

**Addressing conventions**

The address field bit pattern 11111111 is defined as the all-station address. The all-station address shall only be used with command frames, and it shall instruct all receiving data stations to accept and action the associated command frame. Any response to a command with the all-station address shall contain the assigned individual address of the data station transmitting the response. The all-station address may be used for all-station polling and to determine the data link level identification of data station(s) when unknown.

The bit pattern 00000000 is defined as the no-station address and shall never be assigned to a data station. It may be used for testing purposes.

**3.2.3 Consolidation of HDLC classes of procedures**

**Configurations and types of data stations**

The International Standard 7809 [IS7809] defines three fundamental classes of procedures: two unbalanced and one balanced. The unbalanced classes apply to both point-to-point and multipoint configurations, as is illustrated in Figure 3.3. The data transmission facilities may be either dedicated or switched. Only the first facility is described here. A characteristic of the unbalanced classes is the existence of a single primary station at one end of the data link plus one or more secondary stations at the other end(s) of the data link. The primary station, which sends commands and receives responses, is alone and ultimately responsible for the organization of data flow and for unrecoverable data link level error conditions, hence the designation ‘unbalanced’. Secondary stations receive commands and send responses.
3.2 HDLC procedures

Figure 3.3: Unbalanced data link configuration.

The balanced class applies to point-to-point configurations only (see Figure 3.4) over either dedicated or switched data transmission facilities. Here also, only the dedicated transmission facilities are implied. A characteristic of the balanced class is the existence of two data stations, called combined stations, on a logical data link. The combined stations, which send both commands and responses, receive both commands and responses, and are both responsible for the organization of its data flow and for unrecoverable data link level error conditions associated with the transmissions that it originates.

For each class of procedures, a method of operation is specified in terms of the capabilities of the basic repertoire of commands and responses that are found in that class. A variety of optional functions is also listed, which will be discussed below.

Furthermore, the International Standard intends to cover one-way, two-way alternate (TWA) and two-way simultaneous (TWS) data communication between data stations which are usually buffered. Only the case of TWS data communication will be considered here.

Figure 3.4: Balanced data link configuration.
Fundamental classes of procedures

Three operational modes and three non-operational modes are defined. The three operational modes are:

- the *normal response mode* (NRM). In NRM, which is an unbalanced data link operational mode, the secondary station shall initiate transmission only as the result of receiving explicit permission to do so from the primary station. After receiving permission, the secondary station shall initiate a response transmission. This response transmission shall consist of one or more frames. The last frame of this response transmission shall be explicitly indicated by the secondary station. Following indication of the last frame, the secondary station shall stop transmitting until explicit permission is again received from the primary station;

- the *asynchronous response mode* (ARM). In ARM, which is also an unbalanced data link operational mode, the secondary station may initiate transmission without receiving explicit permission to do so from the primary station. Such an asynchronous transmission may contain single or multiple frames and shall be used for information field transfer and/or to indicate status changes in the secondary station. For example, the number of the next expected information frame, transition from a ready to a busy condition or vice versa, occurrence of an exception condition; and

- the *asynchronous balanced mode* (ABM). In ABM, which is a balanced data link operational mode, either combined station may send commands at any time and may initiate response frame transmission(s) without receiving explicit permission to do so from the other combined station. Such an asynchronous transmission may contain single or multiple frames and shall be used for information field transfer and/or to indicate status changes in the combined station. For example, the number of the next expected information frame, transition from a ready to a busy condition or vice versa, occurrence of an exception condition.

The three non-operational modes are:

- the *normal disconnected mode* (NDM);

- the *asynchronous disconnected mode* (ADM); and

- the *initialization mode* (IM), which is not considered here.

The disconnected modes (NDM and ADM) differ from the operational modes in that the secondary/combined station is logically disconnected from the data link, i.e., no information or supervisory frames are transmitted or accepted. These two disconnected modes are provided to prevent a secondary/combined station from appearing on the data link in a fully operational mode during unusual situations or exception conditions since such operation could cause unintended contention in ARM, sequence number mismatch between the primary station and the secondary station or between combined stations, or ambiguity in the primary/combined station as to the status of the secondary station/other combined station.

The three operational modes designate the three fundamental classes of procedures to be: the *unbalanced operation normal response mode class* (UNC), the *unbalanced operation asynchronous response mode class* (UAC), and the *balanced operation asynchronous balanced mode class* (BAC). These classes are shown in Figure 3.5.
Figure 3.5: HDLC classes of procedures.
For each class, the basic repertoire of commands and responses is shown, utilizing single octet addressing, unextended control field format, and a 16-bit FCS. Fourteen optional functions are available to modify the fundamental classes of procedures. These optional functions are obtained by the additions or deletions of commands and responses to or from the basic repertoire, or by the use of alternate address or control field formats or alternate frame checking sequences. The classes of procedures and the optional functions shall be indicated by specifying the designation of the class (either UNC, UAC, or BAC) plus the number(s) of the accompanying optional functions. It should be noted, however, that only the basic repertoire is subject of study in this report, and, hence, the optional functions will not be described further.

3.2.4 Elements of HDLC procedures

Now the fundamental classes of procedures are described, it is about time to define the exact procedures of HDLC operation. International Standard 4335 [IS4335] gives the most extensive and detailed description of these procedures (43 pages!) and, hence, the following will be largely a summary of it.

HDLC control field formats

In HDLC, three types of frames are distinguished:

- the information transfer format frame (I). The function of the I command and response shall be to transfer sequentially numbered frames, each containing an information field, across the data link. The control field of I frames will be encoded as shown in Figure 3.6. The explanation of the various parameters will be given in the next Section;

- the supervisory format frame (S). S commands and responses shall be used to perform numbered supervisory functions, such as acknowledgement, polling, temporary suspension of information transfer, or error recovery. The encoding of the S format command/response control field shall be as shown in Figure 3.7. Only the receive ready (RR) and receive not ready (RNR) S commands and responses will be discussed here (basic repertoire!);

---

6 Option 11 is applicable to the balanced class of procedures only.
7 For example, X.25 LAPB is compatible with BAC 2,8 and BAC 2, 8, 10.
3.2 HDLC procedures

Figure 3.7: Supervisory format of control field bits.

- the unnumbered format frame (U). U commands and responses shall be used to extend the number of data link control functions. From Figure 3.8 it can be seen that five 'modifier bits' are provided which allow up to 32 additional command functions and 32 additional response functions to be defined. Thirteen command functions and eight response functions are defined in [IS4335] to allow for the optional functions shown in Figure 3.5. In this report, only the set normal response mode (SNRM), the set asynchronous response mode (SARM), the set asynchronous balanced mode (SABM), and the disconnect (DISC) unnumbered commands will be used. The U responses will be limited to unnumbered acknowledgment (UA), disconnected mode (DM), and frame reject (FRMR).

HDLC parameters

The following parameters are important for the use of HDLC procedures:

- modulus. Each I frame shall be sequentially numbered with a number which may have the value 0 to modulus minus one inclusive, where modulus is the modulus of the sequence numbers. The modulus equals 8 (or 128, in the extended case). The sequence numbers cycle through the entire range. The maximum number of sequentially numbered I frames that a primary, secondary or combined station may have outstanding, i.e., unacknowledged, at any given time shall never exceed one less than the modulus of
the sequence numbers\textsuperscript{8}.

The number of outstanding I frames may be further restricted by the data station frame storage capability; for example, the number of I frames that can be stored for transmission and/or retransmission in the event of a transmission error;

- frame state variables and sequence numbers. In HDLC operations, each data station shall maintain an independent send state variable $V(S)$ and an independent receive state variable $V(R)$ for the I frames it sends to and receives from another data station. Each secondary station shall maintain a $V(S)$ for the I frames it transmits to the primary station, and a $V(R)$ for the I frames it correctly receives from the primary station. In the same manner, the primary station shall maintain an independent $V(S)$ and $V(R)$ for I frames sent to and received from, respectively, each secondary station on the data link. Each combined station shall maintain a $V(S)$ for the I frames it transmits to the other combined station, and a $V(R)$ for the I frames it correctly receives from the other combined station.

The send state variable denotes the sequence number of the next in-sequence I frame to be transmitted. The send state variable can take a value 0 to modulus minus one inclusive. The value of the send state variable shall be incremented by one with each successive I frame transmission, but shall not exceed $N(R)$ of the last received frame by more than modulus minus one.

Only I frames shall contain $N(S)$, the send sequence number of transmitted frames. Prior to transmission of an in-sequence I frame, $N(S)$ shall be set equal to the value of the send state variable.

The receive state variable denotes the sequence number of the next in-sequence I frame expected to be received. The receive state variable can take the value 0 to modulus minus one inclusive. The value of the receive state variable shall be incremented by one on receipt of an error-free, in-sequence I frame whose send sequence number $N(S)$ equals the receive state variable.

All I frames and S format frames shall contain $N(R)$ and, prior to transmission of an I frame or S format frame, the $N(R)$ shall be set equal to the current value of the receive state variable. The $N(R)$ indicates that the station transmitting the $N(R)$ has correctly received all I frames numbered up to $N(R) - 1$ inclusive;

- the poll/final bit (P/F). The P/F bit shall serve a function in both command and response frames. In command frames, the P/F bit is referred to as the P bit. In response frames, it is referred to as the F bit.

\textit{The P bit.} The P bit set to 1 shall be used to solicit a response frame with the F bit set to 1 from the secondary/combined station.

On a data link, only one frame with a P bit set to 1 shall be outstanding in a given direction at a given time. Before a primary/combined station issues another frame with the P bit set to 1, it shall have received a response frame from the secondary station/combined station with the F bit set to 1. If no valid response

\textsuperscript{8}This restriction is to prevent any ambiguity in the association of transmitted I frames with sequence numbers during normal operation and/or error recovery action.
frame is obtained within a system-defined time-out period, the retransmission of a command with the P bit set to 1 for error recovery purposes shall be permitted.

In NRM, the P bit shall be set to 1 to solicit response frames from the secondary station. The secondary station shall not transmit until it receives a command frame with the P bit set to 1. The secondary station may send I frames upon receipt of an I frame with the P bit set to 1, or upon the receipt of an RR frame with the P bit set to 1.

In ARM and ABM, the P bit set to 1 shall be used to solicit a response, at the earliest respond opportunity, with the F bit set to 1.

- The F bit. A response frame with the F bit set to one shall be used by the secondary/combined station to acknowledge the receipt of a command frame with the P bit set to 1.

In NRM, if the right to transmit was acquired by the receipt of a P bit set to 1, then the secondary station shall set the F bit to 1 in the last frame of its response transmission. Following transmission of the last frame of its response transmission, the secondary station shall stop transmitting until either a subsequent command frame with a P bit set to 1 is received.

In ARM and ABM, the secondary station and the combined station, respectively, may transmit response frames with the F bit set to 0 at any respond opportunity on an asynchronous basis. Following the receipt of a command frame with the P bit set to 1, the secondary/combined station shall initiate transmission of a response frame with the F bit set to 1 at the earliest respond opportunity. In ARM and ABM, the transmission of a response frame with the F bit set to 1 shall not require the secondary station or the combined station, respectively, to stop transmitting response frames. Additional response frames may be transmitted following the frame which had the F bit set to 1. Thus, in ARM and ABM, the F bit shall not be interpreted as the end of transmission by the secondary or combined station, respectively; it shall only be interpreted as indicating the response frame from the secondary/combined station sent as a reply to the previous command frame received with the P bit set to 1.

In ABM, if a combined station receives a command with the P bit set to 1, transmission of a response with the F bit set to 1 shall take precedence over transmission of commands, with the exception of the mode-setting commands, i.e., SABM and DISC.

- The use of the P/F bit to assist in error recovery. As the P and F bits set to 1 are always exchanged as a pair, the N(R) contained in a received frame with the P bit or F bit set to 1 can be used to detect that I frame retransmission is required. This capability provides early detection of I frames not received by the remote data station and indicates the frame sequence number where retransmission shall begin. This capability is referred to as checkpointing and will be explained further in the next Section.
3.2.5 Description of the HDLC procedures in TWS unbalanced operation (UNC and UAC) using the basic repertoire of commands and responses

Setting up the data link

If there has not been any data link set up yet or if data link set up to a secondary station is not yet completed, the secondary station will be in NDM (or ADM). In this mode, the secondary station is logically disconnected from the data link, i.e., no I or S frames are transmitted or accepted. The secondary station capability in a disconnected mode shall be limited to accepting and responding to one of the appropriate mode-setting commands (SNRM, SARM, and DISC), and transmitting a DM response at a respond opportunity. A secondary station in a disconnected mode (NDM or ADM) shall, as a minimum capability, be capable of generating a DM response with the F bit set to 1 in response to a command frame received with the P bit set to 1. Any commands, other than the mode-setting commands, received with the P bit set to 0 shall be ignored by the secondary station. Furthermore, a secondary station in a disconnected mode shall not establish a frame reject exception condition (see also page 37).

The primary station shall initialize the data link with a secondary station by sending a SNRM (or SARM\(^9\)) command and shall start a response time-out function. The addressed secondary station, upon receiving the SNRM (or SARM) command correctly, shall send the UA response at its first opportunity to send and shall set its send and receive state variables to 0. If the SNRM (or SARM) command has the P bit set to 1, the UA response shall have the F bit set to 1. If the SNRM frame has the P bit set to 0, then the secondary station shall wait until a command frame is received with the P bit set to 1 and shall then respond with a single UA frame with the F bit set to 1. In ARM, the UA frame may be followed by additional secondary station transmissions, if pending. If the UA response is received correctly by the primary station, the data link setup to the addressed secondary station is complete, and the primary station shall set its send and receive state variables for that secondary station to 0 and shall stop the response time-out function. If, upon receipt of the SNRM (or SARM) command, the secondary station determines that it cannot enter the indicated mode, it shall send the DM response and sustain the NDM (or ADM). If the DM response is received correctly by the primary station, the primary station shall stop its response time-out function.

If the SNRM (or SARM) command, UA response or DM response is not received correctly, it shall be ignored. The result will be that the primary station's response time-out function will run out, and the primary station may resend the SNRM (or SARM) command and restart the response time-out function. This action may continue until a UA or DM response has been received correctly or until recovery action takes place at a higher level. Recovery action at a higher level will, generally, take place if the primary station has resent the SNRM (or SARM) command a system-predefined number of times (this rule is not part of the International Standard, but follows from empirical considerations).

Disconnecting the data link

The primary station shall disconnect the data link with a secondary station by sending a DISC command and shall start a response time-out function. The addressed secondary station,

\(^9\)Only one secondary station at any time shall be put in ARM.
upon receiving the DISC command correctly, shall send a UA response at its first respond opportunity and shall enter the NDM, or the ADM, as appropriate. If, upon receipt of the DISC command, the addressed secondary station is already in the disconnected mode, it shall send the DM response. The UA or DM response shall have the F bit set to 1 if the DISC command has the P bit set to 1. The primary station, upon receiving a UA or DM response to a sent DISC command, shall stop its response time-out function.

If the DISC command, UA response or DM response is not received correctly, it shall be ignored. This will result in the expiry of the primary station's response time-out function, and the primary station may resend the DISC command and restart the response time-out function. This action may continue until either the UA or a DM response has been received correctly or until recovery action takes place at a higher level. Recovery action at a higher level will, generally, take place if the primary station has resent the DISC command a system-predefined number of times.

The information transfer phase

Regarding the exchange of I and S frames, following the rules of the essential HDLC parameters (P/F bit, sequence numbers and state variables) is very important for proper operation. Hence, the reader is strongly advised to remind the definitions and rules given in Section 3.2.4. For the sake of clarity, it is recalled here that, following data link set up, both V(S) and V(R) shall be set to zero. Furthermore, the maximum length of I frames shall be a system-defined parameter.

If a data station is ready to send an I frame numbered N(S), where N(S) is equal to the last received acknowledgment plus modulus minus 1, the data station shall not send the I frame, but shall wait for further acknowledgments or for a time-out recovery action, as appropriate to that station. When and how to apply such a recovery action is described below.

After a data station receives correctly an in-sequence I frame that it can accept, it shall increment its receive state variable V(R), and, at its next opportunity to send, take one of the following actions:

- if information is available for transmission and the remote data station is ready to receive, it shall act as described above and acknowledge the received I frame(s) by setting N(R) in the control field of the next transmitted I frame to the value of V(R);
- if information is not available for transmission, but the data station is ready to receive I frames, the data station shall send an RR frame and acknowledge the received I frame(s) by setting N(R) to the value of V(R);
- if the data station is not ready to receive any further I frames, the data station may send an RNR frame and acknowledge the received I frame(s) by setting N(R) to the value of V(R).

If the data station is unable to accept the correctly received I frame(s), V(R) shall not be incremented. The data station may send an RNR frame with the N(R) set to the value of V(R).

This means actioning a SNRM (or SARM) command or receiving a valid response to such a command frame.

If the maximum number of sequentially number I frames that a primary or secondary station may have outstanding at any given time is further restricted, say to a fixed number K, then 'modulus minus 1' should be replaced by this number.
Error condition reporting and recovery

The following procedures are available to effect recovery following the detection/occurrence of an exception condition at the data link level. The exception conditions described here are those situations which may occur as the result of transmission errors, data station malfunction or operational situations.

- **The busy condition.** The busy condition shall result when a data station is temporarily unable to receive, or unable to continue to receive, I frames due to internal constraints, e.g., receive buffering limitations. In this case, an RNR frame shall be transmitted with the N(R) number of the next I frame that is expected. Traffic awaiting transmission may be transmitted from the busy data station prior to, or following, the RNR frame. The continued existence of a busy condition shall be reported by retransmission of an RNR frame at each P/F frame exchange.

A data station receiving an RNR frame when in the process of transmitting shall stop transmitting I frames at the earliest possible time. It is suggested that a secondary station in NRM return a frame with the F bit set to 1 before suspending transmission. A secondary in ARM shall perform a response time-out before resuming transmission.

Indication that a busy condition has cleared and that I frames will now be accepted shall be reported by the transmission of an RR, SNRM, SARM or VA frame with or without the P bit set to 1. Clearance of a busy condition at a primary station shall also be indicated by the transmission of an I frame with the F bit set to 1. Clearance of a busy condition at a secondary station shall also be indicated by the transmission of an I frame with the F bit set to 1.

- **N(S) sequence error.** An N(S) sequence error exception condition shall occur in the receiver when an I frame received error-free (no FCS error) contains an N(S) that is not equal to the receive state variable at the receiver. The receiver shall not acknowledge (i.e., not increment its receive state variable) the frame causing the sequence error of any other I frames which may follow until an I frame with the correct N(S) is received. The information field of all I frames received whose N(S) does not equal the receive state variable shall be discarded.

A primary or secondary station which receives one or more I frames having sequence errors, but which are otherwise error-free, shall accept the control information contained in the N(R) field and the P/F bit to perform data link control functions; for example, to receive acknowledgment of previously transmitted I frames to cause a secondary station to respond (P bit set to 1), and, in NRM, to detect that the secondary station is terminating transmission (F bit set to 1). Therefore, the retransmitted I frame may contain an N(R) field and/or P/F bit information that are updated and different from those contained in the originally transmitted I frame.

Following the occurrence of a sequence error, the following means are available for initiating the retransmission of lost I frames or those with errors.

- **Checkpoint recovery** (see also page 33). When a data station receives a frame with the P/F bit set to 1, it shall initiate retransmission of unacknowledged I frames previously transmitted with sequence numbers that are less than the V(S) value that was current at the time of transmission of the last frame with the F/P
3.2 HDLC procedures

bit, respectively, set to 1. Retransmission shall start with the oldest numbered unacknowledged I frame. I frames shall be retransmitted sequentially (to preserve the order of transmission). New I frames may be transmitted if they become available.

Checkpoint retransmission shall not be initiated under the following conditions:

* if a P/F bit set to 1 is received in an U format frame;
* if, after sending a frame with the P/F bit set to 1, a data station receives an acknowledgment to that frame before receiving the correspondent frame with the F/P bit set to 1.

— *Time-out recovery.* In the event that the remote data station, as a result of transmission error, does not receive (or receives and discards) a transmission consisting of a single I frame or the last I frame(s) in a sequence of I frames, it will not detect an out-of-sequence exception condition. The data station which transmitted the unacknowledged I frame(s) shall, following the completion of a system-defined time-out period, take appropriate recovery action to determine the point at which retransmission shall begin.

A primary station should enquire status with an S frame (RR or RNR). When a secondary station has a respond opportunity, and an optional time-out function for unacknowledged I frames has run out, and no new I frames are available, then the secondary station should transmit only the last I frame and should wait until status is received from the primary station.

— *FCS error.* Any frame received with a FCS error shall not be accepted by the receiver and shall be discarded. At a secondary station, no action shall be taken as the result of that frame. At a primary station, if the frame with the FCS error was a response frame with the F bit set to 1, a resulting time-out function shall occur in the primary station prior to initiating recovery action.

— *Command/response frame rejection.* A command/response rejection exception condition shall be established upon the receipt of an error-free frame which contains an undefined or not implemented command/response in the control field, an invalid frame format (a frame containing an information field when no information field is permitted by the associated control field), an invalid N(R) from the primary station or an information field which exceeds the maximum information field length which can be accommodated by the receiving data station.

At a primary station, this exception condition shall be subject to recovery/resolution at a higher level. In the case of an invalid N(R), recovery shall include, at least, the issuance of a mode-setting command.

At a secondary station, this exception condition shall be reported by a FRMR response for appropriate primary station action. The FRMR response shall contain an information field containing the reason for the frame rejection. The information field shall contain the fields shown in Figure 3.9. The functions of these fields shall be as follows:

— the rejected frame control field shall be the control field of the received frame which caused the frame rejection exception condition;
— N(S) shall be the current value of the send state variable at the sender of the FRMR response;
Figure 3.9: Format of the FRMR frame information field.

- C/R set to 1 shall indicate that the frame which caused the frame rejection exception condition was a response frame, and C/R set to 0 shall indicate that the frame was a command frame;
- N(R) shall be the current value of the receive state variable at the sender of the FRMR response;
- w shall indicate that the control field received and returned in bits 1 to 8 inclusive was undefined or not implemented;
- x set to 1 shall indicate that the control field received and returned in bits 1 to 8 inclusive was considered invalid because the frame contained an information field which is not permitted with this command or response. Bit w shall be set to 1 in conjunction with this bit;
- y set to 1 shall indicate that the information field received exceeded the maximum information field length which can be accommodated by the sender of the FRMR response;
- z set to 1 shall indicate that the control field received and returned in bits 1 to 8 inclusive contained an invalid N(R).

The w, x, y and z bits in the information field of the FRMR response may all be set to 0, indicating an unspecified rejection of the frame for one or more of the conditions cited above. If required, the information field contained within the FRMR response may be padded with 0 bits so as to end on any convenient byte-dependent boundary.

Once a secondary station has established a FRMR exception condition, no additional I frames shall be accepted, except for examination of the state of the P bit and the value of the N(R) field, until the condition is cleared by the primary station issuing a mode-setting command. Furthermore, after sending a FRMR response, the secondary station shall stop transmitting I frames if the frame reject exception condition was caused by an invalid N(R), since its direction of transmission is affected. It may, however, if the frame reject exception condition was caused by a command that is undefined or not implemented or an information field which is too long, continue sending I frames. This is because the opposite direction of transmission is affected. The FRMR response shall be repeated at each respond opportunity until recovery is effected by the primary station.

The primary station receiving the FRMR response shall be responsible for initiating the appropriate mode-setting or resetting corrective action by initializing one or both directions of transmission, using the SNRM, SARM or DISC command, as applicable.
3.2 HDLC procedures

3.2.6 Description of the HDLC procedures in TWS balanced operation (BAC) using the basic repertoire of commands and responses

Setting up the data link

If there has not been any data link set up yet or if data link set up to a combined station is not yet completed, the combined station will be in ADM. In this mode, the combined station is logically disconnected from the data link, i.e., no I or S frames are transmitted or accepted. The combined station capability, as a receiver of commands in ADM, shall be the same as that stated above for the secondary station (accepting and responding to SABM and DISC commands, transmitting a DM response as a minimum capability, and not establishing a frame reject exception condition). In addition, since the combined station has the ability to transmit commands at any time, it may transmit an appropriate mode-setting command.

So, either combined station may take the initiative to initialize the data link. It shall send the SABM command and start a response time-out function. The other combined station, upon receiving the SABM command correctly, shall send a UA response and reset both its send and receive state variable to 0. The UA frame shall have the F bit set to 1, if the SABM command was sent with the P bit set to 1. The UA frame may be followed by additional combined station transmissions, if pending. If the UA response is received correctly, the data link setup shall be complete, and the initiating combined station shall set both its send and receive state variables to 0, stop the response time-out function, and enter the indicated mode. If, upon receipt of the SABM command, a combined station determines that it cannot enter the indicated mode, it shall send the DM response. If the DM response is received correctly, the initiating combined station shall stop the response time-out function.

If a SABM command, UA response or DM response is not received correctly, it shall be ignored. The result will be that the response time-out function will run out in the combined station which originally sent the SABM command and that combined station may resend the SABM command and restart the response time-out function. This action may continue until a UA response or a DM response has been received correctly or until error recovery takes place at a higher level. Recovery action at a higher level will, generally, take place if the combined station has resent the SABM command a system-predefined number of times (this rule is not part of the International Standard, but follows from empirical considerations).

Disconnecting the data link

Either combined station may take the initiative to disconnect the data link. It shall send the DISC command and start a response time-out function. The other combined station, in an operational mode, upon receiving the DISC command correctly shall send a UA response and enter the ADM. If, upon receipt of the DISC command, the other combined station is already in the disconnected mode, it shall send the DM response. The UA or DM response shall have the F bit set to 1 if the DISC command has the P bit set to 1. The initiating combined station, on receiving the UA or DM response to a sent DISC command correctly, shall stop its response time-out function.

If the DISC command, UA response or DM response is not received correctly, it shall be ignored. This will result that the response time-out function will expire in the combined station which originally sent the DISC command unless a separate mode-setting command is received, in which case the response time-out function may be stopped. This combined station may resend the DISC command and restart its response time-out function. This action may
continue until either the UA or a DM response has been received correctly, a DISC command has been received correctly, or until recovery action takes place at a higher level. Recovery action at a higher level will, generally, take place if the primary station has resent the DISC command a system-predefined number of times.

Simultaneous mode-setting attempts (contention)

When a combined station issues a mode-setting command (SABM or DISC) and, before receiving an appropriate response, receives a mode-setting command from the remote combined station, a contention situation has developed. Contention situations shall be resolved in the following manner. When the sent and received mode-setting commands are the same, each combined station shall either enter the indicated mode immediately or defer entering the indicated mode until receiving a UA response. In the latter case, if the UA response is not received, the combined station may enter the mode when the response time-out function expires, or the combined station may re-issue the mode-setting command. When the mode-setting commands are different, each combined station shall enter ADM and issue a DM response at the earliest respond opportunity. In the case of a DISC command contention with a different mode-setting command, no further action is required.

The information transfer phase

Regarding the exchange of I and S frames, following the rules of the essential HDLC parameters (P/F bit, sequence numbers and state variables) is very important for proper operation. Hence, the reader is strongly advised to remind the definitions and rules given in Section 3.2.4. For the sake of clarity, it is recalled here that, following data link set up\textsuperscript{12}, both \( V(S) \) and \( V(R) \) shall be set to zero. Furthermore, the maximum length of I frames shall be a system-defined parameter.

If a combined station is ready to send an I frame numbered \( N(S) \), where \( N(S) \) is equal to the last received acknowledgment plus modulus minus 1, the combined station shall not send the I frame, but shall wait for further acknowledgments or for a time-out recovery action, as appropriate to that station\textsuperscript{13}. When and how to apply such a recovery action is described below. The decision whether to send an I frame as a command or as a response, i.e., to use the remote or local address to indicate a P or F bit, respectively, shall depend upon the need to acknowledge a received P bit set to 1 or F bit set to 1.

After a combined station receives correctly an in-sequence I frame that it can accept, it shall increment its receive state variable \( V(R) \), and, at its next opportunity to send, take one of the following actions:

- if information is available for transmission and the remote combined station is ready to receive, it shall act as described above and acknowledge the received I frame(s) by setting \( N(R) \) in the control field of the next transmitted I frame to the value of \( V(R) \);

\textsuperscript{12}This means actioning a SABM command or receiving a valid response to such a command frame.

\textsuperscript{13}If the maximum number of sequentially number I frames that a primary or secondary station may have outstanding at any given time is further restricted, say to a fixed number \( K \), then ‘modulus minus 1’ should be replaced by this number.
3.2 HDLC procedures

- if information is not available for transmission, but the combined station is ready to receive I frames, the combined station shall send an RR frame and acknowledge the received I frame(s) by setting N(R) to the value of V(R);

- if the combined station is not ready to receive any further I frames, the combined station may send an RNR frame and acknowledge the received I frame(s) by setting N(R) to the value of V(R).

If the combined station is unable to accept the correctly received I frame(s), V(R) shall not be incremented. The combined station may send an RNR frame with the N(R) set to the value of V(R).

The I of S frame transmitted will be either a command or a response depending on whether a P bit set to 1 or an F bit set to 1, respectively, is required. If the transmission of a P bit or F bit set to 1 is not required, the acknowledgment frames may be either commands or responses.

Error condition reporting and recovery

The following procedures are available to effect recovery following the detection/occurrence of an exception condition at the data link level. The exception conditions described here are those situations which may occur as the result of transmission errors, data station malfunction or operational situations.

- The busy condition. The same rules and definitions apply as in the unbalanced operation case. In addition, a combined station shall perform a command time-out before resuming transmission. Furthermore, indication that a busy condition has cleared and that I frames will now be accepted shall be reported by the transmission of an RR, SABM or UA frame with or without the P bit set to 1. Clearance of a busy condition at a combined station shall also be indicated by the transmission of an I frame with the F bit set to 1.

- N(S) sequence error. The same rules and definitions apply as in the unbalanced operation case. Furthermore, a combined station which receives one or more I frames having sequence errors, but which are otherwise error-free, shall accept the control information contained in the N(R) field and the P/F bit to perform data link control functions; for example, to receive acknowledgment of previously transmitted I frames to cause a combined station to respond (P bit set to 1). Therefore, the retransmitted I frame may contain an N(R) field and/or P/F bit information that are updated and different from those contained in the originally transmitted I frame.

Following the occurrence of a sequence error, the following means are available for initiating the retransmission of lost I frames or those with errors.

- Checkpoint recovery (see also page 33). The same rules and definitions apply as in the unbalanced operation case. Moreover, if any frame with the P bit set to 1 is received by a combined station, checkpoint retransmission shall be inhibited.

- Time-out recovery. The same rules and definitions apply as in the unbalanced operation case. A combined station should, like a primary station in the unbalanced operation mode, enquire status with an S frame (RR or RNR).
42 Protocols of the data link layer

• **FCS error.** Any frame received with a FCS error shall not be accepted by the receiver and shall be discarded. At a combined station, no action shall be taken as the result of that frame. Though, if the frame with the FCS error was a response frame with the F bit set to 1, a resulting time-out function shall occur in the combined station prior to initiating recovery action.

• **Command/response frame rejection.** A command/response rejection exception condition shall be established upon the receipt of an error-free frame which contains an undefined or not implemented command/response in the control field, an invalid frame format (a frame containing an information field when no information field is permitted by the associated control field), an invalid N(R) from the other combined station or an information field which exceeds the maximum information field length which can be accommodated by the receiving data station.

At a combined station, this exception condition shall be dealt with in either two ways:

- the combined station may follow a course of action similar to that described for a primary station (page 37), where the exception condition is resolved as a higher level function. The combined station shall issue a mode-setting command, as appropriate, as part of this recovery action;

- the combined station may follow a course of action similar to that described for a secondary station and request that the other combined station resolve the exception condition and effect the required recovery. Furthermore, after sending a FRMR response\(^{14}\), the combined station shall stop transmitting I frames if the frame reject exception condition was caused by an invalid N(R), since its direction of transmission is affected. It may, however, if the frame reject exception condition was caused by a command or response that is undefined or not implemented or an information field which is too long, continue sending I frames. This is because the opposite direction of transmission is affected.

• **Contention situations (ABM).** Contention may occur during a mode-setting action. This contention situation shall be resolved through the use of different value time-out functions in each data station. The time-out function employed by the one combined station shall be greater than that employed by the other combined station so as to permit contention situations to be resolved in favour of the specified combined station.

3.2.7 **Time-out function considerations**

Annex A to [IS4335] considers some possible provisions for the use of time-out functions as discussed above. The Annex does not form part of the International Standard, but should be considered as a guideline. It is stated once more that the duration of a time-out period is system-dependent and subject to bilateral agreement.

The considerations (for TWS communications) are as follows:

• **Primary/combined station command reply time-out function**

  - NRM:

    * Start condition: Transmission of a frame with the P bit set to 1;

\(^{14}\)The same rules apply for the format of the FRMR response as in the unbalanced operation case.
3.2 HDLC procedures

* Restart condition: Receipt of an error-free frame with the F bit set to 0;
* Stop condition: Receipt of an error-free frame with the F bit set to 1;

- ARM and ABM:
  * Start condition: Transmission of a frame with the P bit set to 1;
  * Stop condition: Receipt of an error-free frame with the F bit set to 1;

- NDM and ADM:
  * Start condition: Transmission of a frame with the P bit set to 1;
  * Stop condition: Receipt of an error-free frame with the F bit set to 1.

- Primary/secondary/combined station I frame reply time-out function

  - NRM:
    * Not used;
  
  - ARM and ABM:
    * Start condition: Transmission of an I frame;
    * Stop condition: Receipt of an error-free frame with the expected N(R);

  - NDM and ADM:
    * Not used.

- Secondary/combined station command request time-out function

  - NRM and NDM:
    * Not used;

  - ARM, ADM and ABM:
    * Start condition: Transmission of an unnumbered response frame which requests a command;
    * Receipt of an error-free command frame.

It will be shown in the next Chapter that this list is not limitative and that some other conditions on which to start a time-out function have to be added.

To complete the description of HDLC procedures, Annex B of [IS4335] gives a rather extensive list of examples of the use of commands and responses. Also, this Annex does not form part of the International Standard, though it could be used to create a better understanding of several HDLC procedural mechanisms.

3.2.8 Limitations of HDLC

HDLC and its relatives have represented and, for many applications, still represent the state of the art in data link control procedures. Under most conditions they are more flexible and more efficient than the other protocols we have discussed. Nevertheless, HDLC has weaknesses:

1. one limitation results from the fact that HDLC has so many options. Different HDLC devices may not be able to communicate if they do not support the same set of options;
2. HDLC's limitation to a single address is adequate in point-to-point communications or multipoint communications with a single primary station, but it is not adequate in multipoint broadcast environments with all stations treated alike. Both destination and source addresses are needed in such situations;

3. HDLC has also been criticized on the basis that its layering is not clear. In the OSI layering context, its use of flags and of zero insertion are considered to be physical layer features. It is primarily connection-oriented, with acknowledgment of all frames (although it provides limited support for unacknowledged data transfer, using unnumbered data frames), so it is not fully suitable for connectionless services;

4. the portions of HDLC frames most susceptible to errors are the frame-delimiting flags. The FCS does not check these flags and an error in transmitting either flag destroys the framing mechanism. An error in the opening flag may mean a complete frame is lost since character synchronization is never obtained; an error in the closing flag may mean the opening flag of the following frame is interpreted as the closing flag of the previous frame. Such erroneous 'frames' are, with high probability, detected by the FCS, but error recovery for two frames is necessary under such conditions;

5. ambiguities may occur when the unnumbered format is used with alternating P and F bits in commands and responses. Under these circumstances, if an unnumbered command with the P bit set to 1 receives no response and a second unnumbered command with the P bit set to 1 is sent after a time-out, there is no way to tell whether an unnumbered response with the F bit set to 1 is a response to the first or to the second command.
Chapter 4

Formal specification of the HDLC controller

‘Pretend that you’re Hercule Poirot: examine all clues, and deduce the truth by order and method.’

Donald E. Knuth

The previous two Chapters presented a specification of the data link services and the data link control procedures (i.e., the HDLC basic repertoire), respectively. These specifications descend from, and still remain, documents in natural language, elucidated by graphical means, such as time-sequence diagrams. The HDLC documents are perhaps the best example that such informal specifications are, on one hand, inevitable from an historical point of view and very complete, but, on the other hand, rather inescrutable, ambiguous\(^1\), and untransparent. Since it is one of the main objectives of this research to specify the exact behaviour of and interaction\(^2\) between the data link services and control procedures (on DLPDU level), this informality will evidently show to be undesirable, inadequate, and even inexpedient, because it allows too many erroneous and bad designs.

4.1 The use of formal description techniques

4.1.1 The use of formality in the design process

Therefore, the next step in the design process is to introduce formalization of these specifications, describing both the data link services’ and control procedural behaviour as well as their interaction according to the OSI reference model. Where possible, these specifications should be formal, though it should be noted that formality alone is inadequate, because it

\(^1\) This is usually referred to as ‘the inherent ambiguity of natural language’. Understanding can be said to be based on an interpretation model within the reader (a human being).

\(^2\) This is the translation problem stated in Section 2.1.
46 Formal specification of the HDLC controller

can easily lead to incomprehensible detail. A human being must interpret the representation according to the interpretation model of the language (its semantics), which is usually quite different from intuitive interpretation. Furthermore, a purely formal specification must contain all the information required by the interpretation model; one cannot skip a piece of information relying on common understanding backgrounds and this may lead to far too complex representations.

On the contrary, by formal specification, the human being is forced by the representation rules to fill in all significant details required by the interpretation model, allowing one and only one interpretation. Another property and inherent advantage of purely formal specification is that it can be analyzed by a computer: it can be used as a prototype, it can be tested on completeness, consistency and/or conformance, and it can be transformed into program code for automated implementation, for example.\(^\text{3}\)

It is seen that the use of formality in the design process has its advantages and disadvantages, and so, it should be employed having the aforementioned characteristics in mind. Hence, the most important advantages of using a formal description technique should be efficiently utilized, while avoiding possible disadvantages to take effect. Therefore, the main objectives of introducing formality in our specifications are defined to be (in order of importance):

1. to avoid ambiguity of interpretation. Regarding to this objective, a specification should be formal to a certain level, assuring independency of any interpretation model except from the specification language semantics. This will show to be evident in the remainder of this Chapter;

2. to provide an abstract implementation to ensure compatibility. It is not a primary objective here to specify towards hardware, but to make sure that the abstract implementation represented by the specification should ensure compatibility in any derived physical implementation.\(^\text{4}\);

3. to serve as a visual representation for illustration of the followed procedures. Since the two objectives above and this one are likely to be conflicting requirements, some means have to be found to sufficiently satisfy all of them. Still, the creation of a fixed interpretation framework has the highest priority.

At last, another specification objective is

4. to specify more accurately the interfaces between several levels in the OSI reference model. Since two separate and informal specifications are crucial in this case (the HDLC International Standards and the CCITT Recommendations regarding the definition of data link services), it is important to combine these in one specification within the same specification methodology.

\(^\text{3}\)Any of these examples could be specification objectives on their own.

\(^\text{4}\)Of course, this objective can only be fulfilled if the first objective mentioned is also fulfilled. Furthermore, it is obvious that this is also an objective given in by the OSI reference model ("... removal of any technical impediment from communication between systems.").
4.1 The use of formal description techniques

4.1.2 The CCITT Specification and Description Language (SDL)

To fill in these specification goals, several standardized and generally accepted formal description techniques\(^5\) can be fruitfully employed.

- **Estelle** is developed by the ISO and its syntax is very close to that of the PASCAL programming language.
- **LOTOS** stands for Language Of Temporal Ordering Specification and is also standardized by the ISO. It is strongly based on process algebra like Milner's Calculus for Communicating Systems (CCS).
- The **CCITT Specification and Description Language (SDL)** has developed from an informal language to a formalized language defined in CCITT Recommendation Z.100 [Z.100]. It is based on the model of Communicating Extended Finite State Machines (CEFSMs) and it supports both a graphical and a textual representation with a one-to-one translation between them.

[Hulz 88, Chapter 2] summarizes some strong and weak aspects of the different languages. It is seen there that Estelle is strong because of its familiarity with PASCAL, but it has no abstract data typing and is weak because of its closeness to implementation (too many details need to be specified, hence it is not illustrative). LOTOS is a strong language because of its mathematical basis (CCS-alike), but it is not user-friendly, since only a textual form is provided.

SDL, on the other hand, is a human-oriented description technique, providing a well-defined formal use of graphical description with a one-to-one translation to the textual form. Since it is based on Finite State Machines (FSMs), it can be easily understood. SDL is especially designed for the specification and description of telecommunications systems and it provides for some very specific features, like

- the possibility to tackle the complexity of modern telecommunications systems. The level of formality in SDL is not fixed and, hence, more or less abstract specifications can be created, filling in the specification objectives as appropriate;
- since SDL is based on FSMs, it can be easily understood by both computers as well as human beings. The latter is extremely important in the design of telecommunications systems, since it usually is a process involving several working teams (software engineers, hardware engineers, marketeers etc.);
- like in this report, the design of telecommunications systems generally involves the integration of specification parts, produced in different environments and styles (i.e., CCITT Recommendations and ISO International Standards). SDL is able to integrate these specifications, since it supports several design methodologies and offers some interesting design features. The possibility to structure a specification by partitioning and refinement allows for a top-down design;

\(^5\)A distinction should be made here between the terms 'description' and 'specification'. A description specifies the actual behaviour of a system, while a specification deals with a system's required behaviour. It is obvious that this report deals with specifications only. Still, the formal methods used here are commonly referred to as 'formal description techniques'.
• furthermore, SDL makes use of several special techniques and constructs that facilitate the design process, like relative and absolute timers, instantiation of processes, the use of macros and procedures, and signal communication.

In the following Sections, an SDL specification of the data link services and HDLC basic repertoire procedural behaviour will be given. Those not familiar with the basic SDL principals are advised to read Appendix B first.

4.2 Specification of the HDLC byte controller in SDL

4.2.1 Specification strategy

From literature, several attempts to specify the data link services and the HDLC basic repertoire procedural behaviour can be found. The specification given in [ECMA 81] is very complete, but it does not provide for only one interpretation, since the semantics of the specification language used are badly defined. Probably, this specification serves merely illustrative purposes, rather than any other of the specification objectives given above. [Hogr 88], on the other hand, employs SDL for specification of the X.25 LAPB subset of HDLC. Although the specification given there is quite illustrative, still a lot of relevant specification parts are left out and there is no way that this work can possibly ensure compatibility as an abstract implementation. There are not only parts of the specification left to common understanding backgrounds, but some procedures followed do neither conform to the International Standards on HDLC nor to the X.212 Recommendations regarding the definition of data link services. This is not further discussed here.

So, from the precedents in literature one can recognize the importance of using a well-defined specification language (like SDL) together with a firm design methodology. Therefore, a strategic subdivision of design steps should be made. Regarding our specification problem this includes the separate treatment of the following aspects:

• a structure representation. To specify the functional decomposition of the system, the functional relationships between the various system components have to be defined as well as the system’s relationship with its environment;

• a behaviour representation. A behaviour representation describes the exact required (expected) behaviour of the various system components at a lower level of abstraction; and

• quantitative information, including all the information necessary to parameterize a system both to its capabilities and to its performance (e.g., the maximum number of outstanding information frames, the duration of the several timers). Also, the technique of generic system specification will be applied, allowing parameters to be set to a certain value by the system’s environment. This means that those parameters can be implied or are known on an a priori basis within the system boundaries.

The treatment of these aspects in this order presents a top-down design, filling in more and more detail during the specification process.
4.2 Specification of the HDLC byte controller in SDL

4.2.2 Structure representation; the OSI-SDL relationship

Before the actual system specification will be given, something more will have to be said about the relation of SDL to OSI. There are several methods for specifying OSI services and protocols in SDL. With reference to Figure 2.1, it is seen that OSI protocols can be viewed as constituting a vertical or a virtual, horizontal communication. From the same point of view, the modelling of OSI in SDL can be approached in two ways:

- the **horizontal view of protocols**, which is achieved by modelling the virtual, horizontal communication of PDUs as a real communication using a channel substructure in SDL (see [Beli 91, p. 188 ff.]); or

- the **vertical view of protocols**, which is achieved by modelling end-to-end PDU communication as an indirect communication. PDUs from one entity (process) are communicated to another entity (process) indirectly via the underlying service provider.

The latter approach will be followed here, since it fits best to further design steps towards implementation. Examples of this OSI-SDL relationship can be found in [Auli 89], [Beli 89], [Beli 91] and [Hogr 88]. The horizontal view of protocols will not be discussed any further here.

Service specification

Schematically, this vertical view of protocols related to specification in SDL can be modelled in a straightforward manner as a block specification DataLinkService containing two process specifications DLS_A and DLS_B (see Figure 4.1).

![Figure 4.1: The specification of an OSI service in SDL.](image-url)
The users of the data link service are in the environment of the system and can be considered as processes, capable of communicating with the system. Note that the representation of the users is not part of the SDL syntax, and is included only to aid understanding.

A service access point is represented by a channel (DLSAPa and DLSAPb), conveying signals which represent service primitives. A signal may carry values of the sorts given in the signal specification (not shown in this example).

Both local and end-to-end aspects of a service specification are dealt with here. Local behaviour is expressed independently by the process specifications DLS_A and DLS_B. These process types communicate with each other by signals which are internal to the block and are conveyed by the signal route DLroute. End-to-end behaviour is expressed by the mapping, performed by each process type, between service primitives and internal signals on DLroute.

**Protocol specification**

The protocol specification corresponding to the service of Figure 4.1 is modelled by the substructure DataLinkProtocol of the block DataLinkService, in which it is referenced. Its specification is given in Figure 4.2.

![Protocol specification diagram](image)

**Figure 4.2:** The specification of an OSI protocol in SDL.

The block substructure diagram in Figure 4.2 specifies three blocks: entityA, entityB, and PhService. The first two blocks represent data link protocol entities, while the block PhService represents the physical layer service provider. The specification of PhService is analogous to the specification of DataLinkService, and one can see that the specification is fully iterative towards other layers of the OSI reference model.

---

6This mapping is not further discussed, since only local behaviour is of our interest. Furthermore, the process specifications DLS_A and DLS_B can be mirror images if the service is symmetrical.
A protocol entity block can contain one or more process types, depending on the characteristics of the protocol. In this case, two process types, ByteCtrl and BitCtrl, have been chosen. Process type ByteCtrl handles the sending and receiving of PDUs (thus on a byte level), while process type BitCtrl takes care of the transmission of PDUs using the underlying physical layer service. Conceptually, the process types ByteCtrl communicate directly via an implicit channel DLchan, conveying PDUs, but in reality they communicate indirectly via process type BitCtrl and the underlying physical layer service provider.

Since it is not our general goal to describe end-to-end communication\(^7\) and since this end-to-end communication can be derived from the local behaviour of the data link layer entities, only the latter will be specified using the structural decomposition explained in Figure 4.3. Furthermore, only the local behaviour of the process types at PDU level (in the example the process type ByteCtrl) will be described. More precisely, the following parts of the system will be explicitly specified:

- the interface between the DLS user (residing in the environment of the system to be specified) and the data link layer entity, conforming to the X.212 Recommendations. This is effected by the signal communication over the channel DLSAP and appropriate reactions of the data link layer entity, including

- the procedural behaviour (in terms of DLPDUs) of the ByteCtrl part of the data link layer entity, constituting a data station conforming to the HDLC International Standards. This means that the ByteCtrl part functions in terms of both DLSDUs issued by the DLS user, as well as HDLC frames exchanged over

- the interface between the ByteCtrl and the BitCtrl part of the data link layer entity.

\(^7\)Such an end-to-end specification should involve the behaviour given by the time sequence diagrams of Figures 2.3 to 2.11 supplemented with a lower level specification of the course of HDLC frame exchanges.
The interface downwards to the OSI physical layer is not specified, though it will show later that some information exchanged at that level should be known at the level of specification mentioned above.

The system HDLC_Controller

In the Appendix C the complete SDL specification of the HDLC controller is given. It is seen there that the system HDLC_Controller takes the same structural properties as shown above. First, a generic system specification is used to discern between the various types of data stations (functional entities) known by HDLC, namely, the primary station, the secondary station, and the combined station. The use of a generic system specification means that a part of specification is chosen by the value of a certain parameter, i.e., the StTy parameter. For example, if the StTy parameter has the value Sec, the specification for the secondary station is chosen. The value of the StTy parameter is set outside the system environment and is known at system creation time.

Each block of the generic system specification is substructured in the way explained in Figure 4.2, namely in a 'byte controller' and a 'bit controller' part, constituting the aforementioned ByteCtrl and BitCtrl process types. A small modification is made here, since the byte controller and the bit controller part at this stage of the specification are not process types, but blocks containing several processes. The major processes of each byte controller block (for all data station types) are:

- the process TX. This is the transmitter process that takes care of the transmission of HDLC frames. It also checks the arrival of several DLS primitives issued by the DLS user;

- the process RX. This is the receiver process that receives HDLC frames and issues DLS primitives to the DLS user. The processes TX and RX interwork by the exchange of control signals and they are informed about the values of several procedural and system management parameters by

- the process SR, which stands for 'shared resources'. The resources kept in this process are defined as shared values which can be 'viewed' by the processes TX and RX. The process SR provides for the correct administration of these values and can perform some operations on them.

Furthermore, two data management parts are defined for each data station byte controller block: DataMan_TX and DataMan_RX, constituting the data management functions for the transmitter process and for the receiver process, respectively. These channel substructures contain a process InfMan_TX and InfMan_RX, respectively, representing the actual data queues contained by the HDLC controller. These processes are specified informally, since SDL unfortunately has rather poor queue manipulation capabilities.

---

8This is due to the fact that HDLC does not make a clear distinction between OSI data link layer and OSI physical layer functions, as is already mentioned on page 44.

9This is a typical case of choice between formality in the specification (meaning the inclusion of every detail) and visualization of the followed procedures. It could be possible to specify the data queue processes purely formally in SDL, but this would not at all enhance the readability of the specification nor would it add much value to the understanding of the HDLC procedures that form our main specification target; an informal (textual) specification will suffice here.
At last, the block `Primary_Byte_Controller` contains a process `Setup` which is responsible for setting up connections with secondary stations. Setting up these connections is effected by the instantiation of the primary processes `TX` and `SR` on the receipt of a `CONNreq` primitive, while disconnection is effected by the killing of all running primary processes related to that primary-to-secondary connection.

### 4.2.3 Behaviour representation

In the previous Section, the global relationships between processes and the global functional decomposition of the system were illustrated. It was also mentioned briefly how the various processes interwork by communicating signals and sharing variables and how their mutual positions and responsibilities are roughly defined.

However, to specify the exact required behaviour of the total system, the *behaviour representation* of the various processes has to be filled in. In other words, the exact functional behaviour of the processes has to be specified. This is obviously the most crucial and complex part of the entire specification. Therefore, it requires a solid specification methodology to consistently work towards a correct, exact and illustrative specification allowing one and only one interpretation.

Of course, it is clear that the behaviour should represent that what is known from the International Standards and the CCITT Recommendations, as summarized in the previous two Chapters. The main problem that we have to face here is the translation from these textual documents (which we have seen to be open for interpretation and non-systematic) to a formal specification.

Therefore, three ‘translation stages’ are defined to orderly develop the specification. These three stages are explained below.

1. **The three phases of communication**

First, one can distinguish between the *three phases of communication* as is also shown in Figure 2.13 for the data link service description. These three phases are:

- the *DLC establishment* phase, in which the actual data link connection is established using `CONN` type primitives;
- the *Data transfer* phase, which includes normal data transfer as well as resetting corrective actions without clearing the data link connection; and
- the *DLC release* phase, in which the data link connection is gracefully cleared up using `DISC` type primitives.

These three communication phases merely represent the communication from the DLS user point of view and in order to introduce the DLS provider characteristics, a lower level of abstraction should be entered.

2. **Specification of the statical functional behaviour**

The *creation of a state diagram* specifying the *statical functional behaviour* is the next step in the overall behaviour specification. The several communication phases can be subdivided into
distinct states within the overlaying communication process. In Figure 2.12 such a subdivision is shown in terms of DLS primitives, which can be used for the creation of the state diagram. For each communication phase one or more states are introduced to model the behaviour in that phase. This is shown in Figure 4.4.

If the DLS user issues a primitive (a DLSDU), this will normally lead to a reaction by the DLS provider issuing a DLPDU (an HDLC frame). Conversely, the receipt of a DLPDU by the DLS provider can lead to this issuing of a DLSDU by the DLS provider to the DLS user. Both occurrences may lead to a change of state. So, this mutual coupling of DLSDUs to DLPDUs provides for the events that force state transitions. The states induced by this coupling are shown in the second column of Figure 4.4 for each communication phase. It is obvious that these states are also represented in Figure 2.12.

Apart from these states, some more have to be introduced, since there also exist communication sequences residing at the DLS provider level only. For example, the sequence(s) of frame exchanges during a frame rejection exception condition require separate states FRMR Frame Received (for the primary/combined station) and Frame Rejection Reported (for the secondary/combined station) without involving an exchange of DLSDUs. Also, an extra state has to be introduced in case a primary/combined station issues a DISC frame. The station should, as one might think, not return to the initial state Idle immediately, but has to wait for an acknowledgement by the remote station, and may resend the frame if such an acknowledgement is not received within a system-defined time-out period. This is effected by introducing the state Outgoing Disconnection Pending.

The states induced by pure DLS provider actions are shown in the third column of Figure 4.4. A short explanation of each state is given below.

- The state Idle is the starting state for each station type, and each communication session ends at this state. Possible transitions are to states Outgoing Connection Pending, for the primary/combined station, and Incoming Connection Pending for the secondary/combined station.

- The state Outgoing Connection Pending will be turned to if a primary/combined station has issued an SXXM type mode-setting command. The station is waiting for an
acknowledgement for the DLC connection attempt from the remote station.

- The state **Incoming Connection Pending** will be assumed if a secondary/combined station has received an SXXM type mode-setting command from the remote station. The secondary/combined station has to send an acknowledgement to the DLC establishment or should reject the DLC establishment attempt if it cannot accept this DLC establishment attempt.

- The state **Data Transfer Ready** is the state in which normal data transfer can take place (exchange of information and supervisory frames). This can continue until an exception condition occurs, forcing the station into another state.

- The state **DLS User Initiated Reset Pending** indicates that the DLS user of a primary/combined station has issued a RSETreq primitive and the primary/combined station has issued an SXXM type mode-setting command to reset the DLC. The primary/combined station is waiting for an acknowledgement by the secondary/other combined station.

- In the state **DLS Provider Initiated Reset Pending** a secondary/combined station has previously received an SXXM type mode-setting command from the primary/other combined station and should acknowledge the resetting attempt (with a UA response frame) or reject it (using a DM response frame of a DISC command frame, respectively).

- The state **FRMR Frame Received** is used only by primary and combined stations when they have previously received a FRMR response frame from the secondary/other combined station. As a reaction, an SXXM type mode-setting command shall be issued and an acknowledgement from the remote station shall be waited for.

- A similar state is **Frame Rejection Reported**. This state can be assumed by both the secondary and the combined station if a frame rejection condition has occurred (indicated by one or more of the parameters W, X, Y, Z) and a FRMR frame has been sent to report. If the latter sends an SXXM type mode-setting command to clear this exception condition, the state can be left on the issuance of a UA response frame.

- The last state, **Outgoing Disconnection Pending**, is assumed by primary/combined stations only. Here, the data station is trying to gracefully disconnect the DLC by sending DISC commands. It has to wait for an acknowledgement from the secondary/other combined station (a UA or DM response). If such a response is not received within a system-defined time-out period, the retransmission of a DISC command shall be effected.

The nine states that are defined thus far can only represent the functional behaviour of a data station if they are interrelated by the events that cause the transition from one state to another. In other words, the primitives, frames, or exception conditions that cause these state transitions have to be added to complete the functional behaviour representation of a data station. This functional behaviour representation in terms of states and events that cause state transitions can be seen as a *statical* specification. I.e., it can only specify which events in a certain state can cause a transition to another state, but it does not specify when such an event may occur nor which event has the higher priority in case of simultaneous occurrence of two or more possible events. Figure 4.5 should be interpreted, having these restrictions in mind.
Formal specification of the HDLC controller

Figure 4.5: State transition diagram for the statical functional behaviour of the HDLC byte controller.

Notes on the use of this diagram:

SXXM means either the SNRM, the SAR, or the SAR frame.

- [Frame] / [Primitive] means the frame is received and the primitive is issued to the DLS user;
- [Primitive] / [Frame] means that the DLS user has issued the primitive and the frame is sent as a reaction;
- [Frame] / [Primitive] / [Frame] means that the first frame is received and as a reaction the primitive is issued to the DLS user and the second frame is sent;
- [Frame] / [Frame] means that the first frame is received and the second frame is sent as a reaction;
- W, X, Y, Z / [Frame] means a frame rejection condition has occurred and a frame is sent as a reaction.
4.2 Specification of the HDLC byte controller in SDL

Example

As an example, consider a combined station in state Idle, i.e., at system start-up. From Figure 4.5 one can see that, if a SABM command is received, the station will issue a CONNind to the DLS user and move to the state Incoming Connection Pending. In that state, the station can receive a CONNresp from the DLS user, meaning the DLS user accepts the DLC establishment attempt. In that case, the station will send a UA response to the other combined station to confirm this acceptance and turn to the Data Transfer Ready state. On the other hand, the station can, for example, also receive a DISC frame from the other combined station in case of early DLC release. In that case, the combined station shall issue a DISCind primitive to the DLS user, to inform it about the disconnection, and will send a DM frame to the other combined station to acknowledge its disconnection initiative.

In the same way, the entire state diagram can be 'walked through' and it is clear that it can serve as a functional overlay for any communication session between any legal pair of data stations. Still, this does not complete our specification, since the time-dependent behaviour of a station is not yet taken into account. This requires a third step in the specification process.

3. Specification of the dynamical behaviour

This third step comprises the specification of the dynamical behaviour of the functional processes, i.e., a specification, for each state, of the order and priority in which frames and DLS primitives may be issued or received, taking into account the effect of time in the overall procedures. This order and priority of frames and DLS primitives can only be properly embodied in the specification by working very consistently. More precisely, this requires order and method (with reference to the citation at the beginning of this Chapter). A good treatment of this specification is, hence, found in the making of checklists of occurrences for each state; this means asking the same questions at each state to be specified:

- which DLS primitives can be issued at which stage of communication? It is important to recognize that some DLS primitives are destructive to others as pointed out in Chapter 2. So, a DISC type primitive always has the highest priority on other DLS primitives, followed by a RSET type primitive, while all other DLS primitives have equal priority and are not destructive to others.

- The same question can be asked regarding HDLC frames: which frames can be received at which stage of communication? Normally, the frames are dealt with in the order in which they are received, but the consequences of receiving a certain frame are filled in by the question

- what should be the reaction of the issuing of a certain DLS primitive or frame in this data station as well as in the remote data station? One should try to imagine what course of actions occurs in both stations; this is even hampered by the fact that frames can get lost, so, also, the use of timers to detect no-reply or lost-reply occurrences has to be filled in here (this is explained in Section 3.2.7).

- At last, some administrative actions have to be taken in order to keep parameters up to date or to set/reset them. For example, the number of outstanding I frames has to be accounted for, or the number of times a certain frame is being (re)sent. Most of these administrative actions take place in the process SR.
Let's take a look at a part of the specification to see how these questions can help filling in the dynamical behaviour of a process.

**Example**

Consider a combined station's processes TX and RX at system start-up. The station is not yet in any phase of communication, but one can assume it has just left the DLC release phase, i.e., there is no DLC yet. The station will first assume the state Idle. Our first question is: which DLS primitives can be issued at this stage of communication? Obviously, since there is no actual communication yet, only a CONNreq primitive can be issued by the DLS user\(^10\). As a reaction to the issuance of a CONNreq primitive, a SABM frame with the P bit set to 1 shall be issued by the combined station.

The second question is: which frames can be received at this stage of communication? We have seen by the previous question that the combined station may send a SABM command frame on the issuance of a CONNreq primitive: so may the other combined station (see the third question). So this data station may receive a SABM command from the remote combined station. If such a frame is received correctly\(^11\), the station will issue a CONNind primitive to the DLS user and changes to the state Incoming Connection Pending. Furthermore, if one imagines what happens to the other combined station that has previously issued a SABM frame (i.e., that station is in state Outgoing Connection Pending) one can see that this station can also send a DISC frame as a reaction to an issued DISCreq primitive. So, our station of interest can also receive a DISC command frame, which has to be responded to with a DM response frame, since the station has not yet established a DLC with the other station. At last, as [IS4335] tells us, any other command received with the P bit set to 1 shall be responded to with a DM frame with the F bit set to 1.

To complete the specification of this Idle state, the processes' administration has to be updated. For example, if a frame is sent, the values of the P and F bits sent have to be updated via the signals P.Out and F.Out, respectively. If a frame is sent, that might require a retransmission (as is the case here with the SABM frame), retransmission counters have to be updated (\texttt{Set.Ns(0)} and \texttt{Inc.Ns}), and a retransmission timer should be set (\texttt{SetTimer.Ts}). All these administrative actions are reported to the process SR via signal communication and are effected in this process, so both the TX and RX process can share the values of the variables accounted for.

In the same way, the other processes can be specified. However, the third question will show to be the hardest to answer, since it requires the capability of keeping track of what happens in the local station's processes TX and RX and what happens in the remote station's processes TX and RX; there are several simultaneous causal relationships to be explored! Furthermore, what is shown above for the combined station is even simpler than the specification of a primary-to-secondary relationship, since the combined stations are completely symmetrical.

---

\(^10\)One could think that a DISCreq might also be issued here, but that is not true; first, a CONNreq has to be issued by the DLS user (see Chapter 2).

\(^11\)Important note: only correctly received frames are shown in the SDL specifications, since frames that contain an FCS error may be ignored.
4.2.4 Adding quantitative information to the specification

The specification established thus far should be completed with the inclusion of quantitative information in order to define exactly the dimension of the relevant communication parameters. With this information added the specification is actually complete: not only the behaviour of the system is specified, but also its functional capabilities and operational performance.

As will be seen below, most of the quantitative information parameters are defined as synonyms for a variable which is given a value outside the system boundaries. This means that these parameters actually reside outside the environment of the system, are assigned a value at system creation time, and, hence, are known by the system on an *a priori* basis. So, not the practical value of a these parameters is of importance here\(^{12}\) in the first place, but the way in which they are defined/declared and the means by which they contribute to the system's overall scalability.

Therefore, the remainder of this Chapter will discuss the way in which these DLS and HDLC parameters are defined for each station type and what their function is in the entire specification.

Address definitions

In Chapter 2 the DLSAP *called address*, *calling address*, and *responding address* were defined. Formally, in the definition of the CODLS, the called and calling addresses are carried as parameters by CONNreq and CONNind primitives, and the responding address by CONNresp and CONNconf primitives (refer to Figure 2.13). At this stage of address definition, the parameters which take addresses as value all refer to DLSAP addresses. However, two important remarks regarding the use of the different types of addresses have to be made here. First, it is stated in [X.212] that some addresses may be implied, depending on the type of implementation of the data link service. Second, in the specifications of Appendix C, no address-conversion is effected between the DLS user and the DLS provider. So, the DLSAP addresses take the same values as the HDLC addresses which are standardized in [IS3309], [IS7776], and [X.25].

These HDLC addresses are normally one-byte identifications for the data station referred to. Though, the address semantics is quite diverse for each type of data station. For a primary station, the address represents the identity of the secondary station to which a DLC is established. So, the primary station actually has no own identity, which is fully understandable, since there can only be one primary station in a system configuration. Therefore, identification of the station itself is not necessary. Addresses of secondary stations are however important parameters for the primary station, so they are passed as parameters in the CONN type DLS primitives and are used throughout each the specifications as process instance identifications (the Adr parameter is passed by value through the created processes in the primary station's byte controller block).

However, for both the secondary and the combined station, identification is necessary and addresses are defined outside the system boundaries using synonyms. Each data station is assigned an address which is known at system creation time and is a fixed value during any communication session. A secondary needs to be able to identify itself to the primary station

\(^{12}\)This is highly system-dependent and changes with the type of implementation and configuration that is created.
with its own address\textsuperscript{13} and a combined station has to be able to recognize if a received (or sent) frame is a command frame or a response frame, i.e., it has to differ between the A and B address\textsuperscript{14} (see [X.25]).

Quality of service definition and related parameters

The term quality of service (QOS) refers to, as we have seen, certain characteristics of a DLC as observed between the connection endpoints. QOS describes aspects of a DLC that are attributable solely to the DLS provider. Therefore, the QOS parameters will be discussed from this DLS provider point of view.

In Section 2.2.5 six types of QOS parameters were defined and it was noted there that these parameters are generally determined by a priori knowledge and agreement or are implicitly conveyed by the DLS provider. It was also found that Recommendation X.212 does not mention when, if, or how a certain QOS parameter should be implemented.

Therefore, this Section will be used to explain the definition and usage of the DLS provider parameters that occur in the specifications, but it will not show how a certain parameter contributes to a certain QOS nor even to which QOS a certain parameters adds value. This is left to the reader.

The following parameters occur during the specifications of Appendix C:

- the \texttt{Mode} parameter is a relevant parameter that has a very straightforward meaning for the primary and secondary station types. In the case of a primary station, this parameter supports a DLS user's request for a certain operational mode, chosen from UNC or UAC, representing normal response mode and asynchronous response mode operation, respectively. So, the DLS provider in the primary station case, assumes any of these modes from the \texttt{Mode} parameter that is carried by the DLS user CONN type parameters.

A secondary station, however, knows its operational mode from the frames it receives. After a primary station has chosen an operational mode it will start sending mode-setting command(s), either SNRM or SARM, and the secondary station will recognize which \texttt{Mode} to assume on receipt of any of these frames.

In the case of balanced operation, there is only one operational mode possible, so there is no need to introduce a \texttt{Mode} type parameter here;

- the \texttt{MaxSt} parameter indicates the maximum number of secondary stations to which a primary station can be connected. Since two secondary station addresses are reserved (00000000 and 11111111), and since one-byte addressing is used, exactly 254 addresses can be used for secondary station identification. Consequently, \texttt{MaxSt} can at most be 254. The practical value of it depends, however, amongst others on the primary station's total frame storage capabilities;

\textsuperscript{13}Command frames (primary station's frames) always carry the address of the station to which the frame is sent, while response frame (secondary station's frames) always carry the address of the station from which the frame originates.

\textsuperscript{14}The address A stands for 11000000 and the address B stands for 10000000, where the leftmost bit is the first bit transmitted. The previous footnote regarding the use of addresses in command and response frames is also valid for combined stations.
4.2 Specification of the HDLC byte controller in SDL

- **WinSize** defines for any station type separately the size of the assumed sliding window as it is used for I frame transmissions. This parameter relates directly to the stations capability of storing I frames for retransmission. **WinSize** can therefore be defined as the maximum number of I frames that can be stored for retransmission;

- **Mod** is also defined for all station types and usually – certainly in this case – takes the value 8. It is the modulus of the I frame sequence numbers, where they cycle through the entire range if 0 to **Mod** - 1;

- **NsMax**, **NdMax**, and **NrMax** are the retransmission counters for mode-setting command frames (SNRM, SARM, SABM, and DISC) used in DLC establishment, DLC release, and resetting attempts, respectively. These parameters are defined for the primary and the combined station only;

- **TS**, **TD**, **TR**, and **TR** represent the duration of the timers and are, hence, highly system-dependent. These timer durations are defined for the primary and the combined station only. It should be noted that for both combined stations in a configuration the durations of the same time-out functions should have different values in order to resolve possible contention situations (see [IS3309] and [IS4335]).

The relevant criteria for the use of timers are discussed in the next Section.

4.2.5 The use of timers in the specifications

In Section 3.2.7 some time-out considerations are given for each mode of operation. Since these considerations do not form part of what is standardized in the International Standards, one is free to decide on how to use the several time-out functions.

Therefore, the simplifications made in this use of timers throughout the specifications are fully allowed:

- the primary/combined station command reply time-out functions are left unchanged, since they are most crucial in the detection of no-reply or lost-reply events with checkpointing as the relevant retransmission criterium. Therefore,

- the primary/secondary/combined station I frame reply time-out function has become redundant. This can be explained as follows: checkpointing is the retransmission criterium that is based on the exchange of P and F bits. Once an I frame with the P bit set to one is sent, a timer is set in the sending station. Then, basically two exceptional situations can arise. First, the I frame with the P bit set to one can get lost. This means that the remote station does not increment its receive counter (all I frames that follow are not acknowledged) and a timer will run out in the local station forcing it to retransmit all outstanding I frame starting from the I frame that got lost. Second, the response frame with the F bit set to one can get lost causing again the expiry of the time-out function. The retransmission of a frame with a P bit set to one at the local station then initiates the retransmission of the frame with the F bit set to one at the remote station. Hence, the time-out function at the local station together with the checkpointing criterium suffices in accurately regulating the bidirectional flow of I frame with possible errors in both commands and response. The only possible drawback in the usage of this time-out gauge is that it can cause some throughput performance degradation, but this effect can be minimized by delicate timer duration choice;
At last, the use of secondary/combined station command request time-out functions in asynchronous operational modes is surplus. This can be made justifiable using the same arguments are given above and is left to the reader.
Chapter 5

Implementation suggestions and conclusions

So far, the overall HDLC controller design has been extended to the required level of system behaviour specification. Although the leading aim of this research is still an implementation in hardware, the more general specification approach followed has been preferred to a more case-sensitive specification towards hardware for reasons mentioned in Section 4.1.1. Of course, this strategic consideration entails some consequences for further design steps. For example,

- the specification created thus far was never intended to be executable by a machine in a time- and memory-efficient way in the sense a programming language is. This is an essential characteristic of SDL (see Appendix B) and it implies that

- the specification can be put into different implementations, depending on the interpretation of it towards further design steps. Still, the resulting system behaviour should self-evidently be the same for any kind of implementation.

Moreover, since the specification learns one what should be implemented rather than how this should be done, it naturally introduces a next design step that necessarily copes with interpreting the specification in the context of the employed implementation language (or whatever).

In the next Section, some typical implementation issues that will occur in this design step will be pointed out shortly.

5.1 Towards an implementation in IDaSS

5.1.1 The SDLC/HDLC controller in IDaSS

As mentioned in Chapter 1, the case treated here is slightly different, caused by the fact that a complete hardware design in IDaSS is already available. This piece of hardware architecture was initially based on INTEL's 8044 microcontroller and the latest version¹ looks very much like a stripped version of INTEL's rather similar 8051 microcontroller. The instruction set

¹For the latest versions of the SDLC/HDLC controller designed in IDaSS the reader is advised to contact: dr.ir. A.C. Verschueren, Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, the Netherlands (Email: verschueren@eb.ele.tue.nl).
and register definitions are, of course, focussed on operations and routines that are likely to be found in HDLC and HDLC-like protocol handling. The actual protocol routines, constituting the desired behaviour, can be microprogrammed in a 1024 bytes 'micro-ROM'. (For more details regarding the programming capabilities and testing of the controller, see [Hemm 94].)

Of course, the system specifications have been targetted – more or less deliberately – at the underlying knowledge of the existence of this 'ready-to-be-programmed' piece of hardware. Still, they can be said to keep their general validity, and the sometimes striking resemblances between specification and implementation merely result from common sense during the design process rather than an attempt to simultaneously tune in to or join both design concepts.

5.1.2 Possible implementation problems

The similarities between the specification and the implementation reside at several logical levels. For example, the structural decomposition of the system into a bit processor and a byte processor part is obvious. Also, the hardware design is split into a (frame) receiving and a (frame) transmitting process that can work independently and asynchronously, but are able to share values of some important variables (see below). On the functional behaviour level (as far as this can be recognized in a piece of hardware description), the already mentioned instruction set and register definitions provide for virtually any necessary routine that might be needed in microprogramming HDLC controller behaviour, ranging from memory control and timer actions down to simple parameter administration. Without digging into too much detail about the exact system capabilities or any possible programming details, the IDaSS design is very likely to be ready to efficiently implement the complete HDLC controller.

Still, there are some important implementation issues left for further study or explanation. These partly depend on the specification interpretation context and partly on restrictions given in by the developed hardware design. For completeness they are summarized below.

- The use of the processes RX and TX is not as obvious as one might think at first glance. In the specifications, these processes can work totally independent from each other and they can exchange information of various kinds using another process (the process SR) that merely takes care of some administrative functions. In the IDaSS implementation, however, these processes are to be interpreted slightly differently. Complete independency of these processes should, for example, require duplicate ALUs and other logic. This is not at all desirable, and, hence, the receive process is chosen to be event driven. It can perform actions until it must wait for a certain (to be defined) event to occur, at which time the transmit process continues. The receive process continues automatically when the event occurs and switching between the processes takes no time at all. The programmer should take notice of this, certainly in the occasions where the receive and transmit process closely interwork.

- The use of priorities in the specifications cannot be simply programmed in the IDaSS design and some means have to be created to keep track of the occurance of events and to discriminate between higher and lower priority events.

- The implementation can perform subroutines that dig into the program for only one level. This seems to suffice for programming the specified behaviour, notably the macro definitions used. Fortunately, this subroutine handling is also designed to allow for return-to-original-state constructs.
5.1 Towards an implementation in IDaSS

- The specification processes RX and TX interwork using the *shared resources process* SR. In IDaSS, this interworking is generally effected by reading and writing registers that account for a certain parameter’s value. Several of these registers are defined separate for each process and others are assigned to one process only. This should be kept in mind while programming. Furthermore,

- the events that occur from the DLS user (like DISCreq, and CONNind) have to be implemented by some means. Probably, some of these events have to be defined as program events (separately or ‘simultaneously’ for the processes RX and TX) and others can perhaps be seen as simple bit-setting operations (for example, DATAreq or DATAind).

- Perhaps the most important issue that is left open in the design is the *memory management function* needed to efficiently store user data blocks that are received or that are to be (re)transmitted. In the specification, this part of the system is kept informal, but towards implementation some ideas have to be generated to account for this very crucial part of the system. Besides that this part of the implementation calls for a memory efficient solution, also a probably very time critical design might be needed. A logical choice would be to use a *page-wise structure of the memory*, directed by address fields to locate the wanted piece of information. Also, the receipt of other frames (i.e., non-I frames) has to be kept working properly in such an environment.

- Although only byte processor functions are specified, some typical *bit processor functions* have to be accounted for also. For example, some information to recognize frame rejection conditions naturally comes from the bit processor, and this information has to be passed to the byte processor either event-wise or via register settings.

- The specification is given only for the HDLC basic repertoire. It is obvious that some optional functions can easily be added, both in the specification and in the implementation. Still, if such options are considered, the implementor is advised (again) to include these into the specification first (unless it affects some hardware logic only, as is the case in using a 32-bit CRC, for example).

- At last, *testing* has to be performed. Once the system is implemented it can be tested for the correct behaviour\(^2\) and for performance. How such tests can be done or generated is left for further study.

\(^2\)Conformance tests of these kind should actually be performed on the specification. This has not been possible, however, since there was no software available for these purposes during the research. Apart from that, testing would have been hard: the specification is not fully formal and it would virtually impossible to correctly simulate both the physical layer and network layer behaviour that are insuperable in such tests.
5.2 General conclusions

The results of this research towards an HDLC byte processor design in IDaSS have evolved step by step, starting by thoroughly taking knowledge of the available standards relevant to this topic, to a structured design approach leading to the SDL specifications. It is only by this attentive interpretation study and the presented top-down design methodology that these specifications can be said to conform – to a probably very high extent – to the ISO International Standards and CCITT Recommendations and to be relatively easily to implement in hardware. Only after testing the entire design, it is possible to estimate the value of this contribution to the remainder of this much wider research project.

The conclusions to be drawn from this are, hence, quite paradoxical. Hopefully, it has been made acceptable that to claim the correctness of what has resulted from this research is not merely a matter of wishful thinking.
Bibliography

[Asch 86] Aschenbrenner, J.R.
*Open Systems Interconnection*

[Auli 89] Aulia, B.A.N. and P.H.A. Venemans
*A Skeleton for the Specification of OSI Protocols in SDL*

*Modelling OSI in SDL*

[Beli 91] Belina, F., D. Hogrefe, and A. Sarma
*SDL with Applications from Protocol Specification*
BCS Practitioner Series

[Blac 91] Black, U.
*X.25 and Related Protocols*

[Carl 80] Carlson, D.E.
*Bit-Oriented Data Link Control Procedures*

[Cona 83] Conard, J.W.
*Services and Protocols of the Data Link Layer*

[Day 83] Day, J.D. and H. Zimmermann
*The OSI Reference Model*
[Dick 83] Dickson, G.J. and P.E. de Chazal
Status of CCITT Description Techniques and Application to Protocol Specification

[ECMA 81] HDLC Selected Procedures
Standard ECMA-71

[Hemm 94] Hemmen, E.P.J.
Testprogramma voor de 80U SDLC/HDLC-controller
Publ. of the EUT Group of Digital Systems (Trainee report; no. EB 498).

[Hogr 88] Hogrefe, D.
Protocol and Service Specification with SDL: the X.25 case study
University of Hamburg report, no. FBI-HH-134/88.

[Hulz 88] Hulzebos, R.M.
Description of parallel processes in hardware using SDL (The CCITT Specification and Description Language)
Publ. of the EUT Group of Digital Systems (Master's thesis; no. 5352).

[Lini 83] Linington, P.F.
Fundamentals of the Layer Service Definitions and Protocol Specifications

[IS3309] Information processing systems - Data communication - High-level data link control procedures - Frame structure
International Standard 3309.

[IS4335] Information processing systems - Data communication - High-level data link control elements of procedures
International Standard 4335.

[IS7478] Information processing systems - Data communication - Multilink procedures
International Standard 7478.

[IS7498] Information processing systems - Open systems interconnection - Basic reference model
[IS7776] *Information processing systems - Data communication - High-level data link control procedures - Description of the X.25 LAPB-compatible DTE data link procedures*
   International Standard 7776.

[IS7809] *Information processing systems - Data communication - High-level data link control procedures - Consolidation of classes of procedures*
   International Standard 7809.

[IS8471] *Information processing systems - Data communication - High-level data link control balanced classes of procedures - Data-link layer address resolution/negotiation in switched environments*
   International Standard 8471.

[IS8885] *Information processing systems - Data communication - High-level data link control procedures - General purpose XID frame information field content and format*
   International Standard 8885.

[IS8886] *Information processing systems - Data communication - Data link service definition*
   International Standard 8886.

[Lavr 93] Lavrijssen, P.E.O.
   8044 SDLC/HDLC controller
   Publ. of the EUT Group of Digital Systems (Master's thesis; no. 6086).

[Sara 89] Saracco, R. et al.
   *Telecommunications Systems Engineering using SDL*

[Tane 88] Tanenbaum, A.S.
   *Computer Networks*

[Verh 90] Verhoosel, J.
   *A summary of the Specification and Description Language*
   Publ. of the EUT Department of Mathematics and Computing Science (Master's thesis; no. 238).

[Vers 90] Verschueren, A.C.
   *IaASS for ULSI*
   Publ. of the EUT Group of Digital Systems.
[X.25] *Interface Between Data Terminal Equipment (DTE) and Data Circuit-termination Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuits*
Recommendation X.25, Blue Book, vol. VIII, fasc. VIII.2
(Recommendations X.1-X.32).

[X.212] *Data Link Service Definition for Open Systems Interconnection for CCITT Applications*
Recommendation X.212, Blue Book, vol. VIII, fasc. VIII.4
(Recommendations X.200-X.219).

[Z.100] *Functional Specification and Description Language (SDL) - Criteria for Using Formal Description Techniques (FDTs)*
Recommendation Z.100 and Annexes A, B, C and E - Recommendation Z.110,

[Z.100/D] *SDL User Guidelines*
Appendix A

The OSI reference model

Open Systems Interconnection (OSI) is an international standards activity that primarily defines formats and protocols to interconnect systems that have different architectures provided by different suppliers. The basis of OSI standardization is a reference model for the coordination of standards development, called the OSI reference model.

The OSI reference model is briefly discussed in this Appendix. Those familiar with the OSI concepts are offered the choice to skip this Appendix or use it for lookup purposes only. Those not familiar with the OSI concepts or OSI terminology are advised to study this Appendix, or, maybe preferable, read some more advanced literature on this specific topic. A rather advanced study of the OSI reference model is given in numerous books and articles on computer and data communication science, and, of course, the international standards. Aschenbrenner [Asch 86], Black [Blac 91], and Tanenbaum [Tane 88] give a concise description of most of the OSI concepts. For a more extensive and detailed study one is referred to the ISO International Standard 7498 [IS7498].

This Appendix mainly follows the line of the article by Day and Zimmermann [Day 83].

A.1 The beginning of OSI

In the mid-1970's early successes of computer networks evolved and began to achieve considerable attention. The rapidly emerging information technology industry of the 1980's, that followed, made it apparent that to utilize the full potential of computer networks, international standards would be required. In March of 1977, the International Organization for Standardization (ISO) urgently initiated a new subcommittee (SC16) for Open Systems Interconnection (OSI) to address these requirements.

In short, the mission of SC16 was to develop an architecture that would form the basis for the further development of a set of intersystem standards. Still, the task of this standards committee was somewhat different than in most other standardization cases. If there was to be a consistent set of international standards, OSI would have to lead rather than follow the proclaimed commercial development. A set of standards should be developed which emerging products could converge to before the commercial practices were in place and while many of the more fundamental research problems remained unsolved. More formally, see [Asch 86], one could state that the goals of OSI were (and still are)

---

1The term 'open' was chosen to emphasize that by conforming to OSI standards, a system would be open to communication with any other system, obeying the same standards, anywhere in the world.
A.2 The OSI reference model, services, and protocols

SC16 used a layered-architecture approach to break up the problem into (intellectually) manageable pieces. This architecture, referred to as the OSI reference model, is a framework for coordinating the development and/or improvement of OSI standards, and for providing a common reference for maintaining the consistency of all related standards. The OSI reference model is in fact an abstract (or, as one could say, highly conceptual) description of interprocesses communication. This communication takes place between application processes running in distinct systems. It is stressed here, that it is not the intent of the OSI reference model to serve as an implementation specification nor as a basis for appraising the conformance of actual implementations, nor to provide a sufficient level of detail to define precisely the services and protocols of the interconnection architecture.

In OSI, the problem is approached in a top-down fashion, starting with a description at a high level of abstraction which imposes few constraints, and proceeding to more and more refined descriptions with tighter and tighter constraints. It is therefore, that three levels of abstraction can be explicitly recognized: the architecture (the actual OSI reference model), the service specifications, and the protocol specifications. The relationship between those three levels of abstraction is clarified is Figure A.1.

A.2.1 The OSI architecture

The OSI architecture can be said to be the highest level of abstraction in the OSI scheme. A good way to think about this term is to consider the difference between an architecture and a building built to that architecture. Day and Zimmermann [Day 83] use, as a parallel, the example of Victorian architecture, which is a set of rules and stylistic conventions that characterize a particular form. A Victorian building is a building built to those rules and conventions. One cannot walk into a Victorian architecture; one can, on the other hand, walk into a Victorian building. More formally this can also be considered as the distinction
Figure A.1: The OSI reference model, services, and protocols: successively more detailed and more constraining specifications.

between the type of an object (i.e., the architecture) and an instance of that object (i.e., the building). The OSI reference model defines types of objects that are used to describe an open system, the general relations among these types of objects, and the general constraints on these types of objects and relations. Specifications for the lower level of abstraction may define other relations and tighter constraints for their purposes, but these must be consistent with those defined in the OSI reference model.

The OSI architecture also defines, constructed from these objects, relations, and constraints, a seven-layer model for interprocess communication. This part of the architecture will be discussed in Section A.3.1.

A.2.2 The OSI service specifications

The OSI service specifications represent a lower level of abstraction that defines the service provided by each layer in greater detail. The OSI service specification will define tighter constraints than the OSI reference model on the protocols and implementations that will satisfy the requirements of the layer. An OSI service specification defines the facilities provided to the user of the service, independent of the mechanisms used to accomplish this service. It also defines an abstract interface for the layer, in the sense that it defines the primitives that a user of the layer may request with no implication of how or if that particular interface is implemented.

A.2.3 The OSI protocol specifications

The OSI protocol specifications represent the lowest level of abstraction in the OSI standards scheme. Each protocol specification defines precisely what control information is to be sent

---

5 This distinction will also show to be vital in SDL specifications, where, for example, a process graph defines the object type while several instances of that process can exist, behaving according to the process specification.
and what procedures are to be used to interpret this control information. The OSI protocol specifications represent the tightest constraints placed on implementations built to conform to OSI standards, i.e., allowing open systems to communicate while still allowing differences in implementations.

A.3 Elements of the OSI reference model

A.3.1 The seven-layer model

As we have already seen, the OSI reference model defines a layered architecture. This layering is used as a structuring technique to allow the network of open systems to be logically decomposed in independent, smaller subsystems. Each individual system itself should be viewed as being logically composed of a succession of subsystems, each subsystem corresponding to the system with a layer. In other words, a layer is being viewed as being locally composed of subsystems of the same rank in all interconnected systems. Each subsystem, in turn, is viewed as being made of one or several entities. A layer, therefore, comprises many entities distributed among interconnected open systems. Entities in the same layer are termed peer entities. The seven layers of the OSI reference model are shown in Figure A.2.

![The seven-layer OSI architecture](image)

Figure A.2: The seven-layer OSI architecture.

The basic idea of layering is that each layer adds value to services provided by the set of lower layers in such a way that the highest layer is offered the full set of services needed to run distributed applications.

Another basic principle of layering is to ensure independence of each layer by defining services provided by a layer to the next higher layer, independent of how these services are performed. This permits changes to be made in the way a layer or a set of layers operate,
provided they still offer the same service to the next higher layer. The seven layers, as defined by the OSI reference model, are briefly described below.

- The **application layer (AL)** provides identification of intended communications partners and identification of the subjects to be communicated. So, the primary concern of the application layer is with the *semantics* of the application running. All applications reside in the application layer, however, it should be noted that only part of the application layer is in the real OSI system.

- The **presentation layer (PL)** provides data transfer and selection of the *user data syntax*. More specifically, the primary purpose of the presentation layer is to provide independence to application processes from differences in data representation, i.e., syntax.

- The **session layer (SL)** provides session connection establishment and release, turn management, session synchronization, and exception reporting, i.e., the mechanisms for organizing and structuring the interactions between application processes.

- The **transport layer (TL)** provides end-to-end sequence control, flow control, error recovery, multiplexing, and blocking, relieving the upper layers from any concern with providing reliable and cost-effective data transfer.

- The **network layer (NL)** provides independence from the data transfer technology and independence from relaying and routing considerations. The network layer also handles relaying and routing data through as many concatenated networks as necessary while maintaining the quality of service parameters requested by the transport layer.

- The **data link layer (DLL)** provides error detection and, if possible, error correction, establishment and release of data link connections, link flow control, identification, and parameter exchange.

- The **physical layer (PhL)** provides the mechanical, electrical, functional, and procedural means to activate, maintain, and deactivate physical connections, and transparent transmission of bit streams. (The lower three layers are sometimes referred to as the *network layers*).

### A.3.2 Services and service access points

As already mentioned above, each layer provides *services* to the layer above. Of course, an exception has to be made here for the highest layer. A service can be defined as a capability of the \((N)\)-layer which is provided to the \((N+1)\)-entities. It is, however, important to note that not all functions performed within the \((N)\)-layer are services. Only those capabilities that can be seen from the layer above are services.

\((N)\)-entities distributed among the interconnected open systems work collectively to provide the \((N)\)-service to \((N+1)\)-entities, as illustrated in Figure A.3. In other words, the \((N)\)-entities add value to the \((N-1)\)-service they get from the \((N-1)\)-layer and offer this *value-added* service, i.e., the \((N)\)-service to the \((N+1)\)-entities.

The \((N)\)-services are offered to the \((N+1)\)-entities at the \((N)\)-service access points, or \((N)\)-SAPs for short, which represent the logical interfaces between the \((N)\)-entities and the \((N+1)\)-entities. An \((N+1)\)-entity communicates with an \((N)\)-entity in the same system through an
A.3.3 Functions and protocols

An (N)-function is part of the activity of an (N)-entity. Flow control, sequencing, data transformation are all examples of (N)-functions. Cooperation among (N)-entities is governed by one or more (N)-protocols. An (N)-protocol is a set of rules and formats which govern the communication between (N)-entities performing the (N)-functions in different open systems. In particular, direct communication between the (N)-entities in the same system, e.g., for sharing resources, is not visible from outside the system and, thus, is not covered by the OSI architecture.

A.3.4 Naming and addressing

Objects within a layer or at the boundary between adjacent layers need to be uniquely identifiable, e.g., in order to establish a connection between two SAPs, one must be able to identify them uniquely. The OSI architecture defines identifiers for entities, SAPs, and connections as well as relations between these identifiers, as briefly outlined below.

Each (N)-entity is identified with a global title which is unique and identifies the same (N)-entity anywhere in the network of open systems. Within more limited domains, an (N)-entity can be identified with a local title which uniquely identifies the (N)-entity only in that domain. For instance, within the domain corresponding to the (N)-layer, (N)-entities are identified with (N)-global titles which are unique within the (N)-layer.
A.3 Elements of the OSI reference model

Each (N)-SAP is identified by an (N)-address which uniquely locates the (N)-SAP at the boundary between the (N)-layer and the (N+1)-layer.

Correspondence between (N)-addresses served by an (N)-entity and the (N-1)-address used for this purpose is performed by an (N)-mapping function. In addition to the simplest case of one-to-one mapping, mapping may, in particular, be hierarchical with the (N)-addresses being made of an (N-1)-address and an (N)-suffix. Mapping may also be performed by table lookup.

A.3.5 Connections

A common service offered by all layers consists of providing associations between peer SAPs which can be used in particular for data transfer. More precisely, the (N)-layer offers (N)-connections between (N)-SAPs as part of the (N)-services. The most usual type of connection is the point-to-point connection, but there are also multipoint connections which correspond to multiple associations between entities, e.g., broadcast or multidrop communications. The end of an (N)-connection at an (N)-SAP is called an (N)-connection endpoint or (N)-CEP for short. Several connections may coexist between the same pair (or n-tuple) of SAPs.

Each (N)-CEP is uniquely identified within its (N)-SAP by an (N)-connection endpoint identifier or (N)-CEI for short, which is used by the (N)-entity and the (N+1)-entity on both sides of the (N)-SAP to identify the (N)-connection. This is necessary since several (N)-connections may end at the same (N)-SAP. This is also illustrated in Figure A.3.

A.3.6 Data transfer on a connection

Information is transferred in various types of data units between peer entities and between entities attached to a specific service access point.

The data units are defined below and the interrelationship among several of them is illustrated in Figure A.4.

(N)-protocol control information, or (N)-PCI for short, is information exchanged between two (N)-entities, using an (N-1)-connection, to coordinate their joint operation.

(N)-user data are the data transferred between two (N)-entities on behalf of the (N+1)-entities for whom the (N)-entities are providing services.

An (N)-protocol data unit, or (N)-PDU for short, is a unit of data which contains (N)-PCI and possibly (N)-user data. As can be seen from Figure A.4, an (N)-PDU may be mapped one-to-one onto an (N-1)-service data unit, or (N-1)-SDU.

It can be seen that, at each stage of a communication process, a layer may segment a data unit it receives from the next higher layer into several parts, to accommodate its own requirements. These data units must then be reassembled by the corresponding peer layer before being passed up.

It should be noted here that this Appendix only discusses communication between (N)-entities in the so called 'connection-mode'. In this mode, the (N-1)-service requires that an (N-1)-connection be established between (N-1)-SAPs before any communication between (N)-entities can take place. Conversely, when the (N)-entities need no longer communicate, the (N-1)-connection can be released. This connection-mode covers the more traditional teleprocessing. On the other hand, a 'connectionless-mode' is also defined within the OSI framework, in which resources in the (N-1)-layer need not be reserved in advance, but can be allocated dynamically for the transmission of a single (N-1)-SDU.
A.3.7 The relation of OSI to other models

At last, it is addressed here that OSI is not and has not been only the work of the ISO. For example, the OSI model is defined and standardized in close collaboration with CCITT and several other important telecommunications bodies.

OSI provides a set of standards for interprocess communication. The relation between OSI and other standards activities in databases, programming languages, office systems, and operating systems must be clearly established to enable those activities to fully utilize the OSI standards and the results of OSI work. These other activities have developed or will develop their own reference models which must be related to the OSI reference model. It is seen that collaboration between standardization bodies is insuperable.
Appendix B

A summary of the CCITT Specification and Description Language (SDL)

This appendix summarizes the concepts of the CCITT Specification and Description Language (SDL) and is taken partly from [Verh 90]. An even more concise summary of SDL can be found in [Dick 83].

It is stressed here, that this summary is not complete, but has to be seen as an introduction to SDL. For a more detailed description of the language and its methodologies the reader is referred to, e.g., the CCITT standard works [Z.100], Saracco's very complete and well-illustrated book [Sara 89], or, especially for protocol specification purposes [Beli 91]. Annex D to CCITT Recommendation Z.100 [Z.100/D] is well suited for usage as a quick reference.

B.1 Introduction

SDL is designed by the CCITT as a means of support to the specification and description process of telecommunication systems.

SDL can be used to specify the required behaviour and to describe the actual behaviour of a system. It can also be a base for a methodology of documentation to completely represent a system specification. Furthermore, it may be used to represent, at various levels of detail, the functional properties of a system.

A description in SDL describes in natural language the meaning of the corresponding specification, so a wider range of people can understand it. A specification in SDL can be written in a partially formal or in a totally formal way. Partially formal specifications contain statements in natural language, while totally formal specifications, on the other hand, contain only formal (SDL) statements. Of course, only the latter sort of specifications can be checked totally automatically.

SDL is based on an Extended Finite-State Machine (EFSM) model provided with the following properties:

- every combination of input and state yields an output and a next state (which is an ordinary property of an FSM);
- the FSM is extended with auxiliary storage operated on by auxiliary operations;
auxiliary operations are: decision, which makes a choice between several alternatives depending on parameters associated with inputs and on information in auxiliary storage; task, which performs actions on auxiliary storage and manipulates input and output parameters;

interactions between ESFMs are represented by signals which are optionally parameterized;

communication between ESFMs is asynchronous and implemented by FIFO queues.

SDL can be represented in two different forms depending on the same semantic model:

- a Graphical Representation (SDL/GR) using graphical symbols;
- a Phrase Representation (SDL/PR) using program-like statements.

SDL/GR is better suited for expressing ideas and allows the control flow to be easily visualized, while SDL/PR is preferable for use on a machine, because checks can be placed upon it more easily. Because of the availability of graphical terminals, it has become more convenient to use SDL/GR as input for a machine.

A number of differences between SDL and the traditional programming languages can be stated. First, SDL is better capable for conveying precise information between humans, because SDL has a higher level of abstraction. Second, SDL specifications are not intended to be executable by a machine in a time- and memory-efficient way in the sense a programming language is. Third, these specifications can be put into different implementations, depending on the kind of reader.

### B.2 Basic SDL concepts

Systems in SDL can be modelled by using building blocks for expressing the structure of a system. The behaviour of the blocks is specified by using processes, extended finite-state machines communicating through signals. For convenience, these processes can be structured with procedures, to facilitate the design and usability of a process. Processes can express behaviour by using data to keep information of the process. These four topics are discussed in the following four Sections.

#### B.2.1 SDL building blocks

A system\(^1\) in SDL is composed of a number of blocks connected by channels. Each block may contain one or more processes, which describe the behaviour of the block. The only means of communication between processes in two different blocks is by sending signals that are transported by the channels, which is illustrated in Figure B.1. Within a block, processes can communicate with one another either by signals or by shared values. The signals between processes in the same block and between a process and the environment of the block are carried by signal routes as can be seen in Figure B.2. A channel connects two different blocks in the system or a block and the environment of the system. It can be unidirectional or bidirectional. For each direction, the channel contains a list of signal identifiers that can be carried by the channel in that direction.

---

\(^1\)SDL keywords will be printed in the *helvetica* font.
Like a channel, a signal route can be unidirectional or bidirectional. When a signal is delivered to a signal route leading to a block boundary, the signal is given to the channel connected to the signal route. When a signal arrives at a block from a channel and the channel is connected to one or more signal routes, the signal is delivered to the signal route containing the identifier of the signal in its signal list. The signal list of an incoming channel is partitioned into disjunctive sets, one set per signal route connected to the channel. The signal list of an outgoing channel is partitioned into sets, not necessarily disjunctive, one per signal route connected to the channel.

A signal consists of a signal name and possibly a number of parameters carried with the signal. Signals are thus typed, whereas channels and signal routes are not. Signals are used for synchronization between processes and for carrying information from one process to another.

### B.2.2 Processes

A **process** is an extended finite-state machine, defining the behaviour of (a part of) a system. An extended finite-state machine basically consists of **states** and transitions, and uses a proprietary memory. Each process has one **input queue** in which input signals for that process are stored until the process is able to receive them. Thus, there is no separate input queue per signal route or signal type.

**States**

A **state** is a point in the process where an input is needed to proceed and perform a specific sequence of actions.

In a state, the process starts to check the input queue. If the input queue is empty, the process waits until a signal arrives. Otherwise, the first element in the queue is taken out of the queue. Two cases are possible:

- there is an explicit transition following the input of a signal. The process leaves the state and follows the transition;
• there is no explicit transition for the signal. The process implicitly carries out a null transition and enters the original state again.

In both cases the signal ceases to exist.

The save construct

In a state, signals can be saved by using the save construct. This means that a signal named in a save at a state, that is taken out of the input queue, is placed back in the queue at its original position. The process then turns to the next element in the queue.

Enabling conditions

An enabling condition can be added to an input of a signal. If the condition is true, the signal is consumed and the process follows the explicit transition following the input. If the condition is false, the signal is placed back in the queue at its original position and the next element in the queue is taken. The condition is continuously monitored while the process remains in the state and if it changes from being false to being true, as yet the signal is consumed.

Continuous signals

A continuous signal has the same basic properties as the enabling condition, except that no input is attached to it. Thus, when there is no signal in the input queue which can cause a transition, the continuous signal is checked and if it is true, the transition following it is entered. The priority of the continuous signal is always lower than that of any other signal. More than one continuous signal is allowed for each state and every continuous signal is assigned a priority. The continuous signals are then checked in decreasing order of priority.
The continuous signals are checked continuously while the process is in the state, so, if, when checking a continuous signal, a signal or another continuous signal with a higher priority becomes true, the transition following the latter is entered.

Transitions

An output is the sending of a signal from one process to another or to itself. The destination process instance must be expressed in the output.

Tasks

A task is used in a transition to represent a set of operations on variables. For example, a number of assignment statements can be contained in a task.

Decisions

A decision consists of an expression and a number of paths leaving the decision. For every value of the expression exactly one path, which can be followed, must be present. Thus, each path contains a subset of the set of values of the expression. The intersection of each pair of subsets must be empty and the union of all the subsets must be equal to the total set of values. So, the set of values is partitioned over all the paths. A decision is executed as follows: the expression is evaluated and according to the value of the expression, the corresponding path is followed.

Joins

The join concept allows control to be transferred from one point (an out-connector) to another one (an in-connector) of a process. It is not possible to transfer control to another process by means of a join. The out-connector and the in-connector of a join must both contain the same name. More than one out-connector may contain the same name, but all in-connectors must have different names. Thus, it is possible to transfer control from more than one out-connector to the same in-connector. Out-connectors may only be used within transitions. It is not possible to use in-connectors at a state, an input, or a save.

Process creation

Processes can be created in two ways. First, a process can be created at system creation time. The formal parameters of the process, if present, have, at system creation time, an undefined value. Thus, before a process tries to access its formal parameters, they have to be given values. Second, a process can be created by an explicit create request issued by another process. The creating process has to be in the same block as the created process. Actual parameters can be attached to a created process to allow transfer of information.

B.2.3 Procedures

Procedures are intended to

- permit the structuring of the process into several levels of detail;
- represent a complex assembly of items by a single item;
• allow commonly used assemblies of items to be defined and used repeatedly.

A procedure body is very similar to the process body. The only difference in executing a procedure is that upon termination a return is executed, transferring the control back to the process or procedure containing the called procedure. Procedure calls may occur wherever a task is allowed in a process or procedure.

A procedure can send and receive signals. The process which called the procedure is the originating process of a signal sent by the procedure. The procedure uses the input queue of the calling process to receive signals. All signals received are either implicitly saved or explicitly handled by the procedure.

B.2.4 Data handling

Variables can be declared in a process and are local to that process. Only the owner process can change the value of variables. Variables can be initialized directly when they are declared by inserting a value behind the declaration.

Revealed and viewed variables

Processes in the same block can share variables. The process owning the shared variable must declare it as a revealed variable. The process wanting to share the variable must declare a viewed variable with the same identifier and the same sort as the one being shared. The action in which the value of the variable is shared is called a view operation. The viewing process cannot modify the value of the variable.

Exported and imported variables

A process can export a variable to all other processes, not necessarily in the same block. The exporting process must declare the variable as exportable and the importing process must declare a variable with the same name and sort as importable. A variable is exported when the exporting process executes an export action, which causes the current value of the variable specified to be copied into an implicit variable. A variable is imported by an import action, in which the value of the implicit variable is obtained.

Timers

A timer is a special process that is able to send signals to the process which created the timer. A timer must be declared like a variable in the process owning the timer. With a set operation a timer is created and will send a timeout signal to the creating process at the time specified in the set operation. Such a timeout signal is received by the process with an ordinary input. A reset operation stops the timer. Optionally, a timer may contain a list of expressions through which values can be passed to the timer. In this way, a particular instance of a timer can be specified according to the values of the expressions.

B.3 Structuring and refining SDL systems

SDL provides two techniques that allow a top-down specification of large systems: partitioning and refinement. These two aspects will be described below.
B.3 Structuring and refining SDL systems

B.3.1 Partitioning

Partitioning is the subdivision of (a part of) a system into smaller parts where the global behaviour of the combination is equivalent to the unpartitioned part. Partitioning can be applied to blocks, channels, and processes.

A block can be partitioned into a set of (sub)blocks and (sub)channels connecting them. The subchannels are connected to the incoming and outgoing channels of the block. With respect to the signals, the subchannels and the channels of the block are related in the same way as signal routes to channels, as described in Section B.2.1. Each subblock can be partitioned itself, resulting in a hierarchical structure between blocks as is illustrated in Figure B.3. To get an exact representation of the way in which a signal is conveyed, it may, in some cases, be necessary to represent the channel behaviour. Therefore, the channel can be structured in terms of blocks, channels and processes. This structuring can be done independently of the blocks connected by the channel. With respect to the signals, the same rules hold as for the block partitioning.

Processes can be structured into services, which are described in more detail in section B.4.1.

B.3.2 Refinement

Refinement can be defined as the addition of new details to the system’s functionalities, but in practice it is only applied to the division of a signal into subsignals. In this way, lower level signals are transparent to higher levels of abstraction. The subsignals of signals can flow in both directions. The channel carrying the signal is also carrier for all the subsignals. If the channel was unidirectional, it now becomes implicitly bidirectional.

The signal refinement is tightly related to the block partitioning. Suppose, at a particular level, a signal is carried by a channel between two blocks. When the signal has to be refined,
it is natural to also partition the blocks connected by the channel, so all the subsignals are
carried by subchannels between subblocks. In this way, a complete lower level is created,
defining the behaviour of the system in more detail.

**B.4 Additional concepts**

The current version of SDL (the 1988 version) provides some additional concepts to the basic
concepts of SDL. The most important one is the service concept.

**B.4.1 Services**

The service concept is a tool for structuring a process. Services are described in independent
EFSMs, i.e., their states are orthogonal and can be better modelled independently. A process
can be partitioned into services without introducing parallelism. A service can be seen
as a part of the process representing a subbehaviour, which is a 'function' of the process.
Services of the same process share the data of this process, so they do not communicate
by ordinary signal interchange. There is, however, a second mechanism for communication
between services, by the use of so called priority signals carried by signalroutes. In order to be
able to interpret the service and the priority signals semantically, it is necessary to transform
them back to the basic concepts process and signal as described below.

**Priority signals**

Priority signals are used for communication between two different services of a process. No
signals from other processes are allowed to be consumed during the transitions caused by
the priority signals. In this way, priority signals have a higher priority than ordinary signals
and transitions can be ordered and concatenated in some way between two consumptions of
signals of other processes.

**Transformation**

The set of services comprising a process can be combined to make interpretation of the process
possible. In order to do so, two concepts have to be transformed: states and transitions.

States of services can be combined in a tuple containing one position for each service.
The number of possible tuples is then the product of the number of states of each service.
Each tuple is a state of the process to be formed. Of course, a lot of the tuples can never be
reached, because the behaviour of the process does not allow them. In this way, the number
of states used is reduced significantly. In most cases, the number of states of a process that is
partitioned into services is much less than the number of states of an unpartitioned process.

A transition between two states in a service is placed between every pair of tuples which
only differ at the position of the particular service, and of which the first one contains the
first state at the position of the service and the second one contains the second state at the
position of the service.

**B.4.2 Other additional concepts**

Some other additional concepts are macros and generic systems.
Macros

A macro can be used to structure a specification. It consists of a macro definition that can be referenced elsewhere in a system specification. The macro has no scope, so it can be referenced from every point in the system.

Generic systems

A generic system comprises different systems in one specification by the use of system parameters. These parameters have to be provided externally by the environment to obtain a specific system according to the user's need. All the parameters have to be provided before the system can be interpreted. Two additional constructs are available to provide selection of pieces of specification according to the system parameters, namely,

- the select construct, which contains a condition and one piece of specification. This piece of specification is selected statically before interpretation of the system if the value of the condition is true;

- the alternative construct, which contains a condition and at least two pieces of specification. Depending on the value of the condition, one piece of specification is selected statically before interpretation of the system. This construct can only be used in processes, procedures, and services.

B.5 A summary of SDL/GR constructs

For general reference purposes, a summary of the most often used SDL/GR constructs will be given below.
Block reference symbol

Channel symbols

Comment symbol

Connector symbol

Create line symbol

Create request symbol

Decision symbol

Enabling condition symbol

Flow line symbol

Frame symbol

Input symbol

Macro call symbol

Macro inlet symbol

Macro outlet symbol

Option symbol (example)

Output symbol

Priority input symbol

Priority output symbol

Procedure call symbol
Appendix C

SDL specifications of the HDLC byte controller
The system definition describes the entire "HDLC controller" structure and procedural functionality. It is noted here that the bit-oriented functions are referenced, and, hence, not formally specified. At this top level, a distinction is made primarily between the station type (StTy), which is an external parameter, i.e., it is specified or chosen outside the system environment and is known at system creation time. The station type can be any of the legal HDLC station types: primary station (Pri), secondary station (Sec), or combined station (Com). Once the total system is instantiated (or created) it describes the behaviour of any of these types of station. It is seen that, regardless the station type, the interface with the environment is uniform: on one side, the controller interfaces with the Network Layer via a Data Link Service Access Point (DLSAP). The 'communication' which is seen in this DLSAP is represented by the communication of signals via the DLSAP channel, comprising the exchange of Data Link Service (DLS) primitives. These are defined in the macro 'System_Signals_&_Variables' and the communication in terms of these primitives should obey what is standardized in the CCITT X.212 Recommendation. On the other side, the definition of the Physical (Ph) Layer services is not subject of study in this specification, and, hence, the exchange of Physical Layer Service primitives is not described here.

Once a station type has been selected, the specification of the exact behaviour of the HDLC controller conforming to such a station is further decomposed into a block containing a substructure for the specification of byte-oriented functions (e.g., 'Primary_Byte_Controller') and one for the specification of bit-oriented functions, which is referenced, as stated before. For every station type, the exact procedures are described in processes, which normally interwork by signals and shared variables.

These (interworking) processes have to specify exactly the HDLC procedures, as defined in the ISO International Standards, and they have to specify the functional (causal) relationship between HDLC frames and DLS primitives. See other comments for further explanations. *
MACRODEFINITION System_Signals_&_Variables

SYNTYPE Bit = Integer
OPERATORS(ALL)
CONSTANTS 0:1
ENDSYNTYPE Bit;

SYNTYPE ByteLength = Integer
OPERATORS(ALL)
CONSTANTS 0:7
ENDSYNTYPE ByteLength;

NEWTYPE Byte Array(ByteLength, Bit) ENDNEWTYPE;

NEWTYPE UserData String(Bit) ENDNEWTYPE;

NEWTYPE ModeType LITERALS UNC, UAC;
ENDNEWTYPE ModeType;

NEWTYPE StationType LITERALS Pri, Sec, Com;
ENDNEWTYPE StationType;

SIGNAL DLS primitives
CONNreq(Byte, ModeType), CONNind(Byte, ModeType), CONNresp(Byte, ModeType),
CONNconf(Byte, ModeType), DATAreq(UserData), DATAind(UserData), DISCreq, DISCind,
RSETreq, RSETind, RSETresp, RSETconf,
HDLC frames
RR(Byte, Natural, Bit), RR(Byte, Natural, Bit), RNR(Byte, Natural, Bit),
SNRM(Byte, Bit), SARM(Byte, Bit), SABM(Byte, Bit), UA(Byte, Bit), DM(Byte, Bit),
DISC(Byte, Bit), FRMR(Byte, Byte, Natural, Bit, Natural, Bit, Bit, Bit, Bit, Bit, Bit),
Other signals
Bits(Bit);

SIGNALLIST DLSAP_In = CONNreq, CONNresp, DATAreq, DISCreq, RSETreq, RSETresp;
DLSAP_Out = CONNind, CONNconf, DATAind, DISCind, RSETind, RSETconf,
Response_Rejection_Reported;

SYNONYM StTy StationType = EXTERNAL;
**BLOCK Primary_Station**

- The signallists Frames_In and Frames_Out describe the incoming and outgoing HDLC frames for the primary station, assuming the basic repertoire of commands and responses. The BLOCK Primary_Bit_Controller is referenced. Hence, it is not further described here. */

  **SIGNALLIST Frames_In**
  - I, RR, RNR, UA, DM, FRMR;

  **SIGNALLIST Frames_Out**
  - I, RR, RNR, SNRM, SARM, DISC;

**BLOCK Secondary_Station**

- The signallists Frames_In and Frames_Out describe the incoming and outgoing HDLC frames for the secondary station, assuming the basic repertoire of commands and responses. The BLOCK Secondary_Bit_Controller is referenced. Hence, it is not further described here. */

  **SIGNALLIST Frames_In**
  - I, RR, RNR, SNRM, SARM, DISC;

  **SIGNALLIST Frames_Out**
  - I, RR, RNR, UA, DM, FRMR;

**BLOCK Combined_Station**

- The signallists Frames_In and Frames_Out describe the incoming and outgoing HDLC frames for the combined station, assuming the basic repertoire of commands and responses. The BLOCK Combined_Bit_Controller is referenced. Hence, it is not further described here. */

  **SIGNALLIST Frames_In**
  - I, RR, RNR, SABM, DISC, UA, DM, FRMR;

  **SIGNALLIST Frames_Out**
  - I, RR, RNR, SABM, DISC, UA, DM, FRMR;
The processes contained in this block specify the exact HDLC procedural behaviour and the upper layer interface of a primary station. The initial process 'Setup' instantiates the processes 'TX' and 'SR'. The 'TX' process can create the 'RX' process. The data management (DataMan) processes are referenced; a textual specification is given elsewhere. */

MACRODEFINITION Primary_Signals & Variables

SIGNALLIST

P_In1 = DATAreq;
P_In2 = DISCreq, RSETreq;
P_Out = CONNconf, DISCind, DATAind, RSETconf, Response_Rejection_Reported;
F_Out1 = SNRM, SARMM, DISC, RR, RNR;
F_Out2 = I;
F_In = I, RR, RNR, UA, DM, FRMR;
RXTX = UA, DM;
SR1 = TX_Disc, Set_Ns, Set_Nd, Set_Nr, Inc_Ns, Inc_Nd, Inc_Nr, SetTimer_Ts, SetTimer_Td, SetTimer_Tr, SetTimer_Tp, Set_VS, Inc_VS, Set_VSP, P_Out, RNR_OUT;
SR2 = SetTimer_Tp, ResetTimer_Ts, ResetTimer_Tr, ResetTimer_Tp, Set_VS, Set_VR, Set_VSU, Inc_VR, F_In, Rem_Busy, Rem_Clear, FRMReq;
Man_TX = I;

SYNONYM MaxSt Natural = EXTERNAL;
/* MaxSt is the maximum number of secondary stations to which a DLC can be established. Since two addresses are reserved it can take a value from 1 to 254 inclusive. */
The substructure `DataMan_RX` contains one process named `InlMan_RX` which is created by the process `RX` with the formal parameters `Mr` and `Mode` (this is not shown). Hence, any instance of the process type `InlMan_RX` is associated with one and only one primary-to-secondary connection and its actions apply to that particular instance of data link connection.

The main task of the process is to queue the received (error-free) frames from the remote data station, in particular the (user data contained in) I frames. If the maximum number of I frames that can be stored is reached, the process will signal a `Local_Station_Busy` to the process `SR`. If this condition is no longer true, a `Local_Station_Cleared` signal will be forwarded to the process `SR`. The maximum number of I frames that can be stored is a system-dependent parameter, which is not specified here.

The means by which this process performs its task is left to the reader. While reading the rest of the specification, one can however assume a correct and sufficiently fast functioning of this process. At last, once the process `RX` stops to exist, the process `InlMan_RX` will also be stopped.

The substructure `DataMan_TX` contains one process named `InlMan_TX` which is created by the process `TX` with the formal parameters `Adr` and `Mode` (this is not shown). Hence, any instance of the process type `InlMan_TX` is associated with one and only one primary-to-secondary connection and its actions apply to that particular instance of data link connection.

The process `InlMan_TX` actually comprises a user data queue and it can perform some management functions on this queue. The latter is filled with user data blocks coming in from the DLS user via DATAreq type signals carrying `Data` as parameter. Since the queue performs only first-in-first-out (FIFO) actions, the order of the user data blocks stored is always guaranteed. Furthermore, since the queue is assumed to be of finite size (this is different from a pure SDL approach), the process should by some means inform the DLS user in case a queue overflow error occurs or is about to occur. The DLS user should then stop issuing DATAreq primitives and wait until the overflow situation has cleared. This is not shown in the specifications that follow.

Queue management

As is seen, the DLS user fills the queue with user data blocks that are to be sent in I frames. Without being previously sent, such a user data block is not labelled. Labelling is performed only by the sending of a user data block in an I frame. The label appointed to such blocks shall consist of the number VS at the time of sending the particular I frame (this number is of course equal to the NS that is included in the I frame control field). The queue is kept up to date by removing all user data blocks labelled with a number that is less than VSU, the number of the lowest unacknowledged I frame. This action requires a continuous knowledge of the value of VSU (which is contained in the process `SR`). Furthermore, the process `InlMan_TX` should also have knowledge of the value of VS on a continuous basis.

With this in mind, the queue could be built up like shown in the figure on the right. It should be noticed, however, that the numbering scheme is cyclic, which has to be accounted for if it is to be determined which user data blocks are outstanding, which ones are acknowledged and/or which ones are to be retransmitted. Such actions are accounted for by the macro definitions `Lowest_Unacknowledged` and `Checkpoint` (see elsewhere). At last, part of the queue management consists also of the administration of the value of IC, the information frame counter. The process `InlMan_TX` continuously has to account for the number of user data blocks that are stored in the queue but not yet sent in an I frame, and report this to the process `TX` (the parameter IC is actually revealed to this process).

The means by which this process performs its task is left to the reader. While reading the rest of the specification, one can however assume a correct and sufficiently fast functioning by this process. At last, once the process `TX` stops to exist, this process will also be stopped.
The process 'Setup' exists at system creation time and is responsible for the creation (instantiation) of the processes 'TX' and 'SR'. The process shall continuously monitor the issuing of a CONNreq primitive by the DLS User, indicating the initiation of a DLC establishment attempt. The CONNreq primitive shall carry and address (Adr) and a mode parameter (all other OOS parameters are implied or known on an a priori basis). The Adr parameter shall indicate the address of the secondary station to which a DLC is to be established. (Note that there is no explicit DLSAP address-to-secondary address conversion applied here). The process shall, in cooperation with the DLS User, be responsible for the limitation of only one secondary station in ARM at a time. The Mode parameter shall indicate either UNC (for normal response mode operation) or UAC (for asynchronous response mode operation). Both the Adr and the Mode parameters are passed to the 'SR' and 'TX' process as formal parameters. 

DCL Adr Byte; /*Addresses 00000000 */
/*and 11111111 are */
/*reserved. */
Mode ModeType;

Only one secondary station at a time shall be put in ARM.
PROCESS TX
FPAR Adr Byte,
Mode ModeType;

DCL VIEWED discREQ, resetREQ,
frmrREC, LocBusy, RemBusy,
RNRSent, TsExp, TdExp, TrExp,
TpExp Boolean;
VIEWED PF Bit;
VIEWED Ns, Nd, Nr, VS, VR,
VSU, IC Natural;
NS, NR, Diff Natural;
Data UserData;

SYNONYM NsMax, NdMax, NrMax
Natural = EXTERNAL;
WinSize Natural = EXTERNAL;

/* NsMax, NdMax, and NrMax represent the retransmission counters for mode-setting, disconnect, and resetting commands, respectively. These commands are defined outside the system boundaries, and are, hence, declared to be external variables. */

/* Process TX (Transmit) starts in state 'Idle' where early detection of a DISCreq primitive is affected. If a DISCreq primitive has been issued by the DLS User, both the instantiated processes SR and TX will stop to exist. The process TX shall inform the process SR about the early hang-up via the signal 'TX_Disc'.

If no DISCreq has been issued by the DLS User, which is the 'normal' case in a DLC establishment attempt, the receive process (RX) shall be instantiated and a mode-setting command shall be sent. The mode-setting command, either SNRM or SARM, depends on the 'Mode' of operation. This type of mode is given as a formal parameter to each instantiated process SR, TX, and RX. The mode-setting command is always sent with the P bit set to 1. This is also indicated to the process SR. In order to detect no-reply or lost-reply occurrences, the timer Ts shall be set and the retransmission counter Ns shall be incremented by one. The process TX shall move to the next state 'Outgoing Connection Pending'. */
After having sent a mode-setting command, the primary shall have to wait for the secondary station's response or for further instructions by the DLS User. If the secondary station accepts the DLC establishment attempt, it will send a UA response. The receipt of this response is forwarded to the process TX by the process RX. Without further interruptions, the primary station shall set its send state variable V(S) and receive state variable V(R) (effected in process RX) to 0 and enter the 'Data Transfer Ready' state. If a DM response is received by the process RX, this will be forwarded to process TX. In that case, the secondary station has rejected the DLC establishment attempt and the process TX will stop. If a DISC req primitive is issued, a DISC command will be sent to inform the secondary station about the pending hang-up. If the mode-setting command or the secondary station's reply gets lost, the timer Ts will run out and the mode-setting command may be retransmitted. This may be repeated until a valid response is obtained from the secondary station or until the mode-setting command is retransmitted a fixed, system-dependent maximum number of times (NsMax). The latter indicates that the secondary station is not physically connected or is malfunctioning.
The continuous signals with the highest priority are DISCreq, RSETreq and the receipt of an FRMR frame. If any of these conditions occurs, they must be actioned (in the order of priority). A DISCreq shall cause the primary to send a DISC frame. A RSETreq or a received FRMR frame shall cause the primary station to issue a mode-setting command.

The next-highest priority has the expiry of timer Tp, which is set on the transmission of a command with the P bit set to 1. If such a command or the associated response (F=1) gets lost, the timer Tp will expire. Since P and F bits always occur in pairs and since checkpointing is our main retransmission criterion, it is ultimately important to detect the loss of a P/F pair, which is effected by this timer. The primary station shall enquire status with a supervisory frame, i.e. RR or RNA. Depending on the local station's busy condition, this will be an AA or an RNA frame, though, both with the P bit set to 1. Also, VSP shall be updated, since, again, a P=1 frame is sent. (Note: V(S)-1 is taken here instead of V(S), since after transmitting and 1 frame, V(S) is incremented by 1.)
If none of the other higher-priority occurrences has taken place, two options are open, regarding the transmission of information frames by the primary station. First (priority 5), the primary station may have data ready to send in the InfMan_TX queue. This is indicated by the positive value of the number of data 'packets' counter 'IC'. In normal operation, the primary station shall send this data in I frames, which is effected in the macro definition of 'Next_Primary_I_Frame'. It is, however, possible that the remote station is busy (RemBusy is 'true'), and the primary station may not send any I frames, but will, instead, enter the 'Remote_Station_Busy' macro and send a supervisory frame (RR or RNR). If the remote station is not busy, but the local station is busy, the macro definition of 'Local_Station_Busy' will be entered and the primary station shall send an RNR frame (if it is allowed to). Afterwards, it will return to the normal sequence of operations; a station may continue sending I frames after having sent an RNR frame. If the primary station has no information available for transmission (priority 6), it will normally send an RR frame to indicate its status to the remote station and to acknowledge previously received I frames. In this case, also, the remote and/or the local station can indicate a busy condition and appropriate action has to be taken. In case of only the local station busy, the primary station shall, after sending an RNR frame return to its original state, and shall not continue sending I frames, since there is no information available for transmission.
After having received a DISC req primitive from the DLS User, the process TX will send a DISC frame and turn to this state. If a response (DM or UA, with or without the F bit set to 1) from the secondary station is received correctly, the DLC release attempt is successful and this primary station and the associated secondary station are disassociated from any form of data link connection (i.e., stopping the process instances). It is, however, possible that the issued DISC frame or the associated secondary response gets lost. Then, timer Td will run out and error recovery is affected by the retransmission of the DISC command (always with the P bit set to 1). This may continue until a valid response is received from the secondary station or until the DISC command is retransmitted a fixed, system-dependent maximum number of times (NdMax).
Primary station - process TX

DLS User Initiated Reset Pending

- view(disc_REQ,SR)=True;
  PRIORITY 1
  Send DISC

- view(TrExp,SR)=True;
  PRIORITY 2
  view(Nr,SR)

  - (NrMax)
    Set_Nd(0)
    Poll DISC
  - (<NrMax)
    mode
    - (UAC)
      SNRMA(adr,1)
    - (UNC)
      SARM(adr,1)
      Retransmission with the P bit set to 1

- view(discREQ,SR)
  - (True)
    UA(Adr,)
  - (False)
    Set_VS(0)
    Data Transfer Ready

Send DISC

- view(diSC,SR)=True;
  PRIORITY 1; ~vlel(diSC,SR)=True;
  PRIORITY 2
  View
  disc_REQ,SR)

- Set_Timer_Tr
- Inc_Nr
- P_Out
- DLS User Initiated Reset Pending

- DM(Adr,)
- TX_Disc
- X
Primary station - process TX

PROCESS TX

FRMR Frame Received

\( \text{view(disc_{REQ,SR})=True; } \)
\( \text{PRIORITY 1} \)
Send DISC

\( \text{view(TrExp_{SR})=True; } \)
\( \text{PRIORITY 2} \)
\( \text{view(Nr,SR)} \)
\( (\text{NrMax}) \)
\( (\text{<NrMax}) \)
Send DISC

\( \text{view(disc_{REQ,SR})} \)
\( (\text{True}) \)
Poll DISC

\( \text{view(disc_{REQ,SR})} \)
\( (\text{False}) \)
Mode

\( \text{SNRM(Adr,1)} \)
Set_TIMER_T

\( \text{SARM(Adr,1)} \)
Inc_Nr

\( \text{P_Out} \)
FRMR Frame Received

UA(Adr,)

\( \text{Set_VS(0)} \)
Data Transfer Ready

\( \text{Retransmission with the P bit set to 1} \)
The process RX (Receive) primarily monitors HDLC frames coming in from the associated secondary station (identified by Adr). The process is instantiated whenever the TX process has received a CONNreq and no DISCreq has been issued before transmitting a mode-setting command. On receipt of a UA response the DLC establishment can be completed, which is confirmed to the DLS User with a CONN-conf. Receipt of a DM response indicates that the connection attempt has failed, and the process has to be stopped. The issuing of a DISCreq by the DLS User is continuously monitored and actioned. At last, a transmitted mode-setting command or its reply can get lost, resulting in the expiry of Timer Ts. This will normally enable the retransmission of the mode-setting command, unless it has been retransmitted a maximum number of times, or a DISCreq occurs. */
In the 'Data Transfer Ready' state, the process RX can receive I, RR, RNR, and FRMR frames. During operation in this mode, the issuing of a DISCreq or RSETreq primitive by the DLS User is continuously monitored and acted upon. On receipt of (error-free) I frames, the value of V(R) shall be incremented by one if N(S) equals V(R), i.e., if the I frame is in-sequence. On receipt of RR or RNR frames, clearance or busy indications, respectively, from the remote station should be set. VSU updating and checkpointing are performed on a regular basis, namely on the receipt of an I, RR or RNR frame and on receipt of an F=1 frame, respectively.*
Primary station - process RX

PROCESS RX

Data Transfer Ready

I(Adr,NS,0,Data)

view(discREQ,SR)

F_In

Rem_Clear

Rem_Clear

view(discREQ,SR)

view(setREQ,SR)

view(VR,SR)

Inc_VR

(DataAnd(Data)

Lowest_Unacknowledged

Data Transfer Ready

Lowest_Unacknowledged

Lowest_Unacknowledged

Lowest_Unacknowledged

Checkpoint

Exit

Restart_Timer

Entry

Entry

Entry

Exit

Entry

Exit

Entry

Entry

Exit

Entry

Exit
In this state, the primary station has previously received a DISCreq primitive from the DLS User, in order to gracefully clear up the DLC. Therefore, the primary station has sent a DISC command to the associated secondary station. If the secondary station is in NDM or ADM, as appropriate, it will return a DM response with or without the F bit set to 1, depending on the type of disconnected mode. If the secondary station was in an operational mode at the time the DISC command was received, it will return a UA response with or without the F bit set to 1, depending on the type of operational mode.

The receipt of any of these responses is forwarded to the process TX, so it can also stop to exist. If the DISC command is not received correctly, or if the response (UA or DM) gets lost, the timer Td will run out and appropriate recovery is made possible by the retransmission of the DISC command. This may continue until a DM or UA response is received correctly from the secondary station or until the DISC command has been retransmitted a fixed, system-dependent number of times (NdMax). In the latter case, the connection will be assumed to have ceased to exists, though it should be noted that this is a very rare occasion and does not lead to graceful (or correct) DLC release. */
110 Primary station - process RX

DLS User Initiated Reset Pending

- Remote station rejects the resetting attempt

PRIORITY 1

ResetTimer_Tr

Outgoing Disconnection Pending

PRIORITY 2

view(TrExp_,SR)=True;

view(Nr,SR)≤NrMax

view_(discREQ,SR)

(True)

(False)

DISCind

Outgoing Disconnection Pending

Outgoing Disconnection Pending

Outgoing Disconnection Pending

DLS User Initiated Reset Pending

UA(Adr,0)

DM(Adr,)

UA(Adr,1)

UA(Adr,0)

DM(Adr,)

F_In

ResetTimer_Tr

view_(discREQ,SR)

(True)

(FALSE)

RSETconf

Set_VR(0)

Set_VSU(0)

Data Transfer Ready
In this state, the primary station has previously received an FRMR frame from the secondary station and has sent a mode-setting command (SNRM or SARM, as appropriate). The secondary station can respond with a UA response, though it cannot reject this mode-setting action with a DM response as is the case in a DLS User Initiated resetting sequence. Still, the DLS User can issue a DISCreq at all stages and this primitive has to be monitored continuously. Also, the mode-setting command or the UA response can get lost and the timer Tr will expire. This will initiate the right resetting and corrective actions at the primary station. */
PROCESS SR
FPAR Adr Byte,
Mode ModeType;

DCL REVEALED discREQ, rsetREQ, frmrREC, LocBusy, RemBusy,
RNRsent, TsExp, TdExp, TrExp, TpExp Boolean = False;
REVEALED Ns, Nd, Nr, VS, VR, VSU, VSP Natural = 0;
REVEALED PF Bit = 1; Val Natural;
SYNONYM Mod Natural = EXTERNAL;
TS, TD, TR, TP Duration = EXTERNAL;

1(4) Idle

rsetREQ=False

discREQ=True

rsetREQ=True

frmrREC=True

rsetREQ=False

frmrREC=False

frmrREC=False

frmrREC=False

frmrREC=False

Idle

Idle

Idle

Idle

Idle

Inc_Ns

Inc_Nd

Inc_Nr

P_Out

F_In

Ns=Ns+1

Nd=Nd+1

Nr=Nr+1

PF=0

PF=1

Idle

Idle

Idle

Idle

Idle

Idle
It is important to note here that the management of the variable 'LocBusy' is actually effected in the 'DataMan_RX' channel substructure's process 'InlMan_RX'. This process should recognize when the receiver queue is full and cannot accept any further I frames from the remote station. If such a condition occurs, it forwards a 'Local_Station_Busy' signal to the process SR. If, after such a signalling, the condition is no longer valid, it signals a 'Local_Station_Cleared' to the process SR. The channel by which such signals are transported is not shown and the process 'InlMan_RX' is referenced (see elsewhere). *
All frame rejection conditions are reported to the DLS user by the signal 'Response_Rejection_Reported'. The detection of an invalid N(R) is carried out in the process RX of this byte controller part. All other rejection conditions should be detected at the bit controller part and are redirected immediately to the DLS user.

The DLS user should normally issue a RSETreq primitive following such a frame reject condition (i.e., the receipt of the 'Response_Rejection_Reported' signal), or it may issue a DISCreq primitive if the rejection condition cannot be cleared by a resetting action.
The processes contained in this block specify the exact HDLC procedural behaviour (on DLPDU level) and the upper layer interface of a secondary station, for both normal response mode and asynchronous response mode operation. All processes exist at system creation time. The data management (DataMan) processes are referenced; a textual specification is given elsewhere. (*)
SUBSTRUCTURE DataMan_RX

""The substructure 'DataMan_RX' contains one process named 'InfMan_RX' which is created by the process 'RX'.

The main task of the process is to queue the received (error-free) frames from the remote data station. In particular, the (user data contained in) I frames. If the maximum number of I frames that can be stored is reached, the process will signal a 'Local_Station_BUSY' to the process 'SR'. If this condition is no longer true, a 'Local_Station_CLEARED' signal will be forwarded to the process 'SR'. The maximum number of I frames that can be stored is a system-dependent parameter, which is not specified here.

The means by which this process performs its task is left to the reader. While reading the rest of the specification, one can however assume a correct and sufficiently fast functioning of this process. At last, once the process 'RX' stops to exist, the process 'InfMan_RX' will also be stopped.""

SUBSTRUCTURE DataMan_TX

""The substructure 'DataMan_TX' contains one process named 'InfMan_TX' which is created by the process 'TX'.

The process 'InfMan_TX' actually comprises a user data queue and it can perform some management functions on this queue. The latter is filled with user data blocks coming in from the DLS user via DATAreq type signals carrying 'Data' as parameter. Since the queue performs only first-in-first-out (FIFO) actions, the order of the user data blocks stored is always guaranteed. Furthermore, since the queue is assumed to be of finite size (this is different from a pure SDL approach), the process should by some means inform the DLS user in case a queue overflow error occurs or is about to occur. The DLS user should then stop issuing DATAreq primitives and wait until the overflow situation has cleared. This is not shown in the specifications that follow.

Queue management

As is seen, the DLS user fills the queue with user data blocks that are to be sent in I frames. Without being previously sent, such a user data block is not labelled. Labelling is performed only by the sending of a user data block in an I frame. The label appointed to such blocks shall consist of the number VS at the time of sending the particular I frame (this number is of course equal to the NS that is included in the I frame control field). The queue is kept up to date by removing all user data blocks labelled with a number that is less than VSU, the number of the lowest unacknowledged I frame. This action requires a continuous knowledge of the value of VSU (which is contained in the process SR). Furthermore, the process 'InfMan_TX' should also have knowledge of the value of VS on a continuous basis.

With this in mind, the queue could be built up like shown in the figure on the right. It should be noticed, however, that the numering scheme is cyclic, which has to be accounted for if it is to be determined which user data blocks are outstanding, which ones are acknowledged and/or which ones are to be retransmitted. Such actions are accounted for by the macro definitions 'LOWest_Unacknowledged' and 'Checkpoint' (see elsewhere). At last, part of the queue management consists of the administration of the value of IC, the information frame counter. The process 'InfMan_TX' continuously has to account for the number of user data blocks that are stored in the queue but not yet sent in an I frame, and report this to the process 'TX' (the parameter IC is actually revealed to this process).

The means by which this process performs its task is left to the reader. While reading the rest of the specification, one can however assume a correct and sufficiently fast functioning by this process. At last, once the process 'TX' stops to exist, this process will also be stopped.""
The secondary station shall start (at system set-up) in the state 'Idle'. In this state, the receipt of a mode-setting command (either DISC, SNRM, or SARM) shall be monitored continuously. On receipt of an SNRM or SARM command, as appropriate, the station shall jump to the next state, unless its is prematurely interrupted by a received DISC command or no valid response opportunity exists (in UNC). In the first case, the station will respond with a DM frame (if possible) and stay in the 'Idle' state. If a command, other than a mode-setting command, is received with the P bit set to 1, the secondary station shall return a DM response which has the F bit set to 1. Other commands with the P bit set to 0 shall be ignored at this stage. */
The secondary station has previously issued a CONNind to the DLS User (effected by the RX process) and is waiting for further 'instructions' by the DLS User. The latter can respond with a CONNresp, indicating it is willing to accept the DLC establishment. The secondary station will then send a UA response, with or without the F bit set to 1 (depending on the mode and the value of PF) and initialize the relevant parameters for information transfer. The DLS User can, however, also react with a DISCreq, indicating it does not or cannot accept the DLC establishment and the secondary station shall send the DM response (if possible). It is also possible that the DLS Provider signals that the other DLS User has decided to break up the DLC establishment attempt, which is effected by the receipt of a DISC command. In this case, the secondary station will also send a DM frame in response. The receipt of a command, other than a mode-setting command, with the P bit set to 1 shall be responded to with a DM frame with the F bit set to 1, and the station will signal its disconnection action to the SR process by TX_Disc. Again, any other received commands with the P bit set to 0 shall be ignored in this stage, i.e., the secondary station shall not establish a frame rejection condition in a non-operational mode.
PROCESS TX

Data Transfer Ready

Entry

view(disc REC,SR)=True
PRIORITY 1

Send UA Response
Exit

TX_Disc

Data Transfer Ready

Entry

view(srm REC,SR)=True
PRIORITY 2

view(PF,SR)
(0)

DLS Provider Initiated Reset Pending

Data Transfer Ready

view(srm REC,SR)=True
PRIORITY 2

select if (view(Mode SR)=UNC)

view(srm REC,SR)=True
PRIORITY 2

select if (view(Mode SR)=UAC)

DLS Provider Initiated Reset Pending

view(W,SR)_-1;
PRIORITY 3

view(X,SR)_-1;
PRIORITY 4

view(Y,SR)_-1;
PRIORITY 5

view(Z,SR)_-1;
PRIORITY 6

Exit

Recorded Response

Send_FRMR_Response

Exit

Frame Rejection Reported

"/ See the MACRO 'Send_FRMR_Response' for procedures on the use of the parameters W, X, Y, and Z. */
/* For additional comments: see the primary station's process TX */

view(IC_,
  view(InfMan_TX)>0;
  view(RemBusy,SR)
) (True)

view(IC_,
  view(InfMan_TX)=0;
  PRIORITY 8
) (True)

view(RemBusy,SR)
(FALSE)

Entry

Remote Station Busy

Exit

Data Transfer Ready

view(RemBusy,SR)
(FALSE)

Entry

Local Station Busy

Exit

Entry

Local Station Busy

Exit

Entry

Send RR Frame

Exit

Data Transfer Ready

Next Secondary_I Frame

Exit

Data Transfer Ready
The secondary has previously issued a RSETind primitive to the DLS User to inform the latter about the received SNRM or SARM command. If the DLS User accepts the resetting initiative, it will issue a RSETresp primitive to the secondary station and the latter will report this to the primary station by a UA response, unless the primary station has already issued a DISC command. If the DLS User does not or cannot accept the resetting initiative, it will issue a DISCreq primitive and the secondary station will send a DM response in return to the primary station.
On receipt of an SNRM or SARM, as appropriate to the current mode, the secondary station shall send a UA response and return to the 'Data Transfer Ready' state; the frame rejection occurrence is then resolved. The FRMR response shall be repeated at each respond opportunity (chosen to be the possibility of returning an F bit set to 1) until recovery is effected by the primary station. See the macro 'Send_FRMR_Response' for more details on the usage of the W, X, Y, and Z parameters. */
The secondary station shall start (at system set-up) in the state 'Idle'. In this state, the receipt of a mode-setting command (either DISC, SNRM, or SARM) shall be monitored continuously. On receipt of an SNRM command, the mode will be set to UNC, and a CONNind will be issued to the DLS User, unless a DISC frame has been received before this primitive could be issued, or, unless the station has no normal response opportunity and has to wait for a primary station's poll. If an SARM command is received, the station will enter the UAC mode and a CONNind will be issued to the DLS User, unless a DISC frame has been received before doing so. If the DLC establishment attempt is not interrupted by a DISC frame, and a secondary response opportunity exists, the process will turn to the state 'Incoming Connection Pending'. If a command, other than a mode-setting command, with the P bit set to 1 is received, the secondary shall respond with a OM frame (which is effected by the TX process). Other commands with the P bit set to 0 shall be ignored at this stage. */
The secondary station has previously indicated the DLC establishment attempt to the DLS User via a CONNind primitive. It is now waiting for further instructions by the DLS User. If the DLS User accepts the DLC establishment attempt, it will issue a CONNresp to the DLS Provider (i.e., this station) and the secondary station will jump to the state 'Data Transfer Ready', after it has effected some initializations. If the DLS User rejects the DLC establishment attempt, it will issue a DISCreq and the secondary station will return to its initial state. The receipt of a DISC frame, i.e., a DLS Provider DLC release, shall always force the secondary station to report this occurrence to the DLS User and return to its initial state. Again, any other command with the P bit set to 1 shall force the station to return a DM response (F=1) and return to its initial state. Furthermore, any other command with the P bit set to 0 shall be ignored.
126 Secondary station - process RX

PROCESS RX

- Data Transfer Ready

- View (disc.REC,SR)=True
  - PRIORITY 1
  - Disc_rec

- View (PF,SR)
  - (1)
  - View (Mode,SR)
    - (UAC)
    - (UNC)

- Return to the original state

- DiscInd

- Idle

- Data Transfer Ready

- View (W,SR)=1
  - PRIORITY 3
  - View (X,SR)=1
  - PRIORITY 4
  - View (Y,SR)=1
  - PRIORITY 5
  - View (Z,SR)=1
  - PRIORITY 6

- Select if (view(Mode,SR)=UAC)

- RSETInd

- DLS Provider Initiated Reset Pending

- Data Transfer Ready

- Send FRMR Response 7 for procedures on the use of the parameters W, X, Y, and Z.

* See the MACRO "Send_FRMR_Response" for procedures on the use of the parameters W, X, Y, and Z.*
For additional comments: see the primary station's process RX, which performs basically the same functions in the 'Data Transfer Ready' state. Small differences left for the reader. */
The secondary has previously issued a RSETind to the DLS User. If the latter accepts the resetting attempt (by a RSETresp), the secondary will initialize VR(0) and VSU and return to the 'Data Transfer Ready' state. The resetting action can be 'overruled' by a DISC req primitive (DLS User rejection) or a DISC frame (DLS Provider initiated DLC release). */
After a frame rejection condition has been reported and an associated FRMR has been returned by the secondary station, the secondary station will move to this state 'Frame Rejection Reported'. The station is waiting for a mode-setting command from the primary station in order to 'refresh' the data link communication. Depending on the mode the station is in, an SNRM or SARM command will be actioned immediately (if possible, see also process TX), and the relevant information transfer variables (VR) and VSU (in the case of process RX) will be reset. The receipt of a DISC command will, of course, have to be dealt with immediately, with the highest priority.*
Once a secondary station has established a FRMR exception condition, no additional I frames shall be accepted, except for the examination of the state of the P bit (updating of the variable PF) and the value of the N(R) field (updating of the variable VSU), until the condition is cleared by the primary station issuing a mode-setting command (either DISC, SNRM or SARM, as appropriate. See previous page.). Naturally, also RR and RNR frames should be accepted for this purpose, i.e., the variables P_In, Rem_Clear, and Rem_Busy should be updated, and the 'Lowest_Unacknowledged' macro should be called in to consequently update the number of the lowest unacknowledged I frame. Checkpointing should not be performed here, since there is no facility to (re)transmit I frames and the local I frame numbering scheme will be reset after clearance of the frame rejection condition. For the same reasons, and because the value of the N(S) field should not be examined in this very state, the receive state variable VIR) shall not be updated in this situation. No information will be accepted, so no DATAind primitives should be issued as is the case when an error-free and in-sequence I frame is received. */
DCL REVEALED snmREC, sarmREC, discREC, LocBusy, RemBusy, RNRSent Boolean;
REVEALED PF, W, X, Y, Z Bit;
REVEALED VS, VR, VSU, VSP Natural;
REVEALED Mode ModeType;
Val Natural;
UXC ModeType;

SYNONYM Adr Byte = EXTERNAL;
Mod Natural = EXTERNAL;
It is important to note here that the management of the variable 'LocBusy' is actually effected in the 'DataMan_RX' channel substructure's process 'InMan_RX'. This process should recognize when the receiver queue is full and cannot accept any further I frames from the remote station. If such a condition occurs, it forwards a 'Local_Station_Busy' signal to the process SR. If, after such a signalling, the condition is no longer valid, it signals a 'Local_Station_Cleared' to the process SR. The channel by which such signals are transported is not shown and the process 'InMan_RX' is referenced (see elsewhere). */
"Note that the values of W, X, and Y are changed by the bit processor part on the occurrence of the respective frame rejection conditions. The signals Set_W to Set_Y are, hence, not explicitly specified. The signal Set_Z is sent by the receiver process. */
The processes contained in this block specify the exact HDLC procedural behaviour (on DLPDU level) and the upper layer interface of a combined station. All processes exist at system creation time. The data management (DataMan) processes are referenced; a textual specification is given elsewhere.
SUBSTRUCTURE DataMan_RX

/* The substructure 'DataMan_RX' contains one process named 'InfMan_RX' which is created by the process 'RX'.

The main task of the process is to queue the received (error-free) frames from the remote data station, in particular
the (user data contained in) I frames. If the maximum number of I frames that can be stored is reached, the process
will signal a 'Local_Station_Busy' to the process 'SR'. If this condition is no longer true, a 'Local_Station_Cleared'
signal will be forwarded to the process 'SR'. The maximum number of I frames that can be stored is a system-dep­
dendent parameter, which is not specified here.

The means by which this process performs its task is left to the reader. While reading the rest of the specification,
one can however assume a correct and sufficiently fast functioning of this process. At last, once the process 'RX'
stops to exist, the process 'InfMan_RX' will also be stopped. */

SUBSTRUCTURE DataMan_TX

/* The substructure 'DataMan_TX' contains one process named 'InfMan_TX' which is created by the process 'TX'.

The process 'InfMan_TX' actually comprises a user data queue and it can perform some management functions on
this queue. The latter is filled with user data blocks coming in from the DLS user via DATAreq type signals carrying
'data' as parameter. Since the queue performs only first-in-first-out (FIFO) actions, the order of the user data blocks
stored is always guaranteed. Furthermore, since the queue is assumed to be of finite size (this is different from a pu­
re SDL approach), the process should by some means inform the DLS user in case a queue overflow error occurs or
is about to occur. The DLS user should then stop issuing DATAreq primitives and wait until the overflow situation has
cleared. This is not shown in the specifications that follow.

Queue management

As is seen, the DLS user fills the queue with user data blocks that are to be sent in I frames. Without being previous­
ly sent, such a user data block is not labelled. Labelling is performed only by the sending of a user data block in an I
frame. The label appointed to such blocks shall consist of the number VS at the time of sending the particular I frame
(this number is of course equal to the NS that is included in the I frame control field). The queue is kept up to date by
removing all user data blocks labelled with a number that is less than VSU, the number of the lowest unacknowl­
dged I frame. This action requires a continuous knowledge of the value of VSU (which is con­tained in the process SA). Furthermore, the process 'InfMan_TX' should also have know­ledge of the value of VS on a continuous basis.

With this in mind, the queue could be built up like shown in the figure on the right. It should be noticed, however, that the numering scheme is cyclic, which has to be accounted for if it is to be determined which user data blocks are outstand­
ing, which ones are acknowledged and/or which ones are to be retransmitted. Such actions are accounted for by the macro definitions 'Lowest_Unacknowledged' and 'Checkpoint' (see else­where). At last, part of the queue management consists also of the administration of the value of IC, the information frame counter. The process 'InfMan_TX' continuously has to account for the number of user data blocks that are stored in the queue but not yet sent in an I frame, and report this to the process 'TX' (the parameter IC is ac­tually revealed to this process).

The means by which this process performs its task is left to the reader. While reading the rest of the specification,
one can however assume a correct and sufficiently fast functioning by this process. At last, once the process 'TX'
stops to exist, this process will also be stopped. */
The following specification assumes a combined station with address 'A', while the other combined station is assigned address 'B'. Without further restrictions these addresses can be swapped. */
Combined station - process TX

PROCESS TX

Outgoing Connection Pending

UA(B,1) → Send DISC

view (discREQ,SR) → True → DM(B,1) → TX_Disc → Send DISC

(Total) → Set_VS(0) → Data Transfer Ready

view(P,SR) → (0) → DISC(B,1) → P_Out → Timer_Ts is reset at process RX

view(P,SR) → (1) → DISC(B,0) → TX_Disc → Idle

view(TsExp,SR) → True → TX_Disc → Idle

view(Ns,SR) → (<NsMax) → TX_Disc → Set Mode

view(discREQ,SR) → (False) → TX_Disc → Idle

view(discREQ,SR) → True → PRIORITY 1

SetTimer_Td → Set_Nd(1) → Outgoing Disconnection Pending

Combined station - process TX

PROCESSTX

Outgoing Connection Pending

\( \text{view}(\text{disc}_{-}\text{REC},\text{SR}) = \text{True}; \) PRIORITY 2
Entry
Balanced_DM_Response
Exit
Idle

\( \text{view}(\text{sabm}_{-}\text{REC},\text{SR}) = \text{True}; \) PRIORITY 3
UA(A,1)
ResetTimer_Ts
F_Out
Set_VS(0)
Data Transfer Ready

Any_other_command_(A,1)
DM(A,1)
TX_Disc
Idle

Any other command with the P bit set to 1
Incoming Connection Pending

- CONNresp
  - view(disc_REQ,SR)=True, PRIORITY 1
    - Entry
      - Balanced_DM_Response
  - view(disc_REQ,SR)=False
    - Exit
    - Idle

- CONNresp
  - view(disc_REQ,SR)=True, PRIORITY 2
    - Entry
      - Balanced_DM_Response
  - view(disc_REQ,SR)=False
    - Exit
    - Idle

- Balanced_DM_Response
  - view(F,SR)=0
    - Entry
    - UA(A,1)
    - F_Out
    - Set_VS(0)
    - Data Transfer Ready
    - Data Transfer Ready
    - Idle
  - view(F,SR)=1
    - Entry
    - UA(A,0)
    - Set_VS(0)
    - Data Transfer Ready
    - TX_Disc
    - Idle
PROCESS TX

Data Transfer Ready

view(disc_REQ,SR)=True;
PRIORITY 1

Send DISC

Entry

Balanced-UA_Response

Exit

TX_Disc

Idle

Data Transfer Ready

view(disc_REQ,SR)=True;
PRIORITY 2

Entry

view(sabm_REQ,SR)=True;
PRIORITY 3

Exit

DLS User Initiated Reset Pending

view(fmr_REQ,SR)=True;
PRIORITY 4

Entry

Send_SABM_Command

Exit

FRMR Frame Received

view(sabm_REQ,SR)=True;
PRIORITY 5

Entry

Send_SABM_Command

view(fmr_REQ,SR)=True;
PRIORITY 6

vlew(SRL)
PRIORITY 7

vlew(SRL)
PRIORITY 8

vlew(SRL)
PRIORITY 9

view(W,SR)_1;
PRIORITY 6

view(X,SR)_1;
PRIORITY 7

view(Y,SR)_1;
PRIORITY 8

view(Z,SR)_1;
PRIORITY 9

Entry

Send_Balanced_FRMR

Exit

Frame Rejection Reported

* See the macro "Send_Balanced_FRMR" for procedures on the use of the parameters W, X, Y, and Z. *
The next-highest priority has the expiry of timer \( T_p \), which is set on the transmission of a command with the \( P \) bit set to 1. If such a command or the associated response (\( F=1 \)) gets lost, the timer \( T_p \) will expire. Since \( P \) and \( F \) bits always occur in pairs and since checkpointing is the main retransmission criterion, it is ultimately important to detect the loss of a \( P/F \) pair, which is affected by this timer. The combined station shall enquire status with a supervisory frame, i.e. RR or RNR. Depending on the local station’s busy condition, this will be an RR or an RNR frame, though, both with the \( P \) bit set to 1. Also, VSP shall be updated, since, again, a \( P=1 \) frame is sent. (Note: \( V(S)-1 \) is taken here instead of \( V(S) \), since after transmitting and I frame, \( V(S) \) is incremented by 1.)

RR and RNR re-polled after time-out expiry

The subtraction performed is modulo \( \text{Mod} \)
If none of the other higher-priority occurrences has taken place, two options are open, regarding the transmission of information frames by the combined station. First (priority 11), the combined station may have data ready to send in the IntMan_TX queue. This is indicated by the positive value of the number of data 'packets' counter 'IC'. In normal operation, the combined station shall send this data in I frames, which is effected in the macro definition of 'Next_Combined_I_Frame'. It is, however, possible that the remote station is busy, and the combined station may not send any I frames, but will, instead, enter the 'Remote_Combined_Busy' macro and send a supervisory frame (RR or RNR). If the remote station is not busy, but the local station is busy, the macro definition of 'Local_Combined_Busy' will be entered and the combined station shall send an RNR frame (if it is allowed to). Afterwards, it will return to the normal sequence of operations; a station may continue sending I frames after having sent an RNR frame. If the combined station has no information available for transmission (priority 12), it will normally send an RR frame to indicate its status to the remote station and to acknowledge previously received I frames. In this case, also, the remote and/or the local station can indicate a busy condition and appropriate action has to be taken. In case of only the local station busy, the combined station shall, after sending an RNR frame return to its original state, and shall not continue sending I frames, since there is no information available for transmission.
DLS User accepts the attempt to reset the DLC

- **RSTresp**

- **Send DISC**

- **view**(disc_REQ,SR) = True

  - **PRIORITY 1**
  - Entry
  - Balanced_UA_Response
  - Exit
  - TX_Disc
  - Idle

- **view**(disc_REQ,SR) = False

  - **(False)**
  - **(False)**

- **view**(F,SR) = True

  - **PRIORITY 2**
  - Entry
  - Balanced_UA_Response
  - Exit
  - TX_Disc
  - Idle

- **view**(F,SR) = False

  - **(False)**

- **view**(F,SR) = (0)

- **UA(A,0)**

- **UA(A,1)**

- **F_Out**

- **Set_VS(0)**

- **Data Transfer Ready**
On receipt of an SABM command, the combined station shall send a UA response and return to the 'Data Transfer Ready' state; the frame rejection occurrence is then resolved. The FRMR response shall be repeated at each respond opportunity (the possibility of returning an F bit set to 1) until recovery is effected by the other combined station. The case of FRMR contention is not taken into account here. See the macro 'Send_FRMR_Response' for more details on the usage of the W, X, Y, and Z parameters. */
PROCESSTX

- **FRMR Frame Received**
  - **Priorities:**
    - **Priority 1:**
      - \( \text{view}(\text{disc}_{\text{REQ,SR}}) = \text{True}; \)
      - \( \text{Send DISC} \)
    - **Priority 2:**
      - \( \text{view}(\text{disc}_{\text{REC,SR}}) = \text{True}; \)
      - \( \text{Balanced}_{\text{UA Response}} \)
      - \( \text{Exit} \)
      - \( \text{TX Disc} \)
    - **Priority 3:**
      - **Entry**
      - \( \text{view}(\text{disc}_{\text{REC,SR}}) = \text{True}; \)
      - \( \text{Balanced}_{\text{UA Response}} \)
      - \( \text{Exit} \)
      - \( \text{Data Transfer Ready} \)
      - \( \text{Set Vs(0)} \)
    - **Priority 4:**
      - \( \text{view}(\text{TrExp}_{\text{SR}}) = \text{True}; \)
      - \( \text{view}(\text{disc}_{\text{REQ,SR}}) \)
      - \( (\text{NrMax}) \)
      - \( (\leq \text{NrMax}) \)
      - \( \text{view}(\text{disc}_{\text{REQ,SR}}) = \text{False}; \)
      - \( \text{SABM(B,1)} \)
      - \( \text{Set Timer_Tr} \)
      - \( \text{Inc Nr} \)
      - \( \text{P_Out} \)
      - \( \text{Frame Rejection Reported} \)

- **UA(B,1)**
  - \( \text{view}(\text{disc}_{\text{REQ,SR}}) = \text{True}; \)
  - \( \text{Set Vs(0)} \)
  - \( \text{Send DISC} \)
  - \( \text{Data Transfer Ready} \)
DCL VIEWED discREQ, discREC, rsetREQ, sabmREC, TsExp, TdExp, TrExp Boolean;
VIEWED Na, Nd, Nr, vs, VR, VSU, VSP Natural;
VIEWED W, X, Y, Z Bit;
NS, NR Natural;
Data UserData;
RFCF Byte;

SYNONYM A, B Byte = EXTERNAL;
NsMax, NdMax, NrMax Natural = EXTERNAL;

Any other command with the P bit set to 1
**PROCESS RX**

Outgoing Connection Pending

- view(disc_REC,SR)=True;
  - PRIORITY 2
    - DISCind
      - Idle
    - CONNconf
      - Set_VR(0)
      - Set_VSU(0)
      - Data Transfer Ready

- view(sabm_REC,SR)=True;
  - PRIORITY 3
    - Any other command

- Any other command with the P bit set to 1

Combined station - process RX
Combined station - process RX

PROCESS RX

Incoming Connection Pending

CONNrsp

view(discREQ,SR) = True;
PRIORITY 1

view(discREC,SR) = True;
PRIORITY 2

Idle

DISCind

Idle

Any other command with the P bit set to 1

Idle

Incoming Connection Pending

view(discREQ,SR) = True;

Any other command (A,1)

Any other command (A,1)

DISCind

Idle

Data Transfer Ready
ResetTimer_Tp

Outgoing Disconnection Pending

Data Transfer Ready

view(disc_REQ,SR)=True;
PRIORITY 1

view(disc_REQ,SR)=True;
PRIORITY 2

view(set_REQ,SR)=True;
PRIORITY 3

view(sabm_REQ,SR)=True;
PRIORITY 4

DISC_req

DISC_req

RESET_req

SABM_req

DisConnect

Idle

DLS User Initiated Reset Pending

ResetTimer_Tp

RSETind

DLS Provider Initiated Reset Pending

view(W,SR)=1;
PRIORITY 5

view(X,SR)=1;
PRIORITY 6

view(Y,SR)=1;
PRIORITY 7

view(Z,SR)=1;
PRIORITY 8

ResetTimer_Tp

Frame Rejection Reported

* See the macro "Send Balanced FRMR" for procedures on the use of the parameters W, X, Y, and Z. *
/* For additional comments: see the primary station's process RX, which performs basically the same functions in the 'Data Transfer Ready' state. Small differences left for the reader. */

...
PROCESS RX

Data Transfer Ready

I(B,NS,NR,1,Data)

F_In

Rem_Clear

Higher_Priority_Events

Exit

Reset_Timer_Tp

(NS)

Lowest_Unacknowledged

Entry

Checkpoint

Exit

Data Transfer Ready

I(B,NS,NR,0,Data)

Higher_Priority_Events

Entry

\( \text{view}(\text{VR},\text{SR}) \) (NS)

Inc_VR

DATAInd(Data)

Lowest_Unacknowledged

Data Transfer Ready

Entry

Exit

Entry

Exit
PROCESS RX

Outgoing Disconnection Pending

\[ \text{view}(\text{disc}_{\text{REC,SR}}) = \text{True}; \]
\[ \text{PRIORITY 1} \]
\[ \text{Idle} \]

\[ \text{view}(\text{TdExp}_{\text{SR}}) = \text{True}; \]
\[ \text{PRIORITY 2} \]
\[ \text{view}(\text{Nd,SR}) \]
\[ (\text{NdMax}) \]
\[ (\text{<NdMax}) \]
\[ \text{Outgoing Disconnection Pending} \]

DM(B)
UA(B)
DM(B)
UA(B)
Idle
Idle
Idle

Combined station - process RX
PROCESS RX

DLS Provider Initiated Reset Pending

- RSETresp
  - View (discREQ,SR) = True;
    - PRIORITY 1
    - Outgoing Disconnection Pending
      - DISC_rec
      - Set_VR(0)
  - (False)
  - View (discREC,SR) = True
    - DISC_rec

- (False)
- Data Transfer Ready
After a frame rejection condition has been reported and an associated FRMR has been returned by the combined station, it will move to this state "Frame Rejection Reported". The station is waiting for a mode-setting command from the remote station in order to 'refresh' the data link communication. A SABM command will be actioned immediately, and the relevant information transfer variables (V(R) and VSU in the case of process RX) will be reset. The receipt of a DISC command or primitive will, of course, have to be dealt with immediately, with the highest priority. /*
Once a combined station has established a FRMR exception condition, no additional I frames shall be accepted, except for the examination of the state of the P bit (updating of the variable F) and the value of the N(R) field (updating of the variable VSU), until the condition is cleared by the remote station issuing a mode-setting command (either DISC or SABM. See previous page.). Naturally, also RR and RNR frames should be accepted for this purpose, i.e., the variables P_In, Rem_Clear, and Rem_Busy should be updated, and the 'Lowest Unacknowledged' macro should be called in to consequently update the number of the lowest unacknowledged I frame. Checkpointing should not be performed here, since there is no facility to (re)transmit I frames and the local I frame numbering scheme will be reset after clearance of the frame rejection condition. For the same reasons, and because the value of the N(S) field should not be examined in this very state, the receive state variable V(R) shall not be updated in this situation. No information will be accepted, so no DATAind primitives should be issued as is the case when an error-free and in-sequence I frame is received. */
Combined station - process RX

PROCESS RX

- **FRMR Frame Received**
  - **UA(B,1)**
  - **UA(B,0)**
  - **UA(B,1)**
  - **UA(B,0)**
  - **F_in**
  - **ResetTimer_Tr**
  - **view(disc_REQ,SR) = True**
    - **view(disc_REQ,SR) = True; PRIORITY 1**
    - **view(disc_REQ,SR) = True; PRIORITY 2**
    - **view(sabm_REC,SR) = True; PRIORITY 3**
  - **view(disc_REC,SR) = True**
    - **DISC_rec**
    - **Outgoing Disconnection Pending**
    - **PAIOAITY1**
      - **view(disc_REC,SR) = True; PRIORITY 2**
      - **PAIOAITY2**
        - **view(disc_REC,SR) = True; PRIORITY 3**
      - **PAIOAITY3**
        - **view(disc_REC,SR) = True; PRIORITY 4**
      - **PAIOAITY4**
        - **NRMax**
          - **view(Nr,SR)**
            - **(false)**
            - **DISCind**
            - **Outgoing Disconnection Pending**
  - **Set_VR(0)**
  - **Set_VSU(0)**
  - **Data Transfer Ready**

- **(True)**
- **(False)**
Combined station - process SR

PROCESS SR

Idle

SetTimer_Ts

Ts

ResetTimer_Ts

SetTimer_Td

Td

set(Ts,TS)

TsExp=True

reset(Ts)

set(Td,TD)

TdExp=True

TsExp=False

Idle

TdExp=False

Idle

Idle

set(Tr,TR)

TrExp=True

reset(Tr)

set(Tp,TP)

TpExp=True

TpExp=False

Idle

TpExp=False

Idle

Idle

Tp

TpExp=True

Idle
It is important to note here that the management of the variable 'LocBusy' is actually effected in the 'DataMan_RX' channel substructure's process 'InfMan_RX'. This process should recognize when the receiver queue is full and cannot accept any further I frames from the remote station. If such a condition occurs, it forwards a 'Local_Station_Busy' signal to the process SA. If, after such a signalling, the condition is no longer valid, it signals a 'Local_Station_Cleared' to the process SR. The channel by which such signals are transported is not shown and the process 'InfMan_RX' is referenced (see elsewhere). */
**Combined station - process SR**

**PROCESS SR**

```
Idle

TX_Disc

Set_Ns(Val)

Set_Nd(Val)

Set_Nr(Val)

Inc_Ns

Start-up

Ns=Val

Nd=Val

Nr=Val

Ns=Ns+1

Idle

Idle

Idle

Inc_Nd

Inc_Nr

P_In

P_Out

F_In

Nd=Nd+1

Nr=Nr+1

F=1

P=0

P=1

Idle

Idle

Idle

Idle

/* For comments on W, X, Y, and Z: see secondary stations process SR. */

F_Out

Set_W(Val)

Set_X(Val)

Set_Y(Val)

Set_Z(Val)

F=0

W=Val

X=Val

Y=Val

Z=Val

Idle

Idle

Idle

Idle

Idle
```
DCL  REVEALED discREQ, discREC, rsetREQ, sabmREC, LocBusy, RemBusy, RNRSent, TsExp, TdExp, TrExp, TpExp Boolean;
REVEALED Ns, Nd, Nr, Vs, VR, VSU, VSP, Natural;
REVEALED P, F, W, X, Y, Z Bit;
Val Natural;
SYNONYM  A, B Byte = EXTERNAL;
Mod Natural = EXTERNAL;
TS, TD, TR, TP Duration = EXTERNAL
MACRODEFINITION Send_OM_Response

/* This macro shall take care of the transmission of a DM response by a secondary station. If PF is 1, the DM response shall always be sent with the F bit set to 1. If PF is 0, a secondary station in UNC is not allowed to send the response (no normal response opportunity), and a secondary station in UAC may send the response with the F bit set to 0. In all cases, the process SR shall be warned about the disconnection by the 'TX_Disc' signal. */

MACRODEFINITION Send_UA_Response

/* This macro shall take care of the transmission of a UA response by the secondary station. If PF is 1, the response will be sent with the F bit set to 1. If PF is 0, the secondary station in UNC shall return to its original state and wait for a normal response opportunity, and a secondary station in UAC shall transmit the response with the F bit set to 0. */
MACRODEFINITION Balanced_DM_Response

This macro shall take care of the transmission of a DM response by a combined station. Since a response frame is being sent, the value of F has to be checked first in order to determine whether the final bit may be set to 1 or not. The process SR shall be informed about the DM response transmission by the signal "TX_Disc", meaning that the combined station normally assumes a disconnected mode after having sent this response. */

MACRODEFINITION Balanced_UA_Response

This macro shall take care of the transmission of a UA response by the combined station. If F is equal to 1, the response will be sent with the F bit set to 1 and the value of the value of F local to the process SR will be updated. If the value of F is 0, the response will be set with the F bit set to 0. */
In UNC, the restart condition for the command reply time-out function will be the receipt of an error-free frame with the F bit set to 0. In UAC, there will be no such restart condition. Hence, in this mode, the timer $T_p$ will not be restarted. */

*/ This macro will be called whenever a mode-setting command (either SNRM or SARM) shall be issued by the primary station. The mode-setting command will have the P bit set to 1 if this is allowed, i.e., if PF is 1. This polling action shall be reported to the process SR via the "P_Out" signal. Else, the secondary station may not be polled, so the mode-setting command will be sent with the P bit set to 0. In both cases, the timer $T_r$ will be set in order to detect a no-reply or lost-reply to the command previously sent. The accessory retransmission counter $N_r$ will take its initial value of 1, indicating that this is the first time in a sequence that the mode-setting command is being sent. */
General macro definitions

MACRODEFINITION Send_SABM_Command

Entry

view(P,SR) (0)

(1)

SABM(B,1) SABM(B,0)

P_Out

Timer Tr is also set when issuing a P=0 command!

Initial Nr value; SABM command now sent 1 time

SetTimer_Tr

Set_Nr(1)

Exit

/* This macro shall be called whenever a SABM command is to be issued by the combined station. The mode-setting command shall have the P bit set to 1 if this is allowed, i.e., if the variable 'P' local to the SR process is 1. This polling action shall be reported via the signal 'P_Out'. Else, the SABM command will be sent with the P bit set to 0. In both cases the timer Tr will be set in order to detect a no-reply or lost-reply to the command previously sent. The accessory retransmission counter Nr will take its initial value of 1, indicating this is the first time in a sequence that the mode-setting command is sent. */
MACRODEFINITION Local_Station_Busy

/* This macro definition will be invoked whenever the local station indicates a busy condition, i.e., a temporary inability to accept subsequent I frames. Normally, this condition shall be reported by the InfoMan_RX process, due to internal constraints, for example, receive buffering limitations. In this case, an RNR frame shall be transmitted to the remote station, with the N(R) number of the next I frame that is expected by the busy station. After entering this macro, the value of the PF variable shall be examined. If it is 1, a primary station shall set the timer Tp and report its polling action to the SR process. A secondary station shall only indicate that it has sent a final bit. Next, the value of VSP shall be updated to the value of V(S)-1, which is the value of the send sequence number of the last I frame sent, previous to this polling/finalizing action. (Note: V(S)-1 is taken here, instead of V(S), since, after sending an I frame, V(S) is incremented by 1.) At last, the sending of the RNR frame is indicated to the SR process, since the retransmission of it shall only take place at each P/F exchange. In case of PF=0, mainly the same actions occur with adjusted P/F bits. Of course, a secondary station in UNC may not send an RNR frame if it is not previously polled by the primary station. Furthermore, it is seen that the RNR frame shall only be sent with the P/F bit set to 0 if it is not yet previously sent, as stated above. */
The substraction performed is modulo Mod.

For further comments: see the macro definition 'Local Station Busy' which is rather similar. It should be noted, however, that the value of F is tested first, since, in ABM, if a combined station receives a command with the P bit set to 1, transmission of a response with the F bit set to 1 shall take precedence over the transmission of commands, with the exception of mode-setting commands. */
MACRODEFINITION Remote_Station_Busy

/* This macro definition shall be invoked whenever the remote station has reported a busy condition (via an RNR frame), meaning that it is unable to receive or continue to receive any further frames due to internal constraints, for example, receive buffering limitations. If the remote station has reported a busy condition, the local station can also be in a busy condition. This is tested first. Depending on this test one of two other macro definitions is called in. If the local station is also busy then the macro 'Local_Station_Busy' is called, in which the local station shall - if it is allowed to do so - report its busy condition to the remote station. If the local station is not busy, it will call the macro 'Send_RR_Frame' in which it will send an RR frame to report that it is able to receive further frames, or that a busy condition has cleared. */

MACRODEFINITION Remote_Combined_Busy

/* For further comments: see the macro definition 'Remote_Station_Busy'. */
This macro definition shall be invoked whenever the local station decides to transmit an RR frame with an updated N(R), acknowledging the correct receipt of all I frames number up to N(R)-1. After entering this macro, the value of the PF variable shall be examined. If it is 1, a primary station shall update the value of N(R), set the timer Tp and report its polling action to the SR process. A secondary station shall only update the value of N(R) and indicate that it has sent a final bit. Next, the value of VSP shall be updated to the value of V(S)-1, which is the value of the send sequence number of the last I frame sent, previous to this polling/finalizing action. (Note: V(S)-1 is taken here, instead of V(S) since, after sending an I frame, V(S) is incremented by 1.) If the value of PF is 0, a primary station shall send the RR frame with the P bit set to 0. Here, a secondary station shall not transmit the RR frame in NRM (no normal respond opportunity). In ARM, the secondary station may send the RR frame with the F bit set to 0. */
Send_Balance_CCRR

NR = view(VR, SR)  \( (0) \)

For further comments: see the macro definition 'send RR Frame' which is rather similar.

It should be noted, however, that the value of F is tested first, since, in ABM, if a combined station receives a command with the P bit set to 1, transmission of a response with the F bit set to 1 shall take precedence over the transmission of commands, with the exception of mode-setting commands. */
If a primary station is ready to send a next I frame, this macro will be called. First, the value of 'Diff' shall be calculated. This is the difference between V(S) and VSU, i.e., the number of outstanding I frames. This value should be smaller than a system-dependent parameter 'WinSize' (WinSize is at most Mod-1). If it is equal to WinSize, the I frame may not be sent and instead, an RR frame should be sent, indicating that the station is ready to receive further I frames and to acknowledge previously received I frames. If it is allowed to send the I frame, it has to be determined whether the frame should have the P bit set to 1 or 0. Furthermore, N(R) and N(S) should take their actual values and, after sending the I frame, V(S) should be incremented. In case of the P bit set to 1, the Tp timer should be set, 'P_Out' should be updated, and, the station should remember the current value of V(S) in VSP. */
MACRODEFINITION Next_Secoundary_L Frame

For comments regarding the use of 'Diff', the transmission of I frames, the setting of timer Tp, the use of VSP and the incrementing of V(S): see the macro definition 'Next_Primary_L Frame'. Furthermore, in case of UNC, the secondary station is chosen to transmit the F bit only in the last available or transmittable (Diff I) I frame, since it shall have to be quiet after sending the F bit set to 1. This is done to preserve a reasonable throughput. Of course, in this mode of operation, the secondary station may not transmit any available I frame(s) if PF is 0. */
Both subtractions shown are normal subtractions

\[ \text{Diff} = \text{Mod} + \text{view(VS,SR)} - \text{view(VSU,SR)} \]

Both subtractions shown are normal subtractions. 

\( V(S) \) shall not exceed \( N(R) \) of last received I frame by more than \( \text{Mod} - 1 \)

For further comments: see the macro definition 'Next_Primary_L Frame' which is rather similar. It should be noted, however, that the value of \( F \) is tested first, since, in ABM, if a combined station receives a command with the \( P \) bit set to 1, transmission of a response with the \( F \) bit set to 1 shall take precedence over the transmission of commands, with the exception of mode-setting commands. */
MACRODEFINITION Lowest_Unacknowledged

```
This macro definition takes care of the determination of the value of VSU, the variable containing the value of the send sequence number of the lowest unacknowledged I frame that is still in the (re)transmission queue. First, the current value of V(S) is compared with the current value of VSU. If V(S) is at least equal to VSU, the receive sequence number N(R) of the received I, RR, or RNR frame should be at least equal to VSU and at most equal to V(S). In that case, the value of VSU is updated to the value of N(R), indicating that all I frames numbered up to N(R)-1 are acknowledged. If V(S) is smaller than VSU, the receive sequence number of the received I, RR, or RNR frame should be at least equal to V(S) and at least equal to VSU. In that case, the value of VSU is updated to the value of N(R), indicating that all I frames numbered up to N(R)-1 are acknowledged. If N(R) falls out of the valid range VSU ... V(S), this means that an invalid N(R) is received by the station, i.e., an N(R) which identifies an I frame which has previously been transmitted and acknowledged or an I frame which has not been transmitted and is not the next sequential I frame awaiting transmission. In that case, a frame rejection condition has occurred and appropriate action should be taken to resolve this malfunctioning. A primary station shall force the DLS User to issue a RSETreq (preferably) or a DISCreq primitive, in order to reset the communication. This enforcement is effected by the sending of a 'Response_Rejection_Reported' signal to the DLS User. A secondary/combined station should set the variable 'Z' to 1, indicating an invalid N(R) frame rejection condition has occurred. The next frame to be sent by the secondary/combined station will, in that case, be an FRMR response frame, with the associated rejection reason specified. (Optionally for a combined station.) */
```
In this macro definition, checkpoint calculations are made. First of all, VSP is compared with VSU to know the order in which they can appear. Next, VSP is compared with N(R), the receive sequence number of the received I, RR, or RNR frame. If the N(R) does not acknowledge at least all I frames transmitted by the local station previous to, and concurrent with, the last frame which was transmitted by the local station with the P/F bit set to 1, checkpoint retransmission(s) should be initiated. This initiation is effected by means of setting V(S) equal to N(R), i.e., the retransmission shall start with the oldest numbered unacknowledged I frame and the I frame(s) shall be retransmitted sequentially in order to preserve the transmission order. New I frames may be transmitted if they become available and this is made possible by returning to the original state after leaving this macro definition.
This macro definition takes care of the transmission of an FRMR frame by the secondary station if a frame rejection condition has occurred or is still unrecovered. This is indicated by the variables 'W', 'X', 'Y', and 'Z', local to the SR process. First, these variables are updated, and the control field of the rejected frame is 'stored' in the variable RFCF, by some (undefined) means. If more than one of the frame rejection conditions has occurred, w, x, y, and z shall be set to 0. Depending on the operational mode and the value of PF, the FRMR frame shall be sent, with or without the F bit set to 1. The FRMR frame shall contain an information field, which consists of the rejected frame control field (RFCF), the actual values of N(S) and N(R), a C/R bit set to 0 (indicating that a command frame is rejected), and the values of w, x, y, and z. */

Undefined macro which returns the rejected frame control field into the variable 'RFCF'

Check for more than one frame rejection condition

On the use of w, x, y, and z:
- w shall be set to 1 if a command or response was received that is undefined or not implemented;
- x set to 1 shall indicate the receipt of a frame containing an information field when no information field is permitted by the associated control field. Bit w shall be set to 1 in conjunction with this bit (as is shown);
- y set to 1 shall indicate that the information field received exceeded the maximum information field length which can be accommodated by the receiving station;
- z shall be set to 1 if the control field received and returned contained an invalid N(R).

The w, x, y, and z bits in the information field of an FRMR response may all be set to 0 (as shown), indicating an unspecified rejection of the frame for one or more of the conditions cited above. */

See IS 4335 for FRMR information field format
MACRODEFINITION Send_Balanced_FRMR

/* This macro definition takes care of the transmission of an FRMR frame by the combined station if a frame rejection condition has occurred or is still unrecovered. This is indicated by the variables 'W', 'X', 'Y', and 'Z', local to the SR process. First, these variables are updated, and the control field of the rejected frame is stored in the variable RFCF, by some (undefined) means. If more than one of the frame rejection conditions has occurred, W, X, Y, and Z shall be set to 0. Depending on the operational mode and the value of PF, the FRMR frame shall be sent, with or without the F bit set to 1. The FRMR frame shall contain an information field, which consists of the rejected frame control field (RFCF), the actual values of N(S) and n(R), a C/R bit set to 0 (indicating that a command frame is rejected), and the values of w, x, y, and z. */

Undefined macro which returns the rejected frame control field into the variable "RFCF"

Check for more than one frame rejection condition

/* On the use of w, x, y, and z:
- w shall be set to 1 if a command or response was received that is undefined or not implemented;
- x set to 1 shall indicate the receipt of a frame containing an information field when no information field is permitted by the associated control field. Bit w shall be set to 1 in conjunction with this bit (as is shown);
- y set to 1 shall indicate that the information field received exceeded the maximum information field length which can be accommodated by the receiving station;
- z shall be set to 1 if the control field received and returned contained an invalid N(R).

The w, x, y, and z bits in the information field of an FRMR response may all be set to 0 (as shown), indicating an unspecified rejection of the frame for one or more of the conditions cited above. */

See IS 4335 for FRMR information field format
MACRODEFINITION Higher_Priority_Events

Entry

view (discREQ,SR) (True) DISC_req

(view (discREC,SR) (False)

(view ( resetsREQ,SR) (True) RSET_rec

(view (sabmREC,SR) (False) SABM_rec

(view (X,SR) (0)

(view (X,SR) (1)

(view (Y,SR) (0)

(view (Z,SR) (0)

Exit ResetTimer_Tp

Frame Rejection Reported

/* See the MACRO 'Send_Balanced_FRMR' for procedures on the use of the parameters W, X, Y, and Z. */
MACRODEFINITION Initialize_Secondary_SR

```
Entry

snrmREC=False;
sarmREC=False;
discREC=False;
LocBusy=False;
RemBusy=False;
RNRSent=False;
Mode=UNC;
PF=0; VS=VR=
=VSU=VSP=0;
W=0; X=0;
Y=0; Z=0;

Exit
```

MACRODEFINITION Initialize_Combined_SR

```
Entry

discREQ=False;
discREC=False;
resetREQ=False;
sabmREC=False;
LocBusy=False;
RemBusy=False;
RNRSent=False;
TsExp=False;
TdExp=False;
TrExp=False;
TpExp=False;
Ns=Nd=Nr=0;
P=1; F=1; VS=
VR=VSU=VSP=0;
W=X=Y=Z=0;

Exit
```