Formulas, processes, and Petri-Nets applied to the specification and verification of a HDLC protocol

M.Duque Antón, M.Bretschneider*

Philips GmbH Forschungslaboratorium Hamburg
Vogt-Koelln-Str.30 , D-2000 Hamburg 54

Abstract

In specifying a variant of a HDLC protocol we illustrate a method that addresses three problems: decomposition of a complex system into simple components, hierarchical design of a protocol starting from the service specification and resulting in the protocol description and, finally, modular verification of the protocol w.r.t. the service specification. The theoretical basis is given by Process Theory. Descriptions of basic protocol functions are obtained considering projections of the global protocol behaviour onto 'locally' relevant sets of events. Compositionality of semantics allows reasoning about complete (protocol-)components by induction over properties of their constituents. To formalize behaviour we use formulas, process terms and Petri-Nets.

Introduction

The logical structure of communication protocols conceived today is very involved. This applies in particular to protocols associated with the Open System Interconnection reference model, OSI (c.f. [ISO]). Their descriptions are made up of lists of many different functions displaying an intricate proportion of interaction and parallelism. This calls for elaborated construction methods.

There are currently many research efforts being made to overcome these problems. They are related to the development of techniques for design, specification and verification. This paper focusses on a design technique that is based on two basic ideas: progressing from one design step to the next one utilizes the concept of an external and an internal view of an entity. One first depicts this entity as seen externally, afterwards detailing the internal structure. To check the consistency of both views the communication between inner constituents are concealed and the resulting behaviour is proven to be as specified by the external view. The approach comprises the concept of layers in the OSI reference model.

In order to characterize individual components one first picks out their different functions and to assist construction one specifies them separately. This supports the understanding of intuitively clear building blocks. Existing components can be used instead of designing them from scratch. In this respect a top-down refinement and a bottom-up strategy are combined.

The design technique is related to ideas by [LS]. There projections are considered which, however, are used for analysis of protocols rather than construction. A comparable case study employing the specification language Z was presented by [DHKR]. The formal description is based on work

* This research was partially sponsored by RACE project nr. R 1046, SPECS

performed by [Ol1], [BK1], and [Mi]. For specification the trace-logic as defined in [Ol2] and [Zw] is used. The gradual transformation of specifications into processes is due to [Ol2]. Non-interleaving semantics for process terms were defined by [DDM] and [Ol3].

This paper is organized as follows. Section 1 explains the theoretical framework, Sections 2 and 3 introduce the protocol to be considered and its specification. Section 4 – 6 detail the stepwise design of the protocol. Finally, section 7 gives the verification.

1 Theoretical Foundations

To specify the behaviour of a system we first decide which kind of events (communications) are of interest. This defines the alphabet of the system. Next we consider sequences of occurences of these events yielding traces (or histories). Allowed traces are characterized by means of formulas of a many sorted predicate logic. Below we give a syntax which is a simplified version of a proposal by [Zw].

1.1 Syntax

Let h be a distinguished trace variable. Let t range over a set of trace variables not including h, v over a set of non-negative integer variables, and n range over the integer constants. Let a, b range over the set *Comm* of communications and A, B over communication alphabets, i.e. subsets of *Comm*. We introduce the following classes of expressions:

Trace expressions with values in $Comm^*$:

$$te ::= \epsilon \mid a \mid h \mid t \mid te_1 \cdot te_2 \mid te\lceil A \mid te[b/a] \mid$$

Integer expressions taking values in N_0:

$$ie ::= n \mid v \mid \; |te| \; \mid ie_1 + ie_2 \mid ie_1 * ie_2$$

Communication expressions yielding values in *Comm*:

$$ce ::= a \mid te[ie]$$

Boolean expressions yielding values in $\{true, false\}$:

$$be ::= true \mid te_1 \preceq te_2 \mid ie_1 \leq ie_2 \mid ce_1 = ce_2 \mid \neg be \mid be_1 \wedge be_2 \mid \exists t.be \mid \exists v.be$$

We use the subsequent notation: $te\lceil A$ is the projection of te onto A, i.e. te with all communications outside A removed, $te_1 \cdot te_2$ is the concatenation of te_1 and te_2. In te[b/a] all occurences of a in te are replaced by b. $|te|$ is the length of te, $te[ie]$ is the ie-th element in the trace te, if this exists. $te_1 \preceq te_2$ states that te_1 is a prefix of te_2. Notice, that some of the functions are only partially defined.

We will employ the usual logical abbreviations $be_1 \vee be_2$ or $te_1 = te_2$ etc. plus the following ones, the first one denoting the 'counter' of a in the trace te:

$$\begin{array}{lcl} a\#\, te & := & |te\lceil\{a\}| \\ \alpha.h \subseteq A_S & := & h = h\lceil A_S \\ \mathsf{first}\ (te) & := & te[1] \\ \mathsf{last}\ (te) & := & te[|te|] \\ \mathsf{last}_2\ (te) & := & te[|te| - 1] \end{array} \qquad (1)$$

Furthermore, head (te) [1] stands for a trace expression denoting all communications of te but the last. That is:

$$te = \mathsf{head}\ (te) \cdot \mathsf{last}\ (te)$$

We also make use of the following abbreviation: A boolean expression like

$$\mathsf{last}\ (h) = b \wedge P(h) \Rightarrow \mathsf{next}\ (h) = a$$

where P is a predicate stands for

$$\mathsf{last}\ (\mathsf{head}\ (h)) = b \quad \wedge \quad P(\mathsf{head}\ (h)) \quad \Rightarrow \quad \mathsf{last}\ (h) = a$$

In the case of structured communications (ch, m_i), $h\lceil ch$ denotes the sequence $(ch, m_1) \ldots (ch, m_n)$. For details see [Ol2]. In case there are no ambiguities, we will use $h\lceil ch$ as an abbreviation for the sequence of messages $m_1 \ldots m_n$ sent along the channel ch in h.

We now give the formal definition of a specification and its semantics, cf. [Ol1].

1.2 Specifications

Definition 1.1 (Specification) *A specification S is a Boolean expression with at most h as a free variable. A specification S is in normal form, if it is of the form $S \equiv \langle \alpha.h \subseteq A_S \wedge H_S \rangle$ such that h occurs in the history predicate H_S only in the form $h\lceil B$ where $B \subseteq A_S$. The readiness-semantics $[[S]]$ is defined by*

$$[[S]] \; = \; \{h \in A_S^* \mid H_S(h)\} \; \cup \; \{(h, X) \in A_S^* \times \mathcal{P}(A_S) \mid H_S(h) \text{ and } (X = \{a \in A_S \mid H_S(ha)\})\}$$

Thus the ready set X consists of all communications $a \in A$, such that the extended trace ha satisfies H_S. Hence $[[S]]$ is uniquely determined by its traces. To illustrate the definition we provide a simple example: Let be $A_S = \{a, b\}$ and $H_S(h) \equiv (\forall\, h' \preceq h : 0 \leq a\#h' - b\#h' \leq 1 \ \wedge \ b\#h' \leq 1)$. Then $[[S]]$ is given by

$$\{\epsilon, a, ab, aba\} \; \cup \; \{(\epsilon, a), (a, \{b\}), (ab, \{a\}), (aba, \emptyset\}$$

The first set describes all possible histories of S whereas the second defines the traces together with the corresponding set of next possible events. Subsequently, a short-hand notation for history-predicates is employed, yielding $0 \leq a\#h - b\#h \leq 1$ instead of $\forall\, h' \preceq h : 0 \leq a\#h' - b\#h' \leq 1$.

1.3 Transformation into processes

Although the behaviour of a system S in this model is completely explained by a logical formula, one is interested - for implementation purposes - to provide a more constructive view of the system. Following the ideas of [Ol1] this can be done by a gradual transformation of a logical formula into a process term which includes operators like prefixing ("$\rightarrow$"), choice ("$+$") or parallel composition ("$\|$").

The semantic domain for these process (terms) equals the domain of logical formulae, i.e. is given by the readiness-semantics (cf. Def.1.1). Then the fact that a process P exhibits a behaviour $[[P]]$

[1] head(te) is not expressible as a trace expression; for any boolean expression be, however, there exists an equivalent boolean expression $\hat{be}$ not containing head(te), for details see [Zw].

as described by S (safety) and that P allows the execution of all events as specified by S (liveness), is defined in a set-theoretic-approach:

$$P =_{\mathcal{R}} S \quad \Longleftrightarrow_{Def} \quad [[P]] = [[S]]$$

Likewise we define $P =_{R} Q$ for processes P and Q.

The way how the transformation of a specification S (consisting of formulae) into a readiness-semantically-equivalent process P_S (containing prefixing and choice) can be perfomed, is described by the Initial Communication Theorem.

Theorem 1.2 (Initial Communication) *Consider $S \in$ Spec with $(\varepsilon, \{a_1, a_2, ..., a_n\}) \in [|S|]$. Then*

$$((a_1 \rightarrow S[a_1 h/h]) + ... + (a_n \rightarrow S[a_n h/h])) \quad =_{\mathcal{R}} \quad S$$

holds, where $S[a_i h/h]$ results from S by substituting the expression $a_i.h$ for h.

This theorem describes how to expand specifications into processes using a next-possible-event-strategy.

In [Ol1], the conclusion of the theorem only guarantees semantic inclusion $\subseteq_{\mathcal{R}}$ instead of semantic equality $=_{\mathcal{R}}$. For prefix-closed specifications S in normal form, which are the only ones we will encounter here, inclusion can be replaced by equality.

Apart from readiness-semantic it is possible to endow process terms with an operational semantic based on Plotkin-rules. In this case a process term denotes a (non-)deterministic (in-)finite state-machine from which the readiness-semantic can be retrieved (see [Ol1]). Equality on non-deterministic machines can then be defined in terms of various equivalences. Here we use the so-called observational equivalence ([BK1] and [Mi]) and put $P =_{\mathcal{O}} Q$ for processes terms P and Q iff observational equivalence holds between the machines denoted by P and Q. The relation between $=_{\mathcal{O}}$ and the above $=_{\mathcal{R}}$ is given by the following fact: if $P =_{\mathcal{O}} Q$ and P, Q contain no cylces of τ-actions, then $P =_{\mathcal{R}} Q$ follows (see [Ol4]).

1.4 Compositionality

The semantic function $[[\cdot]]$ can be shown to be compositional w.r.t. the structure of processes. To prove $P_S =_{\mathcal{R}} S$ one can use structural induction on the subcomponents of P_S and S. This property justifies the proposed 'divide-and-conquer' approach as it allows us to deduce properties of a process (program) from properties of its constituents. Vice versa, the transformation into processes can be performed componentwise. This facilitates the combination of top-down and bottom-up design. Below we quote several related theorems from [Ol1] which are frequently used in the sequel.

Theorem 1.3 (Decomposition) *Let S, T, U be specifications in normal form with*

$$A_S = A_T \cup A_U \quad \text{and} \quad H_S = H_T \wedge H_U$$

Then $T \, \|_{A_T \cap A_U} \, U =_{\mathcal{R}} S$ *holds.*

The theorem exhibits the degree of parallelism that is inherent to the specification S. Notice that in the following we simply drop the index $A_T \cap A_U$. With this parallel composition $\|$ is associative.

Theorem 1.4 (Hiding) *Let S, T be specifications in normal form and let B be a set of communications such that*

1. $A_S \; = \; A_T \setminus B$
2. $H_T \Rightarrow H_S$ *(safety)*
3. $\neg \, \exists \, h \in A_T^* \, \forall n \in N \; \exists h_n \in B^* : \; H_T[h \cdot h_n / h]$ *(no divergence)*
4. $(h, X) \in [[T]] \; \wedge \; X \cap B = \phi$ *implies* $(h \setminus B, X) \in [[S]]$ *(same liveness)*

Then $T \setminus B =_{\mathcal{R}} S$.

We remark that $\exists h_n \in B^*$ is an abbreviation for $\exists h \in B^*$: $|h| = n$. Recall, that '\' denotes concealment of communications.

1.5 Petri-Net-Semantics

Apart from readiness-semantic it is possible to furnish the language of process terms with an operational non-interleaving semantics [Ol3]. One utilizes the idea of decomposing a process into a set of 'sequential components' which denote local states of a Petri-Net. The resulting Petri-Net gives a distributed view of the process and the specification (in the sense of readiness-semantics $[[\cdot]]$). We use this representation to check whether the formalized requirements meet our intuitive expectations. By this we mean in particular, that the logical specification contains *all* necessary requirements we have in mind. Regarding implementations this view can be applied to decide what degree of parallelism is usefull. For hardware configuration,however, we may have to dissolve some parallelism.

2 Case Study

Our example considers the link layer (c.f. [ISO]). We construct a sliding window protocol resembling HDLC (High-level-data-link-control) using normal response mode. The HDLC protocol is designed to provide reliable full-duplex data transfer between network entities operating on the unreliable physical layer. The physical layer is supposed to leak but not to reorder data.

For the sake of clarity we will consider an idealized version of the data procedures in HDLC:

- Data transfer is assumed to be in one direction only
- Data are regarded as transparent (as we are only interested in the protocol control part)

Moreover, we do not describe the connection management.

The protocol is expected to work on an idealized physical layer channel, consisting of two channels, Data-Chan and Ack-Chan. Data-Chan conveys only data from Sender-Link-Entity to Receiver-Link-Entitiy. Ack-Chan returns acknowledgements. (See figure 1)

3 External View of Link Layer

In this section the service specification, i.e. the required behaviour of the protocol as seen by the network layer is presented. This behaviour should equal a buffer of a specific capacity *b*, supporting data exchange from Sender-Network-Entity to Receiver-Network-Entity. The data exchange should be flow-controlled, i.e. both the buffer and the Receiver-Network-Entity should be protected from

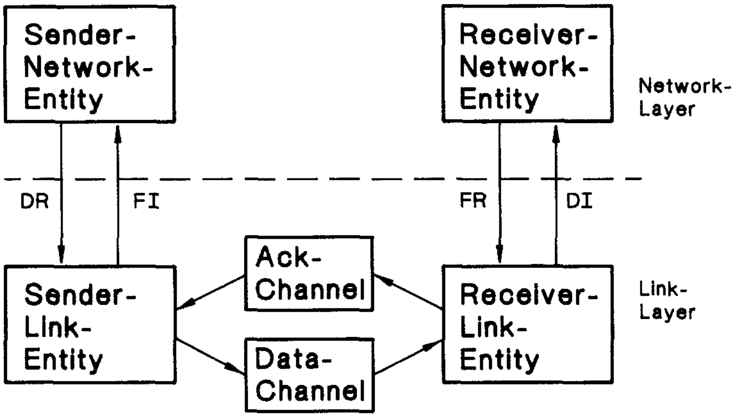

Figure 1: Protocol Architecture

overload. As the specified behaviour will turn out to describe a reusable entity, it is called Back Pressure Unit or BPU for short.

The alphabet of communications BPU may engage in is given by

$$A_{BPU} := \{\ FI^-,\ DR^+,\ FR^+,\ DI^-\ \}$$

Here, we made use of the following abbreviations:

D	: *Data*	R	: *Request*
F	: *Flow*	I	: *Indication* [2]

In this example flow-control is modelled explicitly, whereas it is usually treated implicitly as a part of a frame (e.g. Receive-(non)-Ready, for details see [Ta]). A flow-information F indicates that the receiver has buffer-space available.

The BPU is completely determined by the buffer status (number of waiting_data), the account of credits given to the user sender but not yet used (credit_sender), the sum of both (allocated_buffer), and the account of credits received from the user receiver but not yet used (credit_receiver). Formally, this is expressed as follows (recall, that h is a variable of type trace):

$$\begin{aligned} \mathit{waiting\_data}(h) &:= DR^+\#h - DI^-\#h \\ \mathit{credit\_sender}(h) &:= FI^-\#h - DR^+\#h \\ \mathit{allocated\_buffer}(h) &:= \mathit{waiting\_data}(h) + \mathit{credit\_sender}(h) = FI^-\#h - DI^-\#h \\ \mathit{credit\_receiver}(h) &:= FR^+\#h - DI^-\#h \end{aligned}$$

The parameters specifying the BPU are the buffer capacity b, the credit window size at the sender side c_S and at the receiver side c_R. Here, b limits the allocated_buffer as well as the waiting_data. c_S, c_R limits the credit_sender, c_R limits the the credit_receiver. This leads to the following specification:

$$\begin{aligned} &BPU(b, c_S, c_R)\ [FI^-, DR^+, FR^+, DI^-] := \\ &\langle\ \alpha.h \subseteq A_{BPU} = \{FI^-, DR^+, FR^+, DI^-\} \\ &\wedge\ \ 0 \leq \mathit{allocated\_buffer(h)} \leq b \ \ \wedge\ \ 0 \leq \mathit{waiting\_data(h)} \leq b \\ &\wedge\ \ 0 \leq \mathit{credit\_sender(h)} \leq c_S \ \ \wedge\ \ 0 \leq \mathit{credit\_receiver(h)} \leq c_R\ \rangle, \end{aligned}$$

[2]To assist intuition a direction is attached to events corresponding to sending ("-") and receiving ("+") of data.

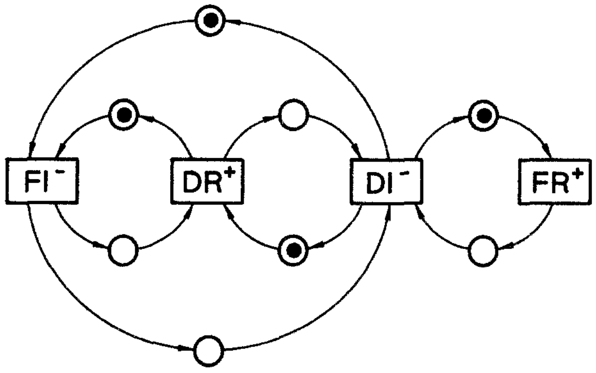

Figure 2: Petri Net corresponding to BPU (1,1,1) with $b = c_S = c_R = 1$

where c_S is less than or equal b. A non-interleaving operational view of BPU is given by the Petri-Net in figure 2. (See also section 1.5). Notice that for instance in the initial state the communications FI^- and FR^+ are allowed to occur concurrently.

4 Stepwise Protocol Design

The protocol is constructed hierarchically in two steps according to the two layers in figure 3. First, the service-specification BPU is divided up into the following components in order to achieve a distributed system: two buffer-managers s-Buff-Mgr, r-Buff-Mgr containing the whole buffer capacity, and a reliable channel (without buffering) connecting both. Next, the reliable channel Rel-Chan is replaced by a pair of unreliable channels Data-Chan and Ack-Chan representing the physical layer, and new protocol components: sender-protocol (s-Prot) and receiver-protocol (r-Prot) dealing with possible losses.

We are ultimately interested in constructing the two entities Sender-Link-Entity and Receiver-Link-Entity that operate on the unreliable channels (see figure 1). In order to achieve these, the components X-Prot resp. X-Buff-Mgr will be utilized (X = s,r). We will consider their parallel composition and conceal the internal communications between them as they are not relevant to the external behaviour description of the Link-Entity-components.

Formally, the X-Link-Entity is defined by

$$\text{X-Link-Entity} := (\text{X-Buff-Mgr} \parallel \text{X-Prot}) \setminus I_X \text{ for } X = s, r$$

where $I_X = \{XbF, XbD\}$ are the sets of the above internal communications. The intuition behind them will be explained in section 6.

We bring to mind that one reason for stepwise protocol design is to ease verification, i.e. to show that the protocol satisfies the specification. Formally, verification amounts to proving:

$$\begin{aligned}&(\text{Sender-Link-Entity} \parallel \text{Ack-Chan} \parallel \text{Data-Chan} \parallel \text{Receiver-Link-Entity}) \setminus I_{Prot}\\ &=_{\mathcal{R}} BPU\end{aligned} \tag{2}$$

where $I_{Prot} = \{(A, -), (D, -), TO, TO\text{-}P, P\}$ is the set of internal protocol communications (see section 6). According to the two above design steps and the definition of X-Link-Entityabove this proof will be split into two parts:

$$(\text{s-Prot} \parallel \text{Ack-Chan} \parallel \text{Data-Chan} \parallel \text{r-Prot}) \setminus I_{Prot} =_{\mathcal{R}} \text{Rel-Chan} \tag{3}$$

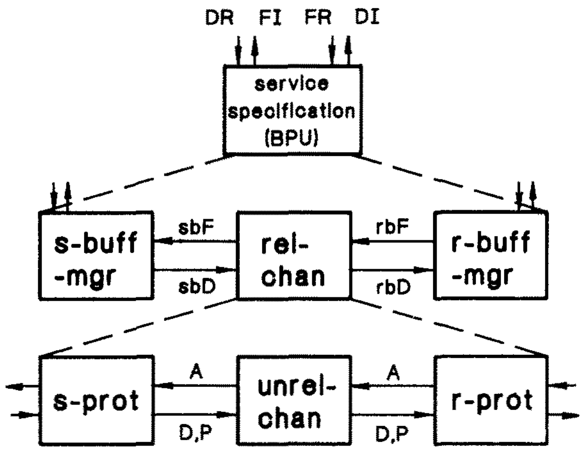

Figure 3: Design Strategy

and

$$(\text{s-Buff-Mgr} \parallel \text{Rel-Chan} \parallel \text{r-Buff-Mgr}) \setminus (I_S \cup I_R) \;=_{\mathcal{R}}\; BPU \tag{4}$$

The proof utilizes the fact that one can interchange the order in which parallel composition of processes '$\|$' and concealment of events '$\setminus$' are applied. The conditions which guarantuee correctness of this proceeding are subject of the so-called Modular Verification Theorem, proven in [BDF].

Notice that equation 4 in particular involves the fact that no additional buffer capcity is created when subdividing *BPU*. (For the proofs see section 7).

5 First Design Step: Introducing Distribution

As mentioned in the previous section, the main feature that is added is the distribution of buffer capacity. It turns out that the component BPU can be used (with some slight modifications) to specify the local buffer managers. The reliable channel Rel-Chan basically represents the window to be realized later, preserving the buffer capacity. For details see [BDF].

6 Second Design Step: Sliding Window

In a second step we consider the underlying unreliable channels. For that reason we add new protocol components: s-Prot and r-Prot. For these protocol components we recognize the following basic functions:

1. Error detection and recovery.
2. Intercommunication with the matching local buffer manager.
3. Indication rsp. inquiry of the buffer status of the receiver link entity.

Item 2 is required as the protocol component uses the buffer of the local buffer manager, item 3 in order to realize flow control. Figure 4 illustrates the structure of the protocol component s-Prot: Here, 'A' means acknowledgement, 'D' stands for data. When s-Prot executes 'sbD^+', control over a buffer place containing one datum is passed from s-Buff-Mgr to s-Prot. Vice versa 'sbF^-'

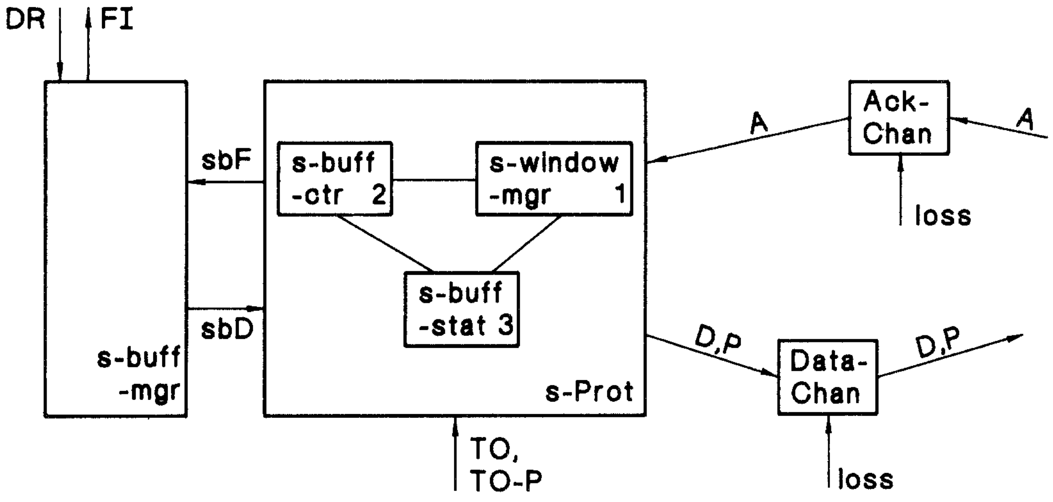

Figure 4: Structure of s-Prot. The numbers in the boxes refer to the basic functions.

stands for returning control of an empty buffer place. 'P' symbolizes the poll mechanism which is needed in order to realize flow control. We recognize that s-Prot may engage in the following communications: sbD^+, sbF^-, D^-, P^- and A^+. Communications A and D are structured:

$$\begin{aligned} A &\longrightarrow A \times k \times Status, \\ D &\longrightarrow D \times k\ , \end{aligned}$$

where k ranges over the sequence numbers $1, 2, \ldots, w+1$ (w = window size) and '*Status*' attains the values "ready" or "non-ready".

6.1 Window Management

The window management components s-Window-Mgr and r-Window-Mgr deal with the main task of the protocol, the detection of and recovery from errors due to unreliable channels. To improve efficiency of the protocol we have chosen a sliding window version (c.f. [Ta]). The sending unit is allowed to send up to w data in sequence without any intermediate acknowledgement. Therefore, any incoming data unit is provided with a sequence number ranging in cyclic order between 1 and $w+1$. This allows the identification of retransmissions by the receiver. Equally, the receiver attaches a sequence number to the acknowledgements indicating the number of the last correct received datum.

s-Window-Manager

1. The basic idea is to maintain an intervall of cyclic sequence numbers. This intervall, the so-called *window* represents data that have been sent but not yet been confirmed by the receiver. An important factor of the specification is the number

$$\text{outstanding\_data}\ (h) := \text{sent\_data}\ (h) - \text{conf\_data}\ (h)$$

which is defined by the number of correct sent data and the number of correct confirmed data. These values will be formally defined below. From these numbers we obtain the actual sequence numbers of the last correct sent datum and -confirmed datum by

$$\begin{aligned} \text{last\_sent datum}\ (h) &:= \text{sent data}\ (h) \quad (\text{mod } w+1) \\ \text{last\_conf\_datum}\ (h) &:= \text{conf\_data}\ (h) \quad (\text{mod } w+1) \end{aligned}$$

Notice, that $\mathsf{last\_conf\_datum}\ (h) \oplus 1$ represents the number of the *first* outstanding datum - provided there are unconfirmed data. Recall that $\oplus$ denotes $+ \pmod{(w+1)}$. Likewise $\ominus$ denotes subtraction with range $0, 1, \ldots, w$.

2. We shall now look more closely at the definition of $\mathsf{sent\_data}\ (h)$ and $\mathsf{conf\_data}\ (h)$: If we abstract from losses and retransmission of data, a typical trace h of send events would be

$$D_1^- \ D_2^- \ldots D_{w+1}^- \ D_1^- \ D_2^- \tag{5}$$

In this case [3] the number of correct sent data simply is the number of D_x^- in h (x arbitrary), i.e. the counter $D^- \# \ h$, cf. section 1. The treatment of retransmission, however, may lead to more complicated traces like

$$D_1^- \ \underline{D_1^-} \ D_2^- \ \underline{D_1^-} \ \underline{D_2^-} \ D_3^- \ \underline{D_3^-} \ \underline{D_3^-} \tag{6}$$

where retransmissions are underlined for illustration as they intuitively do not change the number of correct sent data. Below we list this number for all prefixes of the trace 6 yielding

$$0\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 3 \tag{7}$$

the initial zero referring to the empty trace. For an arbitrary trace the number of correct sent data can be computed by the following rule:

$$\begin{aligned} &\mathsf{sent\_data}\ (\epsilon) = 0 \\ &\mathsf{sent\_data}\ (h) = k \ \Rightarrow \ \mathsf{sent\_data}\ (ha) = \begin{cases} k+1 & \text{if } a = D_{k\oplus 1}^- \\ k & \text{else} \end{cases} \end{aligned} \tag{8}$$

3. We now discuss the receipt of acknowledgements. Omitting multiple confirmations a typical trace of acknowledgements would be (for, say $w \geq 5$)

$$A_1^+ \ A_2^+ \ A_5^+ \ A_6^+ \tag{9}$$

In this situation the receiver confirmed the frames D_3^-, D_4^- and D_5^- with *one* acknowledgement A_5^+. The increment $2 \rightarrow 5$ of sequence numbers indicates that all frames with numbers 3, 4, and 5 have been correctly received. This fact is captured by the following definition of $\mathsf{conf\_data}\ (h)$:

$$\begin{aligned} &\mathsf{conf\_data}\ (\epsilon) = 0 \\ &\mathsf{conf\_data}\ (h) = k \ \Rightarrow \ \mathsf{conf\_data}\ (ha) = \begin{cases} k + (l \ominus k) & \text{if } a = A_l^+ \\ k & \text{else} \end{cases} \end{aligned} \tag{10}$$

The last definition takes into consideration the fact that at most the last correct received acknowledgement may be received several times (iff $l \ominus k = 0$), as the channels preserve the order of data: As an example we have for $w = 2$

$$\mathsf{conf\_data}\ (A_1^+ \ A_2^+ \ A_2^+ \ A_1^+) = 4$$

Notice that a retransmission of, say A_2^+, follows immediately the first confirmation A_2^+.

4. For the specification of the window-management it will be important to know whether the last occurence of a D^- event refers to the *first* transmission of a datum. This fact can be decided by looking at the increment of the function $\mathsf{sent\_data}\ (h)$. The increment $\Delta\mathsf{sent\_data}\ (h)$ is defined to be zero for empty trace and

$$\Delta\mathsf{sent\_data}\ (ha) := \mathsf{sent\_data}\ (ha) - \mathsf{sent\_data}\ (h)$$

[3] D_i^- stands for (D^-, i)

We define the last occurence of D_k^- in a trace h to be a 'fresh' one , if the increment is positive:

$$\mathsf{fresh}_{D^-}(h) \;:=\; \Delta\mathsf{sent\_data}\;(h\lceil D^-) > 0$$

E.g. for

$$h = h_1\; D_1^- \;\text{ where }\; h_1 = D_1^-\; D_2^-\; D_1^-\; D_2^-\; A_2^+\; D_3^-\; A_3^+\; D_1^-$$

we have (assuming $w\;=\;2$) $\mathsf{sent\_data}\;(h\lceil D^-) = 4$ and $\mathsf{sent\_data}\;(h_1\lceil D^-) = 3$, hence $\mathsf{fresh}_{D^-}(h)$. Likewise, we define $\mathsf{fresh}_{A^+}(h)$.

5. We are now prepared to specify the functions of s-Window-Mgr: the sender is allowed to send up to w data in advance without any intermediate confirmation. Trivially it may only expect confirmation for data that have previously been sent. Hence we require

$$0 \le \mathsf{outstanding\_data}\;(h) \le w \tag{11}$$

When the sender forwards a new datum it increments the respective sequence number by 1 (mod $w+1$). Say, the number of the last fresh datum is k, then the number of the previous correct sent datum must be $k \ominus 1$. Furthermore, the very first frame number is 1. These facts are expressed by the formulas

$$\begin{gathered}\mathsf{last}\;(h\lceil D^-) = k \;\wedge\; \mathsf{fresh}_{D^-}(h) \;\Rightarrow\; \mathsf{last\_sent\_datum}\;(\mathsf{head}\;(h\lceil D^-)) \;=\; k \ominus 1, \\ \mathsf{first}\;(h\lceil D^-) = 1\end{gathered} \tag{12}$$

6. The following three predicates 13, 14, 15 describe the *conversation modus* between s-Window-Mgr and r-Window-Mgr. Here, the master-slave modus is employed, the sender being the master. A timer is used to initiate the retransmission of all outstanding data after some period of time.

$$\mathsf{last}\;(h) = TO \;\Rightarrow\; (\mathsf{outstanding\_data}\;(h) > 0 \;\wedge\; \mathsf{last}_2\;(h) \;\neq\; TO) \tag{13}$$

This gives a necessary condition for a timeout. When a timeout has occured we require that the sender retransmit all outstanding data in one go: by this we simply mean that if the last event was a retransmission with number k, then the last but one event is the retransmission $k \ominus 1$ etc. In case that k represents the first outstanding datum (= $\mathsf{last\_conf\_datum}\;(h) \oplus 1$), the last but one event is the triggering timeout. More precisely,

$$\begin{gathered}\mathsf{last}\;(h) = D_k^- \;\wedge\; \neg\mathsf{fresh}_{D^-}(h) \;\wedge\; \mathsf{last\;conf\_datum}\;(h)\;\oplus 1 \;\neq\; k \\ \Rightarrow\; \mathsf{last}_2\;(h) = D_{k\ominus 1}^-\end{gathered} \tag{14}$$

and

$$\begin{gathered}\mathsf{last}\;(h) = D_k^- \;\wedge\; \neg\mathsf{fresh}_{D^-}(h) \;\wedge\; \mathsf{last\_conf\_datum}\;(h)\;\oplus 1 \;=\; k \\ \Rightarrow\; \mathsf{last}_2\;(h) = TO\end{gathered} \tag{15}$$

7. Summarizing, we now define the specification of s-Window-Mgr to be

$$\mathsf{s\text{-}Window\text{-}Mgr}(w)\;[D^-, A^+, TO] \equiv_{Df} \langle \alpha.h \;\subseteq\; \{D^-, A^+, TO\} \;\wedge\; H_{SWM}(h)\rangle$$

where $H_{SWM}(h)$ is the logical conjunction of requirements 11, ..., 15. By the Initial Communication Theorem 1.2 we can derive a process P for this specification. For window-size $w = 1$ it is depicted in figure 5.

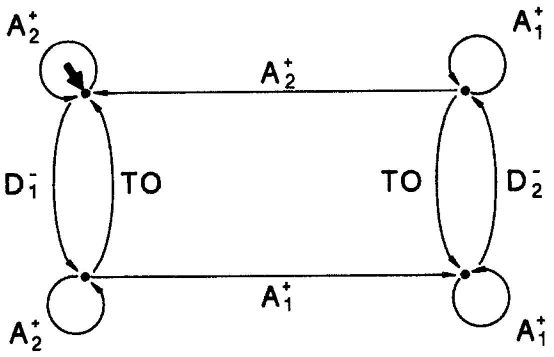

Figure 5: Finite state machine representation for s-Window-Mgr for $w = 1$

r-Window-Manager

The essential concept again is to control the window. This time the window is given by the number of correct received data (rec_data (h)) and the number of correct acknowledged data (ack_data (h)). We define

$$\text{unconfirmed\_receipts } (h) := \text{rec\_data } (h) - \text{ack\_data } (h) \tag{16}$$

and require

$$0 \leq \text{unconfirmed\_receipts } (h) \leq w \tag{17}$$

The corresponding frame numbers are given by the values modulo $w + 1$:

$$\begin{aligned} \text{last\_rec\_datum } (h) &= \text{rec\_data } (h) \pmod{w+1} \\ \text{last\_rec\_datum } (h) &= \text{rec\_data } (h) \pmod{w+1} \end{aligned} \tag{18}$$

As conversation modus is the master-slave principle, every acknowledgement has to confirm the *last* correct received datum:

$$\text{last\_ack\_datum } (h) = k \;\Rightarrow\; \text{last\_rec\_datum } (h) = k \tag{19}$$

Furthermore, we demand that the receiver confirms the receipt as soon as the number of unconfirmed data is maximal ($= w$).

$$\begin{aligned} \text{last } (h) = D^+ \;\wedge\; & \text{unconfirmed\_receipts } (h) = w \\ \Rightarrow\; & \text{next } (h) = A^- \end{aligned} \tag{20}$$

Data which are received out of sequence will simply be discarded. The formal definition of the counter functions rec_data (h) and ack_data (h) in equality 16 correspond to sent_data (h) and conf_data (h), respectively of the s-Window-Mgr definition, see definitions 8, 10 (with the substituition $D^- \rightarrow D^+$ and $A^+ \rightarrow A^-$). The complete specification of r-Window-Mgr is now given by

$$\text{r-Window-Mgr}(w)[D^+, A^-] \equiv_{Df} \langle \alpha.h \subseteq \{D^+, A^-\} \wedge H_{RWM}(h) \rangle$$

where H_{RWM} is the logical conjunction of 17,...,20.

6.2 Buffer-Control

The X-Buff-Ctr, $X = s, r$ processes organize communication between local window manager and local buffer. The following section discusses s-Buff-Ctr. The r-Buff-Ctr works in a similar way. It is not described here.

s-Buffer-Control

The communications s-Buff-Ctr may engage in are sbD^+, sbF^-, D^- and A^+. For the intuition behind sbD^+ and sbF^- recall the first paragraph of section 6, see also figure 4. The s-Buff-Ctr component closely cooperates with the s-Buff-Mgr which stores data that are to be transmitted. s-Buff-Mgr will allocate a number of data, denoted by k = granted (h) to the s-Buff-Ctr. This number may not exceed the window size:

$$0 \leq \text{granted } (h) \leq w \tag{21}$$

The granted data are ready for transmission, m of them have not been involved yet (= waiting (h)). Trivially,

$$0 \leq \text{waiting } (h) \tag{22}$$

Furthermore, there are n data among those k, which have been succesfully passed on but still occupy buffer capacity, i.e. the 'ack' has not been indicated yet. This number, conf_not_ind (h) should be limited to one to optimize performance:

$$0 \leq \text{conf not\_ind } (h) \ \leq 1 \tag{23}$$

In addition, we expect the s-Buff-Ctr to release buffer space (sbF^-) when the first acknowledgement has been received:

$$\text{last } (h) = A^+ \ \wedge \ \text{fresh}_{A^+}(h) \ \Rightarrow \ \text{next } (h) = sbF^- \tag{24}$$

The state of these entities is determined by

$$\begin{array}{ll} \text{granted } (h) & := sbD^+\# \ h - sbF^-\# \ h \\ \text{waiting } (h) & := sbD^+\# \ h - \text{sent\_data } (h) \\ \text{conf\_not\_ind } (h) & := \text{conf\_data } (h) \ - sbF^-\# \ h \end{array}$$

6.3 Monitoring the receiver status

The receiver status indicates whether there is sufficient buffer space available to the receiver to store data coming from sender. The receiver r-Prot has to indicate any shortage to the sender by means of a 'non-ready' (NR) information. According to this the sender has to organize the transmission of data. We therefore have to introduce two new components r-Buff-Stat and s-Buff-Stat (see figure 4). We specify only s-Buff-Stat.

s-Buffer-Status

1. To reflect the status-information we have to distinguish the (number of) correct confirmed data conf_data (h) from the (number of) data which are correct confirmed *and* additionally indicate the status 'ready'. The formal definition of conf_data_ready (h) equals that of conf_data (h) (see 10) except that its value is only incremented if the acknowledgement A_l^+ carries a ready-information.

A 'ready' indicates that the receiver is prepared to accept up to w data. Hence we require

$$\text{sent\_data } (h) \ \leq \ \text{conf\_data\_ready } (h) \ + w \tag{25}$$

I.e. sent_data (h) may exceed conf_data_ready (h) by at most w data. Notice an acknowledgement A^+, carrying a 'non-ready', only indicates a correct receipt.

2. Since s-Prot acts as the master it has to inquire the buffer status from the receiver in case that it is not allowed to send a new datum. This 'poll mechanism' which is timer-triggered is specified by

$$\text{last } (h) \ = TO\_P \ \ \Rightarrow \ \ \text{sent\_data } (h) \ = \ \text{conf\_data\_ready } (h) \ + w \tag{26}$$

$$0 \leq TO\_P\#h - P^{-}\#h \leq 1 \tag{27}$$

$$\text{last}\,(h) \;= TO\_P \;\Rightarrow\; \text{next}\,(h) \;= P^{-} \tag{28}$$

where TO_P denotes a timer, and P^{-} is a 'send poll' event. Notice, that we demand a 'send poll' to occur immediately after a time-out (cf. 28). Finally, the alphabet for this component is $A_{sBS} = \{D^{-}, A^{+}, P^{-}, TO\_P\}$.

7 Verification

We claim that the protocol, as described in sections 4, 5, 6, is semantically identical with the service specification (section 3) . This is described by means of equality 2. Informally, the statement says that the protocol components manage with losses in the channels [4] . Of course, this can be true only if the channels do not loose data infinitely often in sequence. Hence our proof will be based on the following *fairness property*

$$(FP) \quad \neg\, \exists h \in A^{*}_{Chan_x} \;\; \forall n \in N \;\exists h_n \in \{\text{loss}, x^{-}\}^{*} : \; H_{Chan}[h \cdot h_n / h] \tag{29}$$

where H_{Chan} is the predicate describing either of the channels and $A_{Chan_x} \;=\; \{\text{loss}, x^{-}, x^{+}\}$ with $x \in \{D, A\}$.

The proof of the claim is divided up into two parts along the lines of the protocol design (see figure 3): We first establish equaltity 4. The proof can be performed on a logical level by means of the Hiding Theorem 1.4, it is given in [BDF]. We notice that the proof holds for arbitrary parameter values b, c_S, c_R.

To establish equality 3 we make use of the fact that equality based on observational equivalence implies equality based on readiness equivalence (if there are no cycles of τ-actions), cf. section 1.3. In order to show observational equivalence, all relevant specifications of subcomponents are transformed into processes, and their parallel composition is computed. Then equality 3 w.r.t. observational equivalence is proved by means of a CCS-Tool ([Fr]). Due to the large number of states we restricted ourselves to window-size $w = 1$. For details see [DB]. The reason for showing observational euqivalence rather than readiness equivalence is: it is by far to tedious to ensure the 'same-liveness-condition' of the Hiding-Theorem 1.4.

8 Conclusion

The protocol was developed in a top-down manner: we started with the service specification describing the protocol seen as a single unit from a network-layers view. Next, we considered the distribution aspect of the protocol which is made up by two local buffers connected by a channel.

In the first place we assumed this (logical) channel to be reliable, and proved the correctness of the refinement. In a second step we dropped this assumption and designed two protocol components coping with leaky physical channels. Again, this can be proved to be consistent with the service description.

Concerning specification we first identified the manifold (protocol-)functions. For description we utilized formulas of trace-logic as defined by [Ol2]. They provide for a compact representation of characteristic protocol invariants, their length being independent of the window-size. Compositionality of the semantics allowed for the discussion of *one* protocol function *at a time*. This

[4]The specification of a finite channel with non-deterministic losses is standard. It is not presented here.

simplifies the design a lot, as one has to concentrate on one aspect only. Furthermore, compositionality supports reuse of components. Verification can be done using appropriate proof rules (for arbitrary parameters of the specification) and algebraic methods.

Regarding more complex structures it would be valuable to have the notion of states as part of the logic. (We introduced states implicitly only.) This will be one subject for further studies.

References

[BK1] Bergstra, J.A., Klop,J.W.: Algebra of communicating processes, Report CS-R8421, Centrum voor Wiskunde en Informatica, Amsterdam, 1984

[BDF] Bretschneider,M.; Duque Anton,M.; Fink,B.: Constructing and Verifying Protocols using TCSP, Proc. IFIP TC6 on Protocol Specification, Testing and Verification, Atlantic City, June 1988, Aggarwal,S. and Sabnani,K. (ed), North-Holland Publ. Comp.

[DDM] Degano,P.; DeNicola,R.;Montanari,U.: A new operational semantics for CCs based on condition/event systems, Nota Interna B4-42, Dept. of Computer Science, Univ. Pisa, 1986

[DHKR] Duke,R.; Hayes,I.; King,P.; Rose,G.: Protocol Specification and Verification Using Z, Proc. IFIP TC6 on Protocol Specification, Testing and Verification, Atlantic City, June 1988, Aggarwal,S. and Sabnani,K. (ed), North-Holland Publ. Comp.

[DB] Duque Antón, M., Bretschneider, M.: Modulare Spezifikation und Verifikation von Kommunikationsprotokollen, internal report, 1987

[Fr] Fritschi, K.-D.: Automatische Protokollverifikation in CCS, Diplomarbeit, TU Karlsruhe, Karlsruhe 1987

[ISO] ISO: Information Processing Systems - Open System Interconnection, Basic Reference Model, ISO 7498, 1983

[LS] Lam,S.S, Shankar, A.U.: Protocol verification via Projections, IEEE Trans.Softw.Eng. Vol SE-1O,4,July,1984

[Mi] Milner,R.: A calculus of communicating systems, LNCS 92, Springer,1980

[Ol1] Olderog, E.-R.: Process Theory: Semantics, Specification and Verification, in: de Bakker, J.W. et. al. (Ed.): Current trends in concurrency, LNCS 229, Springer, 1986

[Ol2] Olderog, E.-R.: Nets, terms and formulas: Three views of concurrent processes, REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Nordwijkerhout, the Netherlands, May 30 - June 3, 1988

[Ol3] Olderog, E.-R.: Operational Petri Net Semantics for CCSP, in: Advances in Petri-Nets 1987, G.Rozenberg (Ed.), Springer LNCS 266, 1987

[Ol4] Olderog, E.-R.: private communication

[Ta] Tanenbaum, A.: Computer Networks, Prentice Hall, 1981

[Zw] Zwiers, J.:Compositionality, Concurrency and Partial Correctness, Doctoral Thesis University of Eindhoven, 1988.