Synchronous sequence charts in action

Eindhoven University of Technology
Department of Mathematics and Computing Science

Synchronous Sequence Charts In Action

by

Loe Feijs

95/25

ISSN 0926-4515

Computing Science Report 95/25
Eindhoven, August 1995

Synchronous Sequence Charts In Action

Loe Feijs
Philips Research Laboratories Eindhoven,
Eindhoven University of Technology

August 15, 1995

Abstract

We identify a number of styles for using Interworkings (synchronous sequence charts), together with their roles in the context of the OSI reference model. We employ the well-known ABP (alternating bit protocol) to see how Interworkings can and cannot be used. This experiment shows that the charts are attractive from an intuitive point of view, but when used in their purest form, lack sufficient expressive power. Some of the distinctions in style can be interpreted as distinct approaches to adding expressive power.

Categories and Subject Descriptors: C.2.4 [**Computer communication networks**]: Distributed systems – *distributed applications*; D.2.10, D3.2 [**Software Engineering**]: Design – *methodologies, representation*; Language classifications – *very high-level languages*; F.3.1., F.4.3 [**Logics and meanings of programs**]: Specifying and verifying and reasoning about programs – *specification techniques*; Formal languages – *algebraic language theory*; H.4.3 [**Information systems applications**]: Communications applications – *Electronic mail.*

1 Introduction and motivation

Message Sequence Charts (MSCs) [1] and Interworkings (IWs) [2] are graphical languages for the description of the interaction between entities. Interworkings are in many aspects similar to MSCs; the main difference is that IWs describe synchronous communication, whereas MSCs consider asynchronous communication. We use the term 'sequence charts' to cover both MSCs and IWs. Sequence charts are frequently used for the specification, design and testing of communication systems. It has been shown in [2] that Interworkings can be given a formal semantics in terms of the algebra of communicating processes (ACP) [3]. The ITU (the International Telecommunication Union) is developing a standard for MSCs, based on a formal semantics in ACP too (Annex B of Z.120, [4], accepted in April 1995; see also [5]). Sequence charts are frequently used as parts of (or in combination with) Object-Oriented methods, for example Fusion [6], where they are called 'scenarios'.

In our view, the general understanding of the methodological issues related to sequence charts is still in an early phase. The state of affairs is summarised in [7], where it is noted that the CCITT (ITU) standards Z.100 and Z.120 recommend diagramming techniques and formal languages but do not recommend any methodology of the analytical process. Moreover, it is noted that an MSC specifies only a sample of a particular interaction, not a protocol.

MSCs are more general, but also more complex than Interworkings. For this report we shall restrict ourselves to Interworkings, because we feel we are in an early phase of the development of the methodology, and therefore we like to start with the simplest formalism first. Syntactically, Interworking diagrams can always be viewed as MSCs too (but not the other way round), since synchronous communication demands that the order of arrival is the same as the order of transmission, whereas in the asynchronous MSCs messages can cross each other during transmission.

2 Aims and survey

The aims of this report are twofold:

- to identify the various *styles* in which Interworkings can be used;
- to identify the various *views* of a system described by Interworkings.

The fact that there are various styles is related to the fact that one Interworking is only a part of one trace of a system's execution. In general one needs many Interworkings, but of course one wants to avoid writing very many (or infinitely many) Interworkings. This is a problem which can be approached in various ways, which we call 'styles'.

The fact that there are various views is related to the fact that it makes a difference whether one wants to describe a service, or a protocol, or just a protocol entity. Different architectural aspects give rise to different 'views' on a system. The architectural aspects will be discussed using the OSI reference model, which provides us with concepts such as services and layers [8].

We address these two aims together because in general the style used depends on the view under consideration. For our study of styles and views we employ the well-known ABP (Alternating Bit Protocol), which we will place in an OSI context.

This report is organised as follows. Section 3 briefly introduces Interworkings. Section 4 introduces the ABP. Section 5 surveys the various views and styles which will be studied in the subsequent sections. Section 6 gives a service-oriented view. Section 7 gives a protocol entity-oriented view. Section 8 gives a layer-oriented view. Section 9 gives a peer-to-peer view. Section 10 gives a site-oriented view. Section 11 analyses these distinct views and their relationships. Section 12 discusses the results.

We assume that the reader is familiar with the main operators of ACP: $+$ for alternative choice and $\cdot$ for sequential composition. For more information concerning ACP we refer to [3]. We shall use these operators in Sections 4 and 9 to describe example protocols formally. We will also use them in Sections 6, 7, and 8 to compose sequence chart fragments.

3 Interworkings

An example of an Interworking is given in Figure 1. The vertical lines named ENV1, DISPATCHER and ENV2 represent processes and the horizontal arrows labelled message_1, message_2, etc. represent communication actions between the processes. As shown in [2], Interworkings can be given a formal

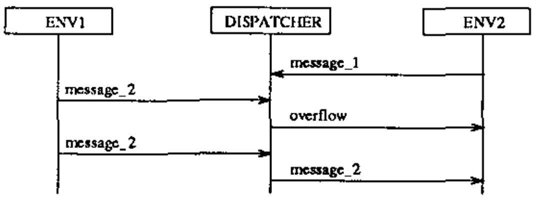

Figure 1: Example Interworking iw_0 with three processes.

semantics in terms of ACP. Roughly speaking, the vertical ordering of events in the diagram is interpreted by using the '$\cdot$' operator of ACP for sequential composition. If we abbreviate message_1 by m_1, message_2 by m_2, overflow by o, DISPATCHER by d, ENV1 by e_1, ENV2 by e_2 we find that for the diagram iw_0 of Figure 1

$$[\![\mathrm{iw}_0]\!] = c(e_2, d, m_1) \cdot c(e_1, d, m_2) \cdot c(d, e_2, o) \cdot c(e_1, d, m_2) \cdot c(d, e_2, m_2)$$

Here we will not consider additional language features of Interworkings, such as internal actions. For this example the semantics seems trivial, but in Interworkings with more processes, a subtle point arises in the sense that one Interworking may contain a number of alternative behaviours (in the ACP interpretation this means that a + appears), which are however equivalent in a certain sense. There is not one total ordering of all the events along a kind of global time line; instead, the Interworking specifies how each process has its own ordering. For an example and a brief explanation of this point we refer to Section 10. We consider the availability of this formal semantics as a strong point of the Interworking formalism. We believe that results of the investigations of Sections 6 to 11 may be of help in making better use of this strong point.

4 The alternating bit protocol

We will adopt the details of the ABP as formally described and analysed using Process Algebra as given in [9]. Rather than starting with its description in Process Algebra we will place the protocol in a context by starting from the OSI concepts, identifying the relevant services and layers first. This is the subject of Section 4.1. Next, we will give an informal description of the working of the protocol (Section 4.2). After that, we will give the formal details in Process Algebra (Section 4.3).

4.1 Architectural issues

The typical positioning of a protocol in a layered open-system architecture (see e.g. [8]) is shown in Figure 2.

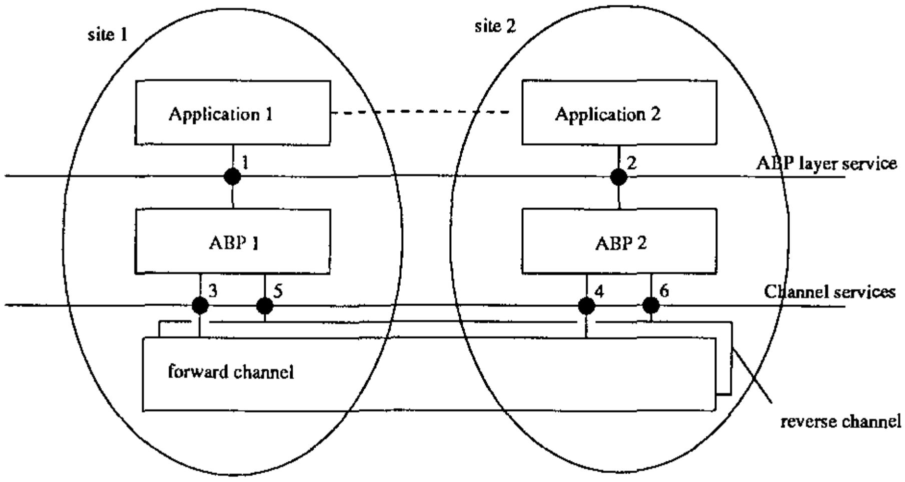

Figure 2: Positioning of the ABP in a layered architecture

The OSI model proposes a total of 7 layers; here we will focus on three layers, one of which is called the ABP layer. The function of this layer is to transmit data packets without damage from site 1 to site 2. To do so it uses the service of the underlying layer, which is here assumed to consist of two unreliable channels, one for each direction. There is a layer above the ABP layer, which we called the Application layer, which uses the services offered by the ABP layer. Services are made available at certain points only, the so-called Service Access Points (SAPs), shown in Figure 2 as black dots. For example, at site 1 there is one SAP for requesting the transmission of a data packet (SAP 1) and at site 2 there is a SAP for receiving the indication of a transferred data packet (SAP 2). The SAPs are numbered from 1 to 6. Both the ABP layer and the Application layer consist of two 'protocol entities' (processes), one at each site. The channel service has two SAPs at each side: one for the forward channel and one for the reverse channel. Note the dashed horizontal line, which represents a 'peer-to-peer' communication channel. This is an abstraction from the true communication channel, which of course does not exist within one layer but passes through the services of the layer below.

In an algebraic setting, as in [9], it is customary, or even necessary, to use short symbols, like s_1 and r_2 for send and receive at sites 1 and 2, respectively. We start from an OSI-style of naming the various service primitives involved. The ABP service consists of one service element, which is data transfer. In real protocol services there are also other service elements, which are related to e.g. connection management, medium access management, etc. There are two service primitives, called request and indication.

Service	Service element	Primitive	Parameters
Data transfer	ABP-DATA	request	User data
		indication	User data

The channel service also comprises one service element. The primitives are almost the same, except for the fact that there is a new data element, not in the User data: ce, for channel error. There is one data transfer service for each of the two channels.

Service	Service element	Primitive	Parameters
Data transfer	C-DATA	request	User data
		indication	User data $\cup$ {ce}

4.2 Informal description

First let us describe the service offered by the ABP. The ABP will repeatedly accept a data packet from the application at site 1 and transfer it to site 2 where it can be received. The contents of the data packet is not modified, nor is any data packet delivered twice or more times. No data packets are lost and the order of the data packets is preserved. There will be no 'spontaneous' packets.

Next let us describe the protocol implementing this service. The two channels are unreliable, but at least they give a warning when the data are corrupted. The warnings are coded by a special value called '*ce*', for channel error. The forward channel is used to transfer the application's data packets together with an additional toggle bit. The sender begins with the value 0 for the toggle bit and then expects an acknowledgement from the receiver. An acknowledgement consists of a toggle bit which is echoed by the receiver and which is conveyed by the C-DATA service of the reverse channel. If the acknowledgement is corrupted, the sender retransmits his data package together with the unchanged bit value. Also if the acknowledgement contains the wrong bit value, the sender retransmits his data package together with the unchanged bit value. But as soon as the right acknowledgement arrives, the sender is ready to accept the next data package from the application, which will be handled in a similar way, but with a reversed value for the toggle bit.

4.3 Formal description

A formal description of the protocol can be found in [9]. In the formal description, which will be summarised below, processes are named S for sender, R for receiver, K for forward channel and L for the reverse channel; together they form a process called ABP. Figure 3 shows the place of these processes.

The formalisation is given in ACP. The service is viewed as a single process ABP satisfying the equation

$$\boxed{ABP \;=\; \sum_{d\in D} r_1(d)\cdot s_2(d)\cdot ABP}$$

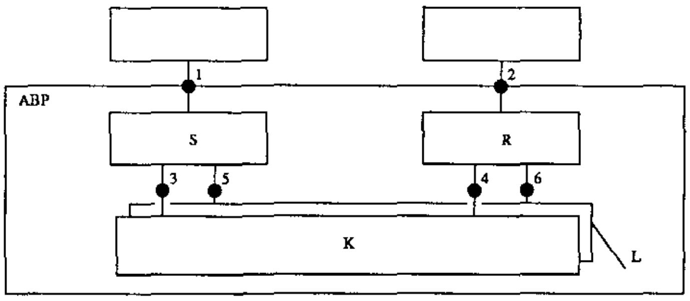

Figure 3: Positioning of the formal processes.

where D is the set of data values, that is, application layer packets. As usual in ACP, summation ($+$ and $\sum$) denotes a choice among a set of alternatives, whereas '$\cdot$' denotes sequential composition. The atomic steps such as $r_1(d)$ and $s_2(d)$ denote receiving d at SAP 1 and sending d at SAP 2, respectively (in ACP, SAPs are usually called 'ports'). We use the convention that a step s_i done by some sending process corresponds to an r_i done by a receiving process. In ACP this is usually formalised using a so-called communication function, whose definition we leave implicit here (and similarly for the encapsulation).

The service as offered by the forward channel, viewed as a process K, is given by the equations

$$\boxed{\begin{array}{rcl} K & = & \sum\limits_{f \in D \times B} r_3(f) \cdot K^f \\ K^f & = & (\tau \cdot s_4(ce) + \tau \cdot s_4(f)) \cdot K \end{array}}$$

As usual in ACP, τ is a 'silent step', used here to model the fact that the channel itself will internally decide whether an error occurs, which is $s_4(ce)$, or a correct transfer, described as $s_4(f)$. $D \times B$ is the set of pairs consisting of a data package $d \in D$ and a bit $b \in B$, where $B = \{0, 1\}$. And of course the reverse channel, viewed as a process L, satisfies analogous equations.

The sender process S is given by the four equations

$$\boxed{\begin{array}{rcl} S & = & RM^0 \\ RM^b & = & \sum\limits_{d \in D} r_1(d) \cdot SF^{db} \\ SF^{db} & = & s_3(db) \cdot RA^{db} \\ RA^{db} & = & (r_5(1-b) + r_5(ce)) \cdot SF^{db} + r_5(b) \cdot RM^{1-b} \end{array}}$$

where we simply write db for the pair consisting of data d and bit b. The auxiliary term RM^b models the sender when Reading a Message in the state where toggle bit b is to be used next. Similarly, SF^{db} is the sender Sending a Frame with data d and bit b. And RA^{db} is when it is trying to Receive an Acknowledgement for d and b.

Finally the receiver process R is also given by four equations.

$$\boxed{\begin{aligned} R &= RF^0 \\ RF^b &= (\sum_{d \in D} r_4(d(1-b)) + r_4(ce)) \cdot SA^{1-b} + \sum_{d \in D} r_4(db) \cdot SM^{db} \\ SA^b &= s_6(b) \cdot RF^{1-b} \\ SM^{db} &= s_2(d) \cdot SA^b \end{aligned}}$$

The auxiliary term RF^b models the sender when Receiving a Frame when b is the correct bit value. Similarly, SA^b is the receiver Sending an Acknowledgement for received bit b. And SM^{db} is the receiver when Sending a Message with data d and bit b to its application.

The following table summarises the relation between the process identifiers used in ACP and the model of Figure 2.

Process	States	Entity	Comment
ABP	-	ABP 1 + ABP 2 + channels	ABP service
S	RM^b, SF^{db}, RA^{db}	ABP 1	sender
R	RF^b, SA^b, SM^{db}	ABP 2	receiver
K	K^f	forward channel	-
L	L^b	reverse channel	-

The following table summarises the relation between the send and receive actions used in ACP and the service primitives mentioned in Section 4.1.

Action	Service primitive	From	To
s_1	ABP-DATA.request	Application 1	ABP 1
s_2	ABP-DATA.indication	ABP 2	Application 2
s_3	C-DATA.request	ABP 1	forward channel
s_4	C-DATA.indication	forward channel	ABP 2
s_5	C-DATA.indication	reverse channel	ABP 1
s_6	C-DATA.request	ABP 2	reverse channel

5 The ABP in Interworkings

In their purest form, Interworkings refer to concrete messages, by which we mean that each message with its actual parameters is shown explicitly. This is sufficient for giving examples of system runs and for representing traces obtained from running the system or a simulation of the system. But one will often want to go beyond this use, to arrive at a more or less complete system specification. But in general, one process or set of processes can exhibit an infinite number of different behaviours. This means that the Interworking language must be implicitly or explicitly extended, either by adopting new syntactic constructs, or by adopting a suitable view on the semantics of a set of Interworkings. We call Interworkings thus used *generic* Interworkings. There are various approaches to generic Interworkings, leading to distinct styles, for which we propose names as follows:

- the operator style (compose Interworkings using operators),
- the inductive style (use Interworkings plus an induction principle),

- the bounded style (give all Interworkings with $< N$ messages),
- the negative style (give Interworkings which should *not* happen).

The operator style, the inductive style and the bounded style will be first explained in Section 6 (one subsection for each style). The explanation of the negative style is postponed to Section 9.

There are also important differences with respect to the view of the system one wants to describe. We distinguish the following views:

- service descriptions (e.g. describing the entire service ABP),
- entity descriptions (e.g. describing the process ABP 1),
- layer descriptions (describing interaction patterns within one layer),
- peer-to-peer descriptions (describing communication along the dashed lines in Figure 2),
- site descriptions (e.g. describing site 1).

Section 6 gives a service-oriented view (in three different styles). Section 7 gives a protocol entity-oriented view (in three different styles). Section 8 gives a layer-oriented view (in three different styles). Section 9 gives a peer-to-peer view and Section 10 gives a site-oriented view.

6 Service descriptions

6.1 Operator style

We use the Kleene star $*$ to denote unbounded repetition (zero or more times). We can treat the data d inside an Interworking as a formal parameter, bound by a Σ construct to denote that for each data value there is an alternative behaviour. In this way we capture an infinity of distinct behaviours in a single expression. The ABP service specification is

$$\Big(\sum_{d \in \text{User data}} \text{iw-s-opb-1}(d)\Big)^*$$

where iw-s-opb-1 is given by Figure 4 below (opb = OPerator Based). Here it is understood from the architecture that 'request' refers to the primitive ABP-DATA.request and that 'indication' refers to ABP-DATA.indication. Here we do not address the precise nature of the $\sum$ and $+$ operators used; a proper treatment requires Mauw's delayed choice operator [10].

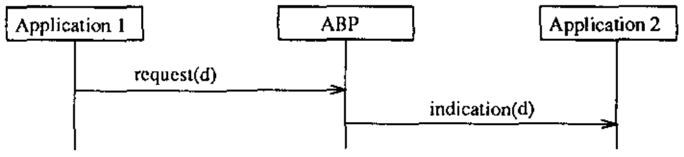

Figure 4: Interworking iw-s-opb-1(d).

Of course it is not hard to invent other syntatic means for the repetition operator. We could for example mark a certain vertical position in the diagram with a label (sometimes called 'condition') and then postulate that the second occurrence of that label means that there is an option for looping back to the position of the first occurrence. This is shown in Figure 5. If so desired one could use a special arrow or a 'GOTO' as well.

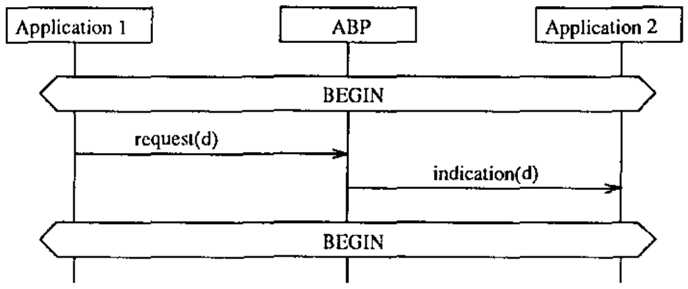

Figure 5: Interworking for ABP service using looping.

The forward channel service specification is

$$\Big(\sum_{d \in \text{User data}} \text{iw-s-opb-2a}(d) + \text{iw-s-opb-2b}(d)\Big)^*$$

where iw-s-opb-2a(d), which describes the intended behaviour, and iw-s-opb-2b(d), which describes the behaviour in case of a channel error, are given by Figures 6 and 7 below. Here it is understood from the architecture that 'request' refers to the primitive 'C-DATA.request' and that 'indication' refers to the primitive 'C-DATA.indication'.

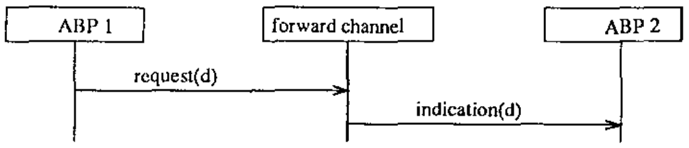

Figure 6: Interworking iw-s-opb-2a(d).

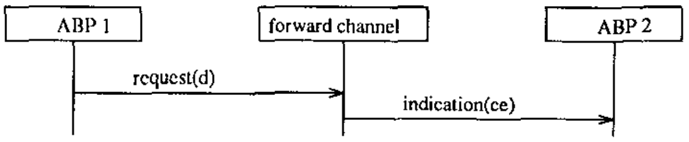

Figure 7: Interworking iw-s-opb-2b(d).

The reverse channel service specification is similar to the forward channel service specification, and has therefore been omitted.

6.2 Inductive style

Next, let us avoid the operators which are used to compose the Interworkings into a description covering all service behaviours. Instead, we will use more Interworkings, each describing one scenario. We still parameterise the Interworkings over the data values (d, d_1, d_2, etc.), but apart from that, each Interworking corresponds to one behaviour. We will begin with the ABP service. After that we shall see that the forward channel presents an additional complication. The number of traces of the ABP service is $\Pi_{n\in\mathbb{N}}|\text{User data}|$, because the trace has countably infinitely many request-indication pairs, each of which can have one of |User data| different data values. So if |User data| = 1 (User data is a singleton), there is only one trace (but this is a reduced kind of service). If User data is a countable set, there are uncountably infinitely many traces. For example, if User data $= \mathbb{N}$ there exists a trace which contains request(3), indication(3), request(1), indication(1), request(4), indication(4) and so on, successively transferring all digits of π.

Even if we allow ourselves to parameterise over d_1, d_2, d_3, etc., we still cannot cast the behaviour into one Interworking because an Interworking is a finite diagram in the sense that it has only a finite number of arrows. We can solve this by adding a key-word 'etc.' at the end of the Interworking and we postulate that this shall only be used if there is an obvious pattern in the arrows shown, from which it is clear how to produce any desired number of additional arrows. In Figure 8 below for example, it is obvious that the first four arrows that should come at the place of the 'etc.' are request($d4$), indication($d4$), request($d5$), indication($d5$). This means that we assume that there is some induction principle, which is not of a mathematical nature, but which, from a practical point of view, may exist when the writers and readers of the Interworkings are able and willing to understand each other.

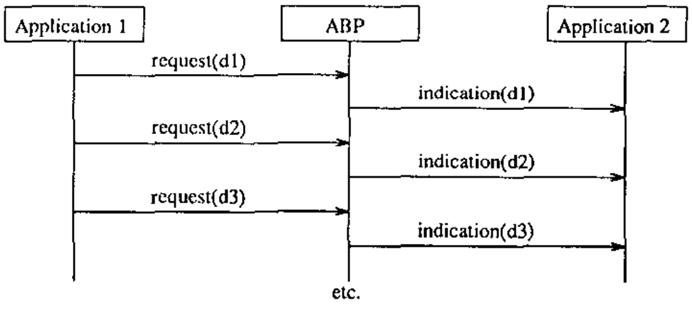

Figure 8: Interworking iw-s-ind-1$(d_1, d_2, d_3, \ldots)$ (0 times ce).

If IND is the operator which extends a given finite diagram to an infinite trace according to the assumed induction principle, then we can define that Figure 8 represents the set of behaviours in

$$\{\text{IND(iw-s-ind-1)}(d_1, d_2, d_3, \ldots) | d_1, d_2, d_3, \ldots \in \text{User data}\}$$

or, using sum notation, the alternatives of

$$\sum_{d_1}\sum_{d_2}\sum_{d_3}\cdots \text{IND(iw-s-ind-1)}(d_1, d_2, d_3, \ldots)$$

The forward channel allows a greater variety in behaviours because it has both desired and undesired behaviours. Figure 9 gives the desired behaviour. Of course this 'request' is 'C-DATA.request'.

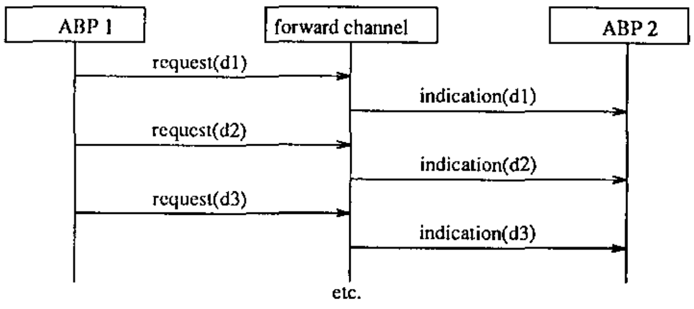

Figure 9: Interworking iw-s-ind-2a($d_1, d_2, d_3, \ldots$) (0 ce).

Next we will present a set of Interworkings which model the behaviours in which there is precisely one occurrence of a channel error. This error may occur during the first transmission, or during the second transmission, or during the third transmission and so on.

These are shown in Figures 10, 11 and 12, respectively. But these are only the first three of an infinite sequence and therefore we add the key-word '<u>etc.</u>' at the caption of the last of these figures (i.e. Figure 12), together with a clue about the applicable induction principle.

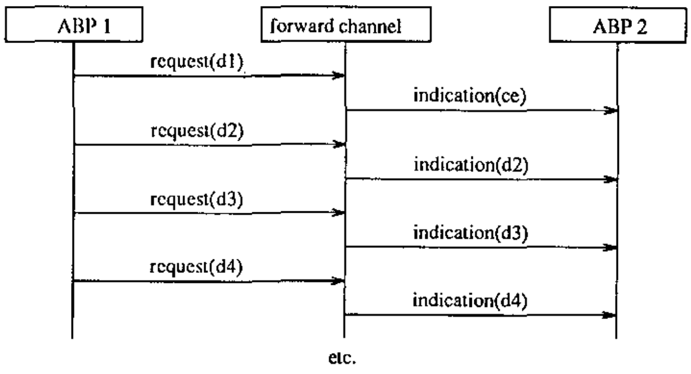

Figure 10: Interworking iw-s-ind-2b($d_1, d_2, d_3, \ldots$) (1 ce).

Next we could continue to present the Interworkings with two channel errors. If the first ce occurs for d1, then there is an infinity of positions for the second ce. In general, the first ce occurs for some d_i and the second for some d_j. This is a two-dimensional space since we can choose both i and j from $\mathbb{N}$ (except for $i = j$), but there is a construction due to Cantor which tells us that there are ways of enumerating these possibilities: first all combinations with $i + j = 2$, then the combinations with $i + j = 3$ and so on. We could give some more Interworkings, but in view of space limitations we will conclude this process now by giving one diagram, together with the '<u>etc.</u>' which tells us that we need all combinations with two ce. And finally we conclude this by the key-word <u>etc.</u> which tells us that after the combinations with one and two occurrences of ce, we get those with 3, 4, 5, 6 and so on.

It depends now on the precise nature of the channel whether this is an adequate specification of the channel. If we want to fix the channel as in Section 4.3, we are not finished yet, because for example the channel behaviour

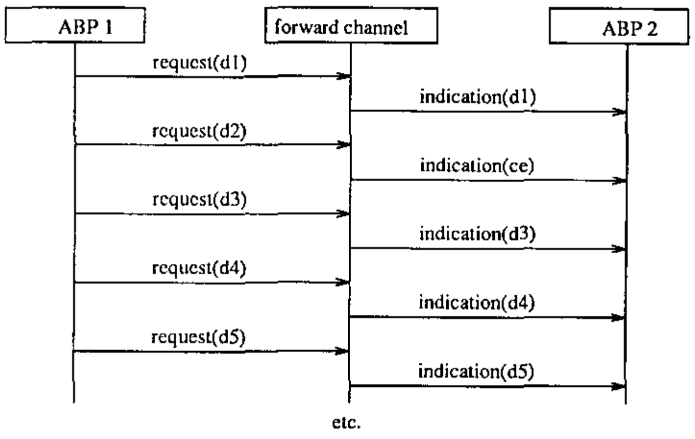

Figure 11: Interworking iw-s-ind-2c($d_1, d_2, d_3, \ldots$) (1 ce).

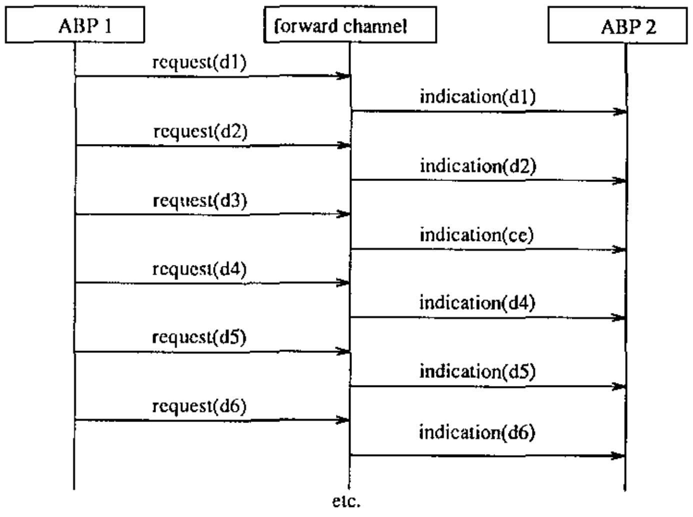

Figure 12: Interworking iw-s-ind-2d($d_1, d_2, d_3, \ldots$), etc. (all combinations with one ce).

which gives its first ce after 3 requests, the second after 1 subsequent request, the next after 4 more requests, following the digits of π, has not yet been included. If we want that too, we could write etc., with the intention that all combinations, including those with an infinite number of occurrences of ce, are included. So apart from the parameterisation over the User data, there are $|\mathbb{R}|$ behaviours. These cannot be enumerated.

As a matter of fact, there is an alternative way of capturing this set of $|\mathbb{R}|$ behaviours, which is to define that the forward channel has the set of behaviours in

$$\{\text{IND(iw-s-ind-2a)}(d_1, d_2, d_3, \ldots) | d_1, d_2, d_3, \ldots \in \text{User data} \cup \{\text{ce}\}\}$$

(using the same Figure 9) or using sum notation, the alternatives of

$$\sum_{d_1} \sum_{d_2} \sum_{d_3} \cdots \text{IND(iw-s-ind-2a)}(d_1, d_2, d_3, \ldots)$$

where it is however understood that the d_1, d_2, d_3, ... now range over User data $\cup$ {ce}. Despite the simplicity of the latter alternative way, this

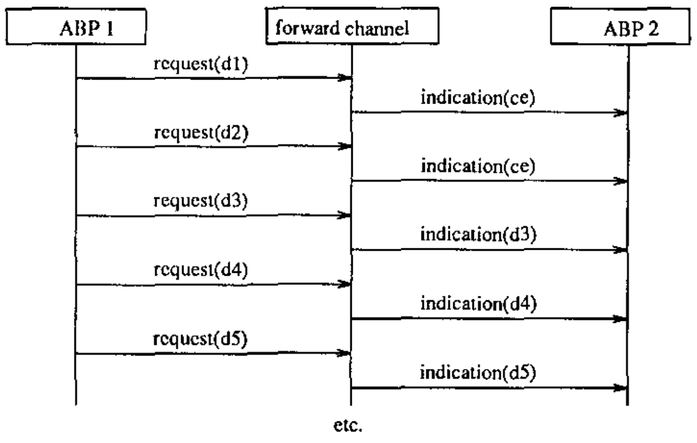

Figure 13: Interworking iw-s-ind-2e($d_1, d_2, d_3, \ldots$), etc. (all combinations with 2 ce), etc (all numbers of ce).

does not seem the typical way in which Interworkings are used: it seems to be more natural to single out the 'ce' case because in the ABP implementation there are entirely different scenarios for the normal case and the 'ce' case.

The reverse channel service specification is similar to the forward channel service specification, and has therefore been omitted.

6.3 Bounded style

Instead of using operators or induction principles to capture an infinite set of behaviours, one can also adopt the viewpoint that the description is satisfactory if all behaviours with N communication actions are adequately described. This viewpoint is defendable when using Interworkings for designing tests or simulation runs, where it is impossible to test an infinite set of behaviours anyhow (if we know bounds on the number of states of the finite state machines we can indicate an N which is really adequate).

For the ABP service we need one diagram for each N. For $N = 2$ this is already given as iw-s-opb-1(d) (Figure 4). For $N = 6$ this is already given as iw-s-ind-1(d_1, d_2, d_3) (Figure 8). It is understood that each diagram is implicitly quantified over its data variables, so if we say that for $N = 2$ the ABP is given as iw-s-opb-1(d) this means that its behaviours are the alternatives of $\sum_d$ iw-s-opb-1(d).

For the forward channel we will restrict ourselves to even N, because the diagrams with odd N are unique extensions of those for $N-1$. Let NDN(N) be the number of diagrams needed for a given number of communications ($N \in \mathbb{N}$).

For $N = 0$ there is only one trivial diagram, so NDN(0) = 1.

For $N = 2$ we need two diagrams, viz. the diagrams obtained by taking the first two arrows only in iw-s-ind-2a(d_1) and iw-s-ind-2b(d_1). So NDN(2) = 2.

For $N = 4$ we need four diagrams, viz. the diagrams obtained by taking the first four arrows only in iw-s-ind-2a(d_1, d_2), iw-s-ind-2b(d_1, d_2), iw-s-ind-2c(d_1, d_2) and iw-s-ind-2e(d_1, d_2). So NDN(4) = 4.

For $N = 6$ we need eight diagrams, wence $\mathrm{NDN}(6) = 8$. In general, $\mathrm{NDN}(N) = 2^{N/2}$, so the number of diagrams grows exponentially with the number of communications covered.

As a matter of fact, there is no fundamental distinction between the inductive style and the bounded style. The inductive style usually requires some nested induction principles, but one could always add a statement '<u>etc</u>' to a bounded description for a length N, implying that the reader is supposed to have grasped the idea and is able to give the diagrams for $N+1$, $N+2$ and so on (at least, in principle). In that sense it is an induction on N. In this report we shall employ the sections on 'bounded style' to have a closer look at the growth of NDN as a function of N.

7 Entity descriptions

7.1 Operator style

The ABP 1 entity (i.e. the sender process) is described by

$$\left(\begin{array}{ll} & \sum_{d \in \text{User data}} (\mathrm{iw}_1(d) \cdot (\mathrm{iw}_2(d) + \mathrm{iw}_3(d))^* \cdot \mathrm{iw}_4(d)) \\ \cdot & \sum_{d \in \text{User data}} (\mathrm{iw}_5(d) \cdot (\mathrm{iw}_6(d) + \mathrm{iw}_7(d))^* \cdot \mathrm{iw}_8(d)) \end{array} \right)^*$$

where $\mathrm{iw}_1 \ldots \mathrm{iw}_8$ are given by Figures 14 to 21. Let us here ignore the problem that formally this only yields the Interworkings with two complete cycles, one for bit 0, and one for bit 1 (and not the incomplete ones). It is understood that here for example $d0$ is the concatenation of the user data value d with the bit value 0. It is possible to push the usage of formal parameters one step further, treating the toggle-bits in a generic way too; this could reduce the number of auxiliary Interworkings by a factor of two, but in the presentation given here we choose to treat the toggle bits as concrete values.

The Interworking diagrams should be interpreted as descriptions of the behaviour of the sender process (ABP 1) under the assumption that the environment (Application 1, both channels) behaved as in the Interworking. In particular, these diagrams are not supposed to give any information about the possible behaviours of that environment. The fact that in general this statement makes a difference becomes apparent when we treat the receiver (ABP 2) along the same lines.

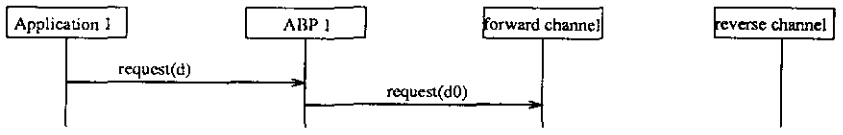

Figure 14: Interworking for ABP 1 (iw_1).

The above description refers to eight auxiliary Interworkings; each of these Interworkings describes a particular 'phase' of the entity being modelled. Actually, we see that these are only small fragments of scenarios; we seem

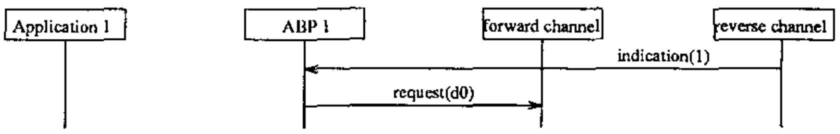

Figure 15: Interworking for ABP 1 (iw_2).

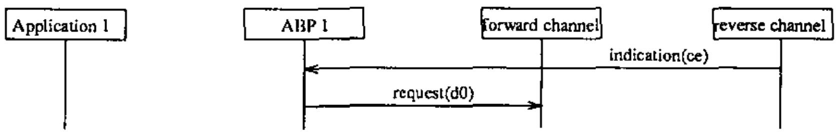

Figure 16: Interworking for ABP 1 (iw_3).

to be (ab)using Interworkings to describe the individual transitions of the process. So iw_1 is one transition, iw_2 is one transition and so on. Such transitions are often modelled in SDL, which is an important state-oriented language for describing processes, see e.g. [11]. This is shown in Figure 22, which combines the transitions of iw_1, iw_2, iw_3 and iw_4 into a single transition diagram.

If we were to duplicate this (replacing 0s by 1s and conversely) we find the full SDL description of the sender. The full SDL description has not been included in order to save space. We conclude that when using the operator-based style for Interworkings, and adopting state transition diagram constructs by way of 'operators' one essentially arrives at SDL (although it must be noted that Interworkings have synchronous communication and SDL has asynchronous communication). Note that parallel branching can be viewed as a representation of the operator + and looping as a representation of the operator *.

We will now turn our attention to the receiver. The ABP 2 entity (i.e. the receiver process) is described by an operator-based expression which refers to six auxiliary Interworkings.

$$\left(\begin{array}{l} \displaystyle (\sum_{d \in \text{User data}} (\mathrm{iw}_1(d) + \mathrm{iw}_2(d)))^* \\ \displaystyle \cdot \sum_{d \in \text{User data}} \mathrm{iw}_3(d) \\ \displaystyle \cdot\ (\sum_{d \in \text{User data}} (\mathrm{iw}_4(d) + \mathrm{iw}_5(d)))^* \\ \displaystyle \cdot \sum_{d \in \text{User data}} \mathrm{iw}_6(d) \end{array}\right)^*$$

where we ought to give 6 Interworkings $\mathrm{iw}_1 \ldots \mathrm{iw}_6$ (restarting the numbering scheme), but here we do not show the diagrams.

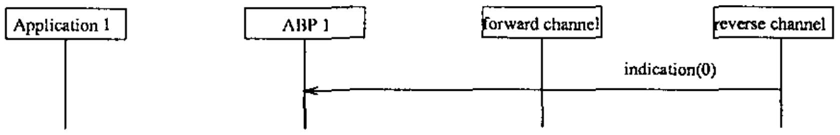

Figure 17: Interworking for ABP 1 (iw_4).

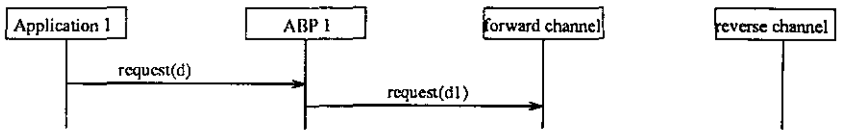

Figure 18: Interworking for ABP 1 (iw_5).

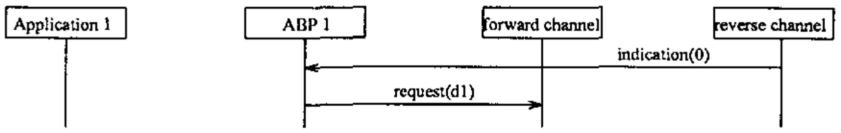

Figure 19: Interworking for ABP 1 (iw_6).

These diagrams describe the behaviour of the receiver process (ABP 2) under the assumption that the environment (Application 2, both channels) behaves as in the given Interworking. Note that the receiver modelled in this way is prepared for acting upon reception of a C-DATA.indication($d1$) as its very first input, in which case the receiver will return C-DATA.request(1). Of course this will not happen when the given implementation of ABP 1 and the given forward channel are used, but the fact that it doesn't happen is a property of the combined system of ABP 1, ABP 2 and the channels, not of the receiver's algorithm, which is described in isolation here.

7.2 Inductive style

We are again going to avoid the operators. We will describe the ABP 1 process. There is an uncountable number of traces. The first Interworking given shows the behaviour of ABP 1 when no channel errors occur. The diagram editor used did not allow for subscripts, so from now on we have variables d, $d1$, $d2$, etc. and it is understood that in this section for example $(d1, 0)$ is the concatenation of the user data value $d1$ with the bit value 0.

As before, we will define that Figure 23 represents the set of behaviours given by its instances, $d1$, $d2$, $d3$ ranging over all user data.

Next we need an infinite set of Interworkings to describe how the ABP 1 process copes with one channel error and with one wrong bit value, which could occur at the first transmission (Interworkings iw-p-ind-2a and iw-p-ind-2b in Figures 24 and 25), the second transmission (figure omitted), etc.

In the same way we need Interworkings to describe how the ABP 1 process copes with two channel errors etc. etc.

Actually, we should have given considerably more Interworkings (at least 20 or so) before ending with the statements etc., etc. and etc., but we have

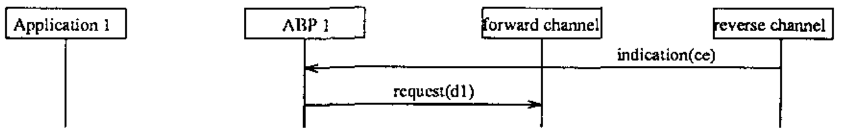

Figure 20: Interworking for ABP 1 (iw_7).

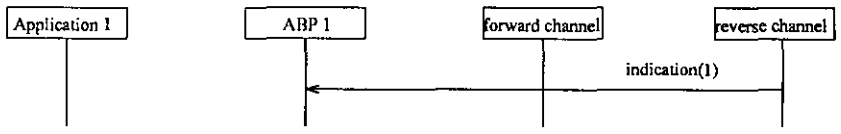

Figure 21: Interworking for ABP 1 (iw_8).

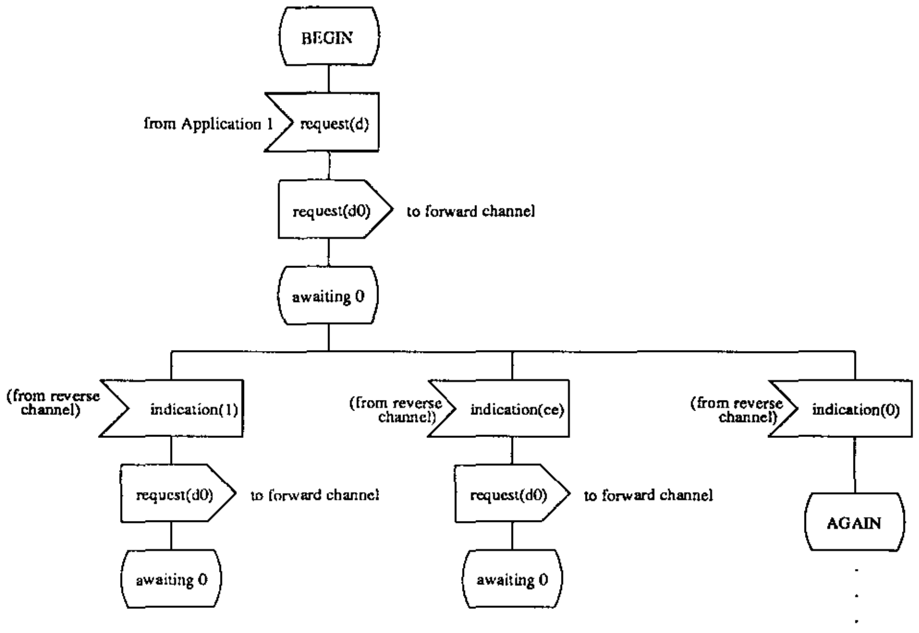

Figure 22: Alternative SDL view of Interworkings iw_1 to iw_4 for ABP 1.

omitted them here.

The other processes can be done in the same way.

7.3 Bounded style

Let NDN(N) be the number of diagrams needed for a given number of communications ($N \in \mathbb{N}$). We will present this for the sender, ABP 1.

For $N = 0, 1, 2$ there is only one diagram, so NDN(0) = NDN(1) = NDN(2) = 1.

For $N = 3$ there are 3 diagrams, and since the ABP 1 process will react in

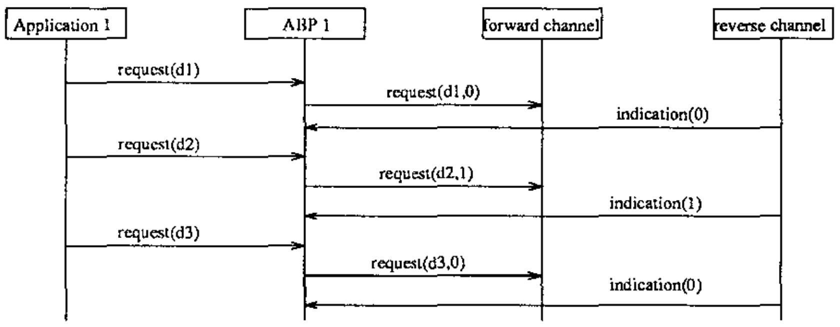

Figure 23: Interworking iw-p-ind-1($d_1, d_2, d_3, \ldots$) (0 times ce).

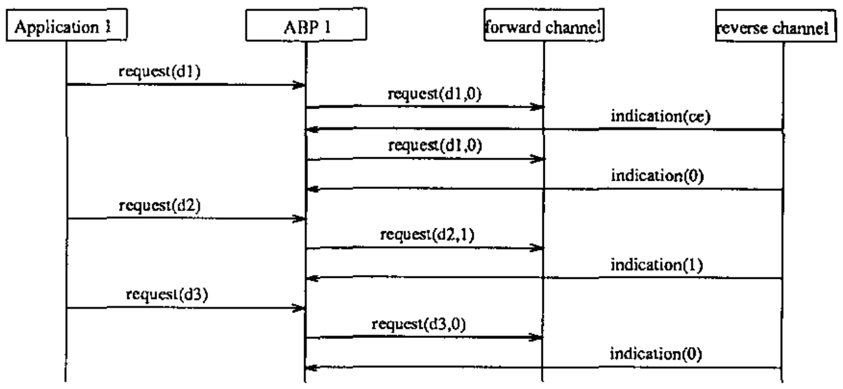

Figure 24: Interworking iw-p-ind-2a($d_1, d_2, d_3, \ldots$) etc. (all combinations with one ce).

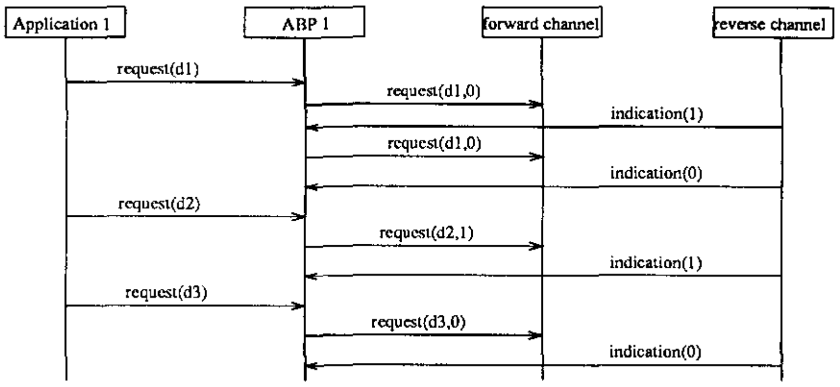

Figure 25: Interworking iw-p-ind-2b($d_1, d_2, d_3, \ldots$) etc. (all combinations with one wrong bit value).

a deterministic way upon the third communication, we find that NDN(3) = NDN(4) = 3. The corresponding Interworkings are the length 3 (4 resp.) prefixes of the Interworkings in Figures 23, 24 and 25.

In general, the Interworkings of ABP 1 are prefixes of the unfolding of the SDL diagram of Figure 22. A part of this unfolded tree is shown in Figure 26. The nodes labelled B are derived from the states labelled 'BEGIN' and 'AGAIN', whereas the nodes labelled W are derived from the wait states 'awaiting 0' and 'awaiting 1'. There are also intermediate nodes, corresponding to the intermediate states between a send and the subsequent receive action. From this diagram we can conclude that the number of Interworkings of a length N is described by the following equations (for $N > 0$):

$$\begin{aligned} B(N) &= W(N-1) \\ W(N) &= I(N-1) \\ I(N) &= B(N-1) + 2W(N-1) \end{aligned}$$

with initial values given by $B(0) = 1$ and $W(0) = I(0) = 0$. Please read $B(N)$ as the number of BEGIN or AGAIN nodes at level N and similarly $W(N)$ as the number of waiting nodes and $I(N)$ as the number of intermediate nodes. Therefore NDN(N) = $B(N) + W(N) + I(N)$. For the first 10 even values

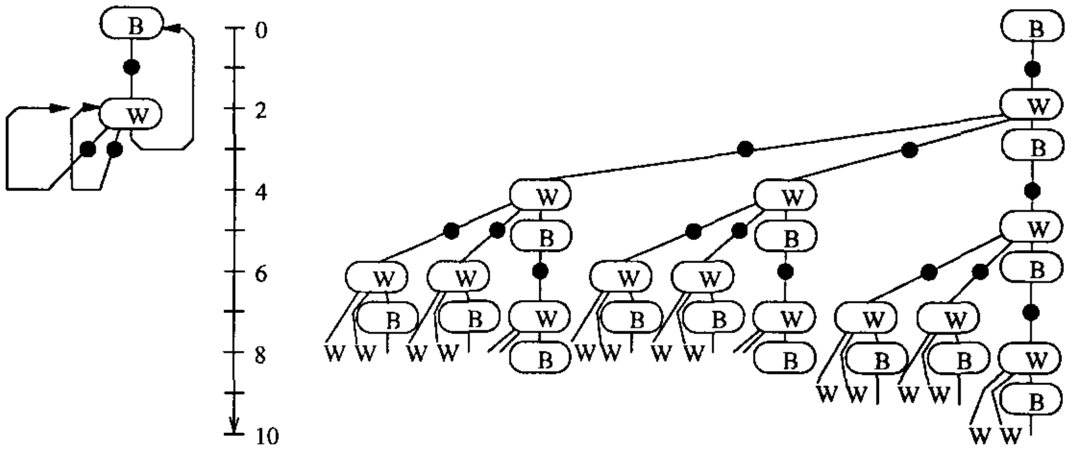

Figure 26: Unfolding the process graph of ABP 1.

of N we have calculated the number of Interworking diagrams needed, as shown in the following table. Please compare the table with Figure 26 and note that for example a horizontal line at depth 6 goes through 9 branches of the tree (four W nodes, one B node and four I nodes), or, in other words, $W(6) = 4$, $B(6) = 1$ and $I(6) = 4$; so $\text{NDN}(6) = 4 + 1 + 4 = 9$.

N	2	4	6	8	10	12	14	16	18	20
NDN(N)	1	3	9	25	67	177	465	1219	3193	8361

The recurrence relations for B, W and I can be solved, yielding a closed form for NDN(N). First we eliminate B and I, finding $W_n = 2W_{n-2} + W_{n-3}$ for $n > 2$. Applying the 4-step procedure of [12] pp. 323–326 we get a closed form for the generating function $W(z) = \sum_n W_n z^n$ which is $z^2/(1-2z^2-z^3)$. From this we obtain $W_n = (\frac{3}{10}\sqrt{5} - \frac{1}{2})\phi_1^n - (\frac{3}{10}\sqrt{5} + \frac{1}{2})\phi_2^n + (-1)^n$ and $\text{NDN}(N) = (1 - \frac{1}{5}\sqrt{5})\phi_1^N + (1 + \frac{1}{5}\sqrt{5})\phi_2^N - (-1)^N$. Here constants ϕ_1 and ϕ_2 are given by $\phi_1 = \frac{1}{2} + \frac{1}{2}\sqrt{5}$ (the golden ratio) and $\phi_2 = \frac{1}{2} - \frac{1}{2}\sqrt{5}$. Taking approximate values for the constants involved, we see that

$$\text{NDN}(N) \approx 0.553 \times (1.618)^N + 1.447 \times (-0.618)^N - (-1)^N$$

The number of diagrams grows exponentially with the number of communications covered. Roughly speaking, each decision to take one more communication action into account (increasing N by one) amounts to a growth of the number of diagrams needed by a factor of 1.618.

In view of the above analysis, the bounded style cannot seriously be considered as a useful specification technique, but the calculation of the number of Interworkings needed is still relevant because it shows how many test runs would be needed to test an implementation of ABP 1 when aiming at exhaustive testing of all traces up to a given length.

8 Layer descriptions

A layer description is a description in which all processes involved in one layer of the protocol are shown in each diagram. For the ABP protocol this means that there are six process lines. The advantage of these descriptions is that they contain much information and that they can help the reader to obtain an overview of what happens during a complex sequence of events. But precisely because of the fact that nothing is left out, the diagrams tend to be large in the sense that they contain many process lines and many interactions.

8.1 Operator style

For the entity descriptions it was already clear that the operator style becomes unattractive when each individual linear fragment must be represented by a named Interworking, in particular if there are many such fragments. If we cast the general interaction pattern of the ABP into this form, we get an operator-based expression as follows:

$$
\begin{array}{ll}
(\sum\limits_{d_1 \in \text{User data}} & \mathrm{iw}_1(d_1) \\
& \cdot\, (\mathrm{iw}_2 \cdot (\mathrm{iw}_3 + \mathrm{iw}_4) \cdot \mathrm{iw}_5(d_1))^* \\
& \cdot\, \mathrm{iw}_6(d_1) \\
& \cdot\, (\mathrm{iw}_7(d_1) \cdot (\mathrm{iw}_8(d_1) + \mathrm{iw}_9) \cdot \mathrm{iw}_{10})^* \\
& \cdot\, \mathrm{iw}_{11} \\
\cdot \sum\limits_{d_2 \in \text{User data}} & \mathrm{iw}_{12}(d_2) \\
& \cdot\, (\mathrm{iw}_{13} \cdot (\mathrm{iw}_{14} + \mathrm{iw}_{15}) \cdot \mathrm{iw}_{16}(d_2))^* \\
& \cdot\, \mathrm{iw}_{17}(d_2) \\
& \cdot\, (\mathrm{iw}_{18} \cdot (\mathrm{iw}_{19}(d_2) + \mathrm{iw}_{20}) \cdot \mathrm{iw}_{21})^* \\
& \cdot\, \mathrm{iw}_{22} \\
)^* &
\end{array}
$$

This expression refers to 22 auxiliary Interworkings, but of course with extra parameterisation we will need fewer. In view of space limitations, we will not show these 22 Interworkings. Let us sketch the idea behind this expression however: iw_1 described a request(d_1) from Application 1 to ABP 1, which is followed by a request(d_1) from ABP 1 to the forward channel. The repetition of $(\mathrm{iw}_2 \cdot (\mathrm{iw}_3 + \mathrm{iw}_4) \cdot \mathrm{iw}_5(d_1))$ describes the sequence of events caused by the ABP 2 repeated request(1) answers, which are repeated until a 0-bit in iw_7 is received. The second repetition describes the attempts of ABP 1 to receive the 0-bit returned by ABP 2. After that we arrive at $\mathrm{iw}_{11} \ldots \mathrm{iw}_{22}$ which describe the second phase of the protocol. The second phase is similar to the first phase, except for the fact that 0s and 1s are interchanged (this is the obvious opportunity for extra parameterisation).

Note that some of the fragments do not depend on the data values (d_1 or d_2) because they only convey 'ce' values or toggle bits.

Although the SDL notation is useful for entity descriptions, as shown in Section 7.1 (see Figure 22), the SDL notation as such can hardly be used when a second or third (vertical) process line must also be represented. Each communication action is not just an input or an output of the main process, but it always connects the lines of two processes. Yet, if we allow repetition operators inside the diagrams, we are able to bring all fragments together in one diagram. Of course one can use loops or labels and GOTO constructs, just like we did in Figure 5, to achieve more or less the same effect. In Figure 27 we show the first half of this diagram. If we were to add another copy of it, replacing all 0s by 1s, all 1s by 0s, and replacing d_1 by d_2, we would get one diagram, such that all possible behaviours would be obtained by running through this diagram one or more times.

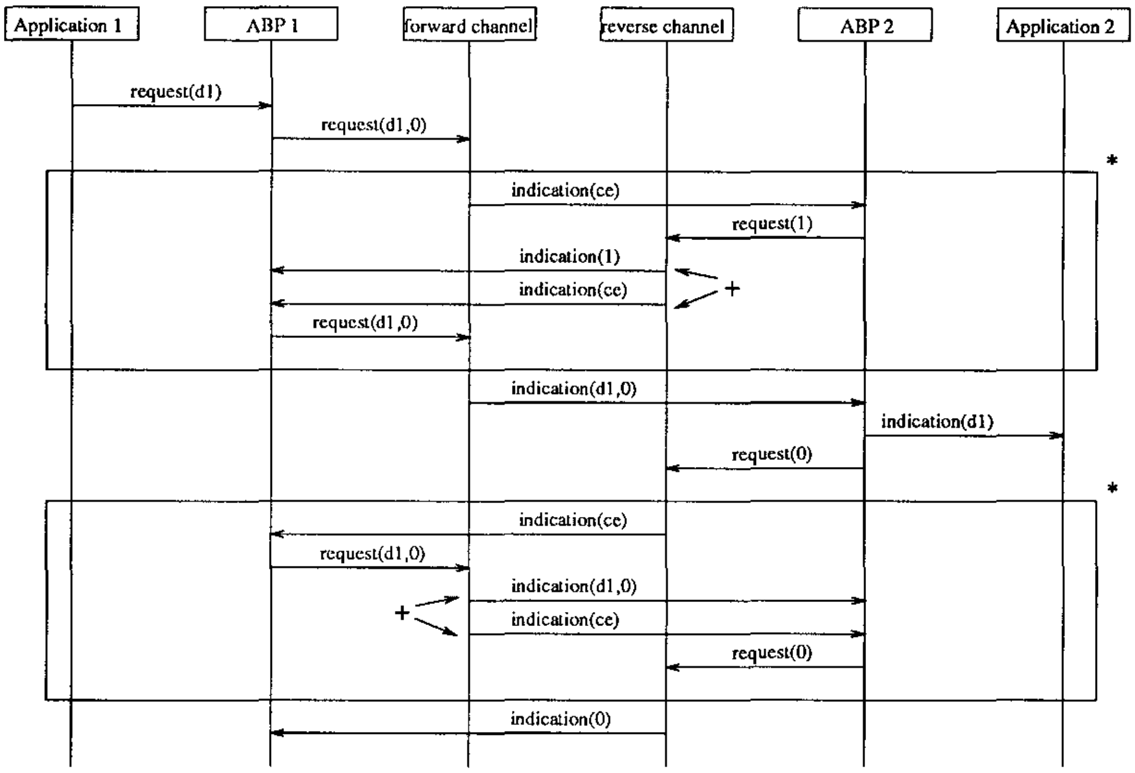

Figure 27: Operator-based layer description of the first protocol phase.

8.2 Inductive style

In the inductive style, each Interworking presents one trace. To capture the entire behaviour, one diagram is not sufficient, but an infinite number of diagrams is needed. Only a finite number of them is given, together with hints (the 'induction principle') on how to proceed and generate as many of them as desired.

We are not going to show all of them, only three. The first Interworking, given in Figure 28, shows the interaction when no channel errors occur. The additional dashed lines and dots will be used in Section 8.3 only; the reader can ignore them for the time being.

As before, we define that Figure 28 represents the set of behaviours given by its instances $d1$, $d2$, $d3$ ranging over all the user data. It is understood that the pattern of the first twelve messages (involving d_1 and d_2) is repeated, but for other data values (d_3, d_4, etc.).

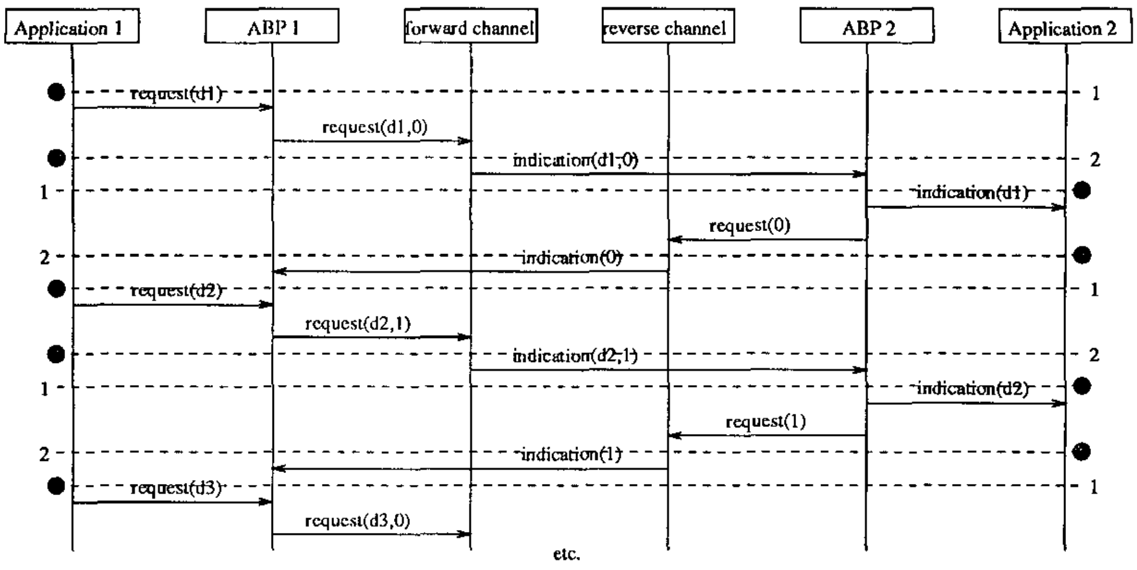

Figure 28: Interworking iw-i-ind-1$(d_1, d_2, d_3, \ldots)$ (0 times ce).

Next we need an infinite set of Interworkings to describe how the ABP 1 process copes with one channel error, which could occur at the first (forward) transmission (Interworking iw-i-ind-2a in Figure 29) or the reverse transmission (iw-i-ind-2b in Figure 30). It could also occur at the second forward or reverse transmission (figure omitted), etc.

In the same way we need Interworkings to describe the interaction with two channel errors etc. etc.

Actually, we should have given considerably more Interworkings before ending with the statements etc., etc. and etc., but we have omitted them here.

8.3 Bounded style

Let NDN(N) be the number of diagrams needed for a given number of communications. In this section we will calculate this number for the layer diagrams containing all the processes. In general, the Interworkings of ABP 1 are prefixes of the unfolding of the diagram of Figure 27. There are only two branching points in the first phase of the protocol. We shall draw these as shaded circles in two versions (dark and light shading, numbers 2 and 3, respectively). We have already included such shaded circles in Figures 28 to 30. In addition, it is convenient to label two additional states which may arise during execution of the protocol: the black state (number 1), which is the initial phase, and the white state, which occurs after the reception of a wrong toggle bit (number 4). The second phase of the protocol follows the same pattern as the first phase, only the 0s and 1s have been interchanged, which of course does not matter for the number of Interworkings involved.

The resulting state transition diagram is shown in Figure 31. This figure also shows a partial unfolding of the diagram, from which we can easily obtain the number of diagrams for $N = 1$ to 9. From this diagram we can

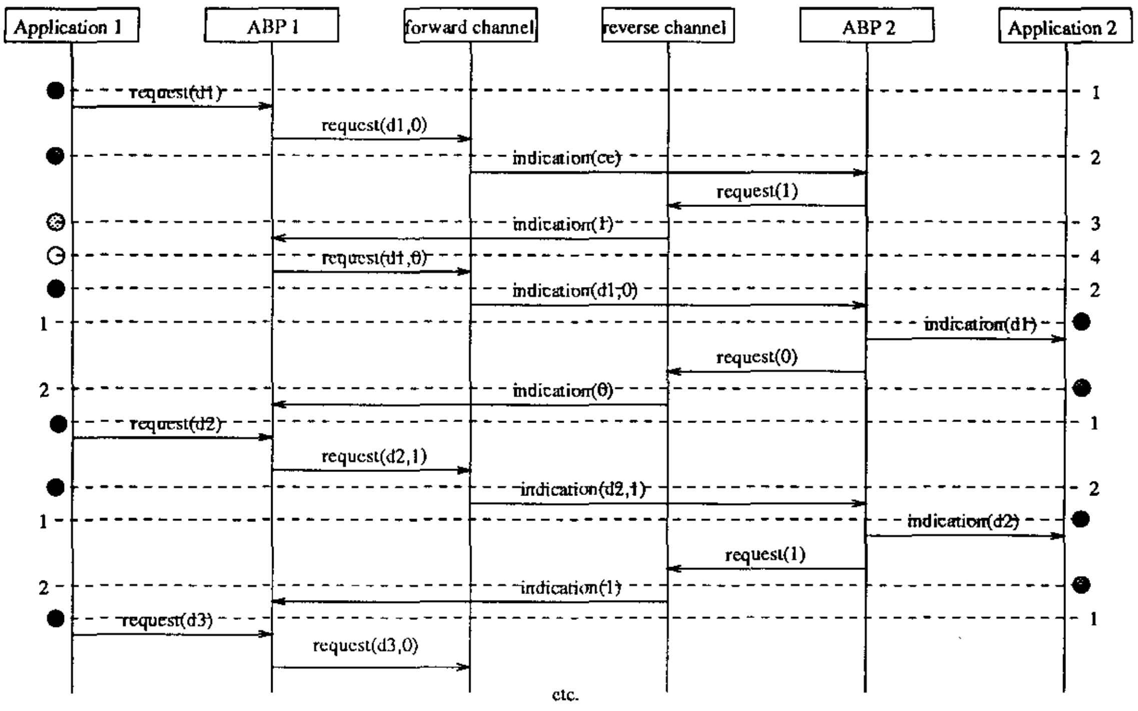

Figure 29: Interworking iw-i-ind-2a$(d_1, d_2, d_3, \ldots)$

conclude that the number of Interworkings of a length $N > 0$ is described by the following equations:

$$\begin{aligned}
B(N) &= D(N-1) \\
D(N) &= B(N-2) + W(N-1) \\
L(N) &= D(N-2) \\
W(N) &= 2L(N-1) \\
I(N) &= B(N-1) + D(N-1)
\end{aligned}$$

The initial values are given by $B(0) = 1$, $B(N) = 0$ for $N < 0$, $D(N) = L(N) = W(N) = I(N) = 0$ for $N \leq 0$. Please read $B(N)$ as the number of black nodes at level N, $D(N)$ the darkly shaded, $L(N)$ the lightly shaded nodes, $W(N)$ as the number of white nodes and $I(N)$ as the number of intermediate nodes.

Clearly, $\text{NDN}(N) = B(N) + D(N) + L(N) + W(N) + I(N)$. For the first ten even values of N we have calculated the number of Interworking diagrams needed, as shown in the following table:

N	2	4	6	8	10	12	14	16	18	20
NDN(N)	1	2	4	7	14	24	47	82	156	279

When comparing this table with the tabel of Section 7.3, one may not conclude that the growth of NDN was more problematic in Section 7.3 than it is here. The main difference is that in Section 7.3 about half of the messages were omitted: the behaviour of process ABP 1 involves 6 messages when

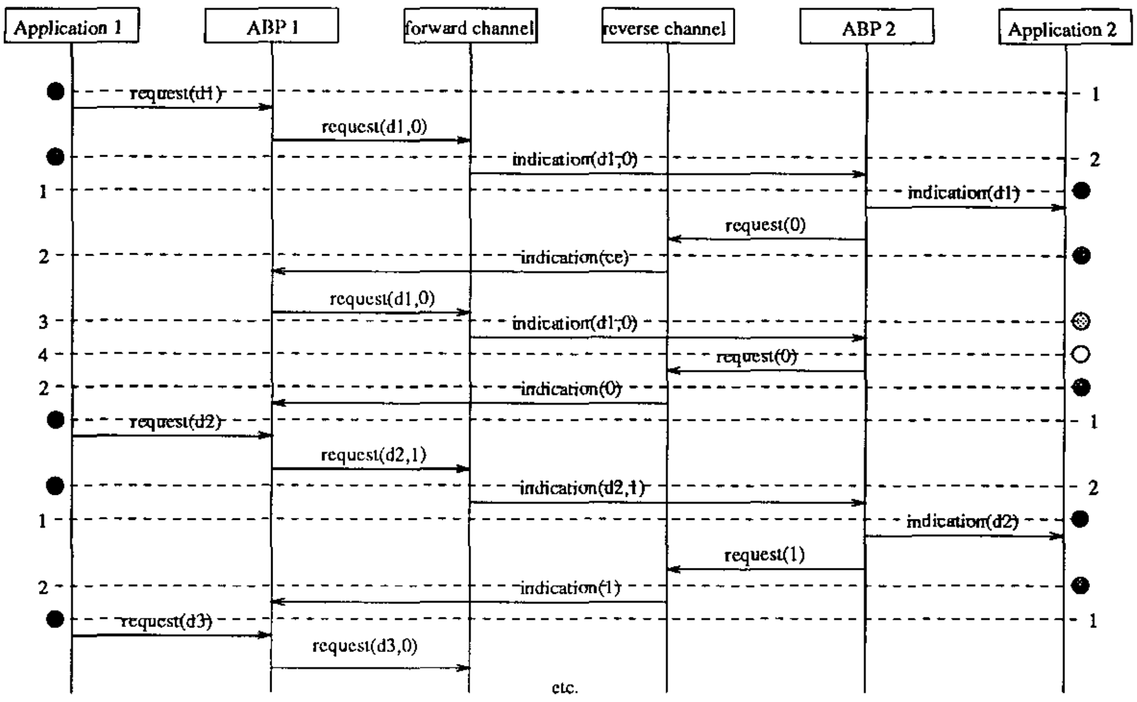

Figure 30: Interworking iw-i-ind-2b($d_1, d_2, d_3, \ldots$) etc. (all combinations with one ce).

going through two protocol phases without errors, whereas the same layer description shows all 12 messages involved.

The number of diagrams grows exponentially with the number of communications covered. For N=40, for example, there are 119618 diagrams.

9 Peer-to-peer descriptions

These descriptions are concerned with the communications performed by one process and its direct partners or an interaction pattern involving many partners. But they are more abstract than the descriptions given so far. They describe so-called 'virtual communications', i.e. communication actions which do not exist as such, but which are obtained by omitting the underlying service, as though the peer protocol entities were to communicate directly, without a mediator. We will give an example first, and will supply the details later (see Figure 32). In order to demonstrate this idea of peer-to-peer description we need to introduce another protocol than the ABP (peer-to-peer does not work well when the underlying medium is not perfect).

The example is an extremely simplified transfer protocol (see for example [13] Section 7.4 for a more comprehensive example of a file transfer protocol). To be able to present this example, we will extend the architecture of Figure 2. We put an FTP layer between the application layer and the ABP layer, as shown in Figure 33. The function of the new layer is to break long messages into shorter ones and to reassemble them again at the receiving site. So its service is to transfer long strings S from site 1 to site 2. To be

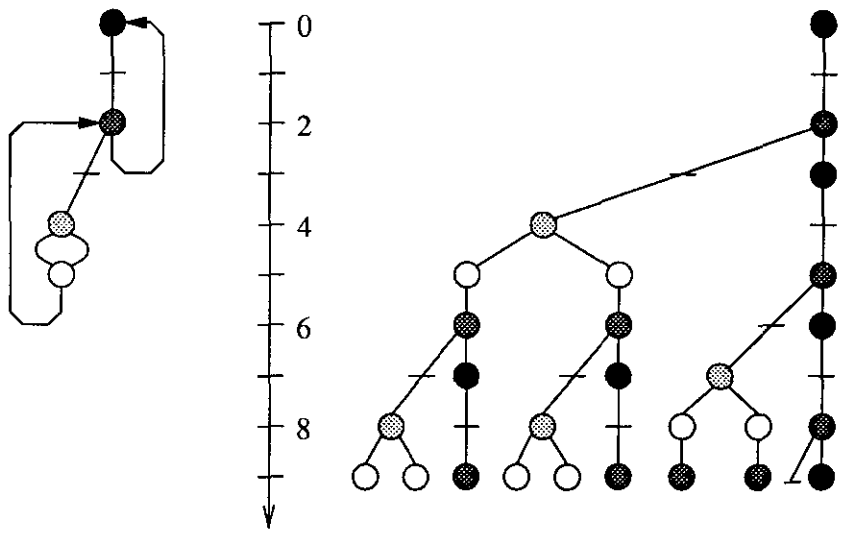

Figure 31: Unfolding the graph of the protocol layer description.

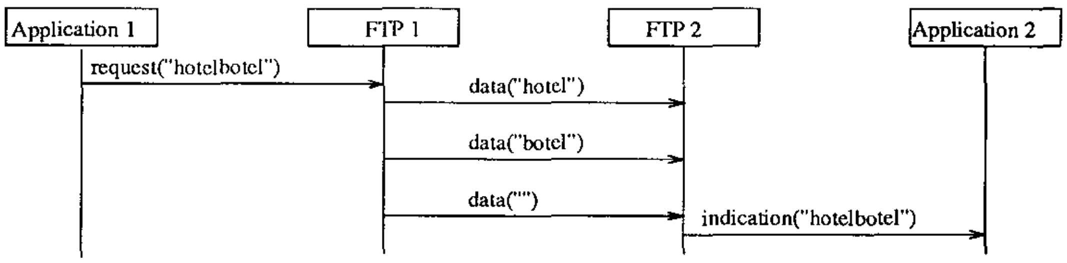

Figure 32: Peer-to-peer Interworking for string S_0 = "hotelbotel".

able to do so it requires the service of the underlying layer (the ABP layer), which provides a reliable channel in one direction. The extended architecture will also be used in Section 10.

The following table summarises the relation between the send and receive actions used in ACP and the usual 'request' and 'indication' service primitives. The indices in s_1, s_2 etc. refer to the service access points in Figure 33.

Action	Service primitive	From	To
s_1	ABP-DATA.request	FTP 1	ABP 1
s_2	ABP-DATA.indication	ABP 2	FTP 2
s_7	FTP-DATA.request	Application 1	FTP 1
s_8	FTP-DATA.indication	FTP 2	Application 2

The protocol works by splitting the given (non-empty) string S into short substrings of a length L, for some $L > 0$. When the transfer is ready, an empty string is sent. When the number of characters in the original string is not a multiple of L, the remaining characters are sent as one short string. This is illustrated in Figure 32 for $L = 5$ and string "hotelbotel" The sender process F_1 is given by the following two equations:

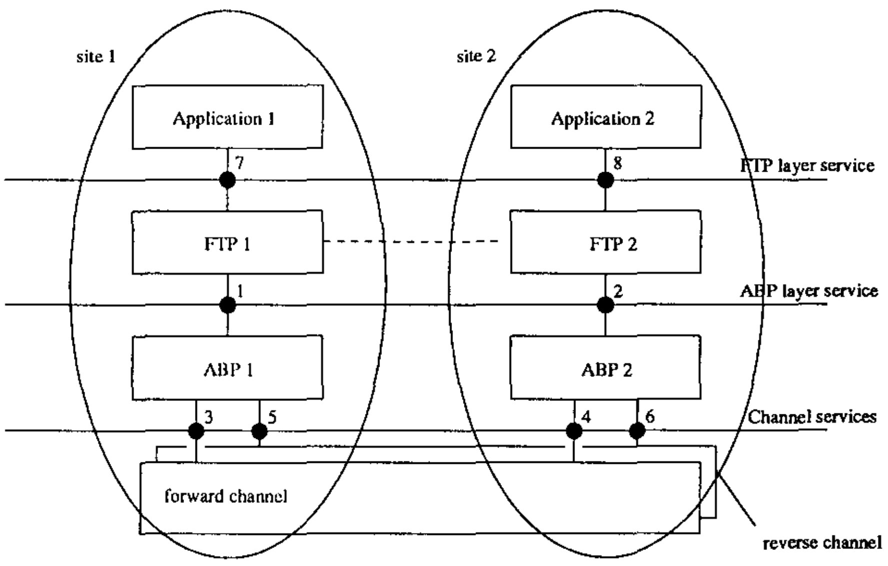

Figure 33: Adding an FTP layer in the architecture

$$\boxed{\begin{array}{rcl} F_1 & = & \sum_{|S|>0} r_7(S)\cdot F_1(S) \\ F_1(S) & = & \text{if } |S| \geq L \text{ then } s_1(S[0..L-1])\cdot F_1(S[L..|S|-1]) \\ & & \qquad\qquad \text{else if } |S| = 0 \text{ then } s_1(\epsilon)\cdot F_1 \\ & & \qquad\qquad\qquad\qquad \text{else } s_1(S)\cdot F_1 \end{array}}$$

where ϵ is the empty string and where we write $S[m..n]$ to indicate the substring of S which begins at index position m and ends at position n (inclusive). We start counting index positions at 0. We write $|S|$ for the length of S and we write $\mathrm{cat}(S_1, S_2)$ for the concatenation of strings S_1 and S_2. The receiver process F_2 is given by the following two equations (there is a deliberate mistake in it, to which we shall return later):

$$\boxed{\begin{array}{rcl} F_2 & = & F_2(\epsilon) \\ F_2(S) & = & r_2(\epsilon)\cdot s_8(S)\cdot F_2 \;+\; \sum_{|d|>0} r_2(d)\cdot F_2(\mathrm{cat}(S,d)) \end{array}}$$

The two processes are supposed to communicate by send (s_1) and receive (r_2) messages; it is understood that an $s_1(d)$ action of F_1 arrives as an $r_2(d)$ action at F_2. In the peer-to-peer descriptions we have one primitive only, shown as 'data', which is a kind of combination of a request and an indication (or s_8 and s_7, which is the same).

We will return to the methodology of Interworkings. As before, we can distinguish the operator style, the inductive style, the bounded style and the negative style. We shall not elaborate on the differences between the first three of these styles here, since the problems are similar to those studied for the ABP (but simpler because the protocol has no internal non-determinism). Essentially, the operator, inductive and bounded styles amount to giving a sufficient number of diagrams like Figure 32, but for other initial strings S.

We shall use the file transfer protocol to illustrate the negative style later in this section.

Let us take a closer look at Figure 32 and the abstraction step involved in arriving at this figure. The diagram of Figure 32 is an abstraction of the diagram of Figure 34 (apart from the fact that the applications are omitted in Figure 34).

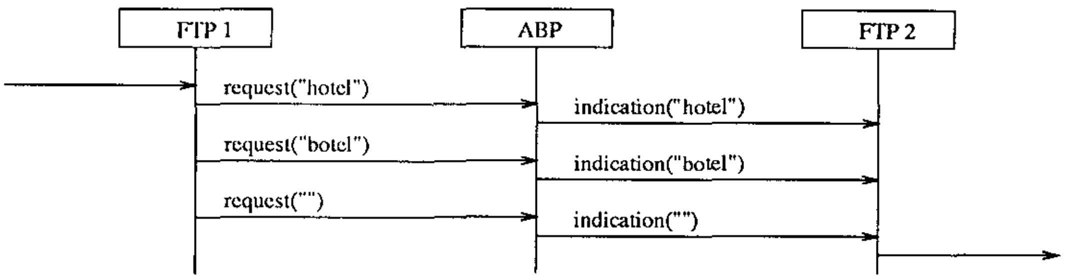

Figure 34: Interworking without peer-to-peer abstraction.

For each pair of communication actions in Figure 32 there is precisely one communication action in Figure 34. More precisely, a pair consisting of a message request(d) from FTP 1 to ABP immediately followed by an indication(d) from ABP to FTP 2 is translated into a single communication action of the form data(d) which goes from FTP 1 to FTP 2.

Please note that the peer-to-peer descriptions can only be given as an Interworking because by the end of each communication action it has already been determined which process will initiate the next communication. This is typical of master-slave protocols or token-passing protocols. In general, however, both applications can initiate a communication action. If for example we assume that FTP 2 can send a spontaneous 'disconnect' to FTP 1, we get for example the scenario of Figure 35. So, we could be forced to use MSCs instead of Interworkings. If the underlying service can loose messages, we

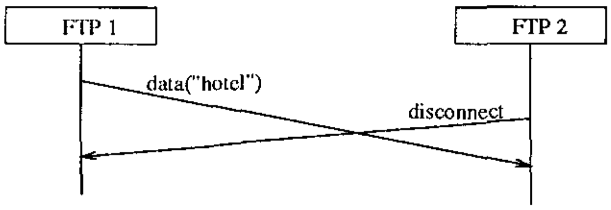

Figure 35: Peer-to-peer description with simultaneous initiatives.

have an additional problem, calling for arrows which begin at the sender's process line, but end halfway (or are dashed, or otherwise distinguished so as to indicate the error). For example, if the underlying service (unlike the ABP) sometimes looses its messages, the scenario of Figure 36 may occur.

As promised, we shall now explain the negative style. The negative style is to draw one or more diagrams for traces which *should not occur*. Consider for example Figure 37 (assume S_0 = "hoteldebotel" and $L = 5$). We included put a deliberate flaw in the formulation of the protocol. The given algorithm

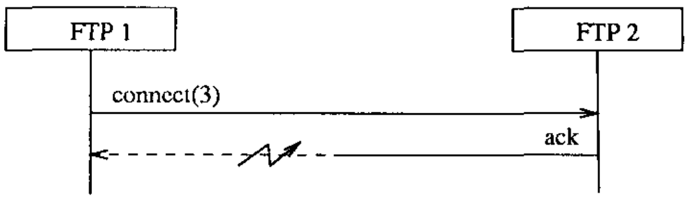

Figure 36: Peer-to-peer description when the service can loose a message.

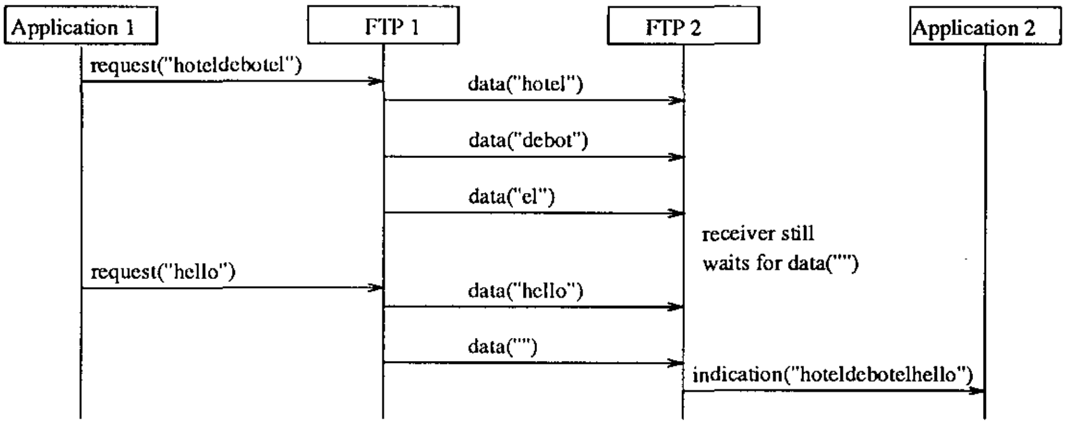

Figure 37: Interworking demonstrating error ($S_0 =$ "`hoteldebotel`", $L = 5$).

for F_1 only sends the empty string if the length of S_0 is a multiple of L. Figure 37 demonstrates the error: the empty string is not sent and therefore the receiver does not see the end of the first string. This use of Interworkings occurs frequently: to illustrate an error, which is repaired subsequently. In this case the error is easily repaired by changing the definition of the sender process F_1 as follows:

$$\boxed{\begin{array}{rcl} F_1 & = & \sum_S r_7(S) \cdot F_1(S) \\ F_1(S) & = & \text{if } |S| = 0 \text{ then } s_1(\epsilon) \cdot F_1 \\ & & \qquad \text{else if } |S| \geq L \text{ then } s_1(S[0..L-1]) \cdot F_1(S[L..|S|-1]) \\ & & \qquad\qquad \text{else } s_1(S) \cdot F_1(\epsilon) \end{array}}$$

In Figure 38 a scenario is shown for the repaired protocol, using the same string which went wrong before.

10 Site descriptions

In this section we shall use the extended architecture presented in Figure 33. This is because when there are three layers only, the difference between entity descriptions and site descriptions vanishes. A site description shows all the interactions which occur at one site. So a site description of site 1 contains all the communications which occur at ports 1, 3, 5, 7, whereas a description of site 2 contains those of ports 2, 4, 6, 8. This is for example useful when one person or team is in charge of analysing or designing all protocol entities

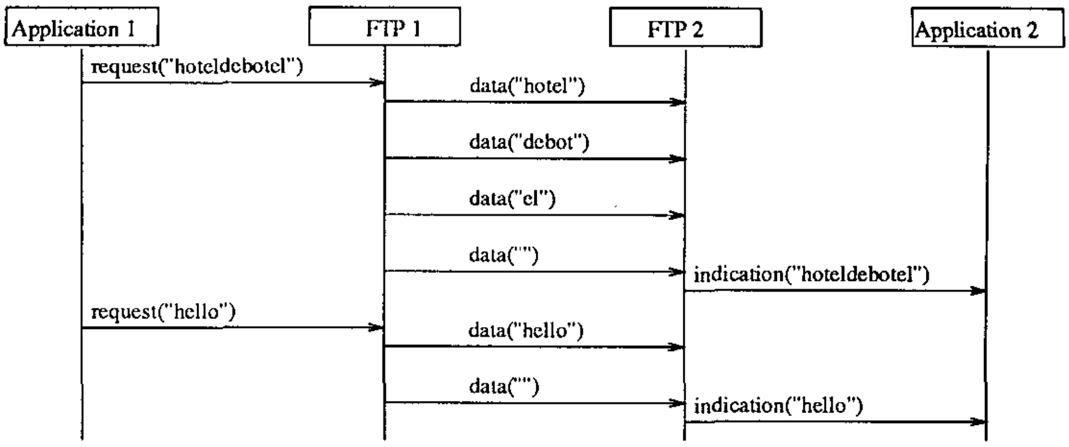

Figure 38: Interworking demonstrating bug-fix.

of one site. This is also the kind of description of course obtained when all events occurring at one site are logged, which can be done locally, unlike for example a layer description, which requires some distributed logging. In practice, site descriptions are usful when the subdivision of the protocol design into layers is still to be analysed. Once there is a clear layering, the diagrams are less useful.

For example, let us ask the question: "what exactly happens at site 1 when the application invokes request(`"hotelbotel"`) and the reverse channel fails to deliver a correct data value for its first request?" This is shown in Figure 39. Figure 40 shows the corresponding sequence of events at site 2. It is possible to simplify this diagram one step further, by combining the forward channel and the reverse channel into one environment process, but we shall not explore that option here.

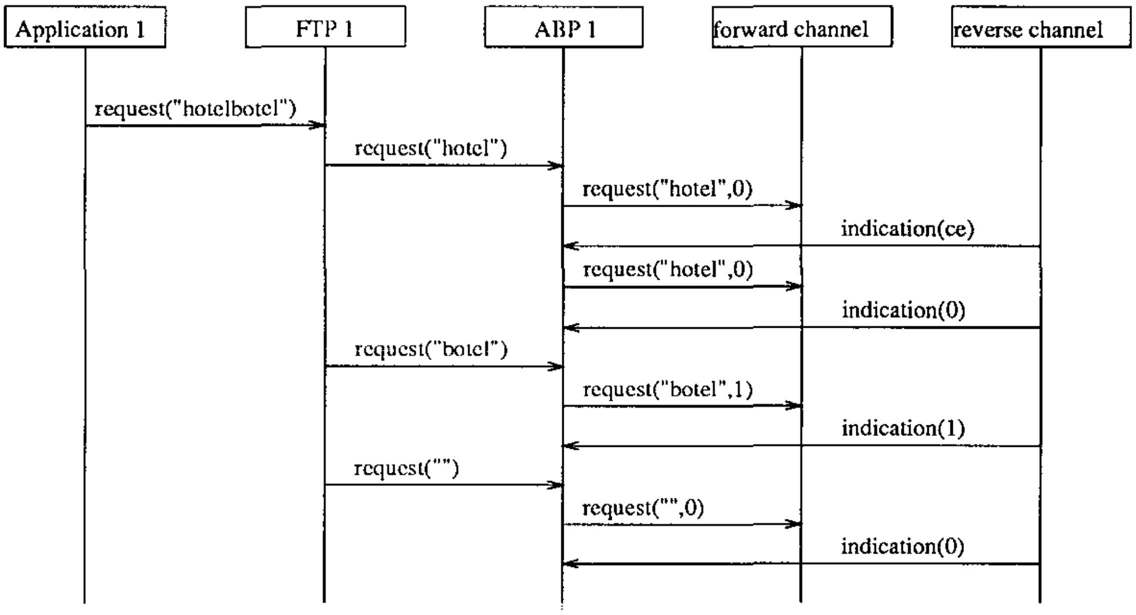

Figure 39: Site description for site 1.

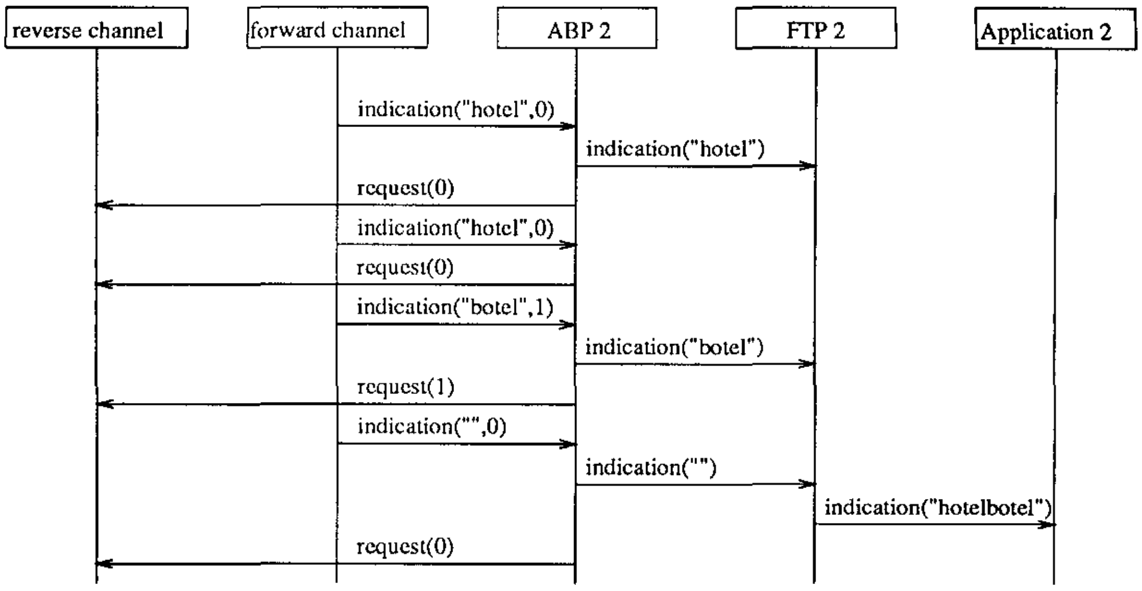

Figure 40: Site description for site 2.

As before, we can distinguish the following styles:

- the operator style,
- the inductive style,
- the bounded style,
- the negative style.

We shall not elaborate on the differences between these styles here, since the problems are similar to those studied for the ABP (but more complex because we are looking at many processes at the same time). Although one could try to achieve a certain completeness (for example by using the inductive style) with respect to the specification of the protocol entities ABP 1 and FTP 1 from the site 1 descriptions, this is not a good idea. It is already hard to arrive at a complete description when specifying ABP 1 or FTP 1 in isolation. Trying to combine the descriptions implies specifying a more complex system (2 processes in parallel). These diagrams are of course not suitable for specifying a service, because a service is a distributed concept, which in general cannot be explained from the point of view of one site only.

Figure 40 presents a good opportunity for discussing the phenomenon of equivalent diagrams. The last two messages of Figure 40 do not share a process line. If we interchange two adjacent communication actions which share no process line, then we get another diagram (putting request(0) before indication("`hotelbotel`")) which can be considered equivalent to the given diagram. Conversely, one could define that each diagram represents the entire equivalence class of traces. This idea is the essence of the approach of Mauw, Winter and Van Wijk, who define an ACP term (containing the + operator) as the semantics of a given diagram (see e.g. [2]).

11 Relating the views

We shall relate the various views by defining an impractical, but theoretically interesting view, which is the view in which all actions of all the processes are shown, without the omission of any processes and without service-abstraction or peer-to-peer abstraction. The processes and connections involved are shown in Figure 41.

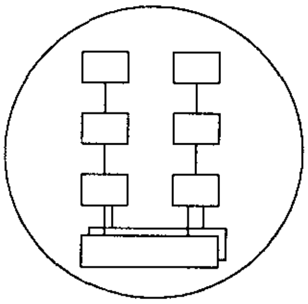

Figure 41: The complete view

We can obtain the various views through the omission of certain processes, possibly combined with an abstraction step. A number of views are shown in Figure 42. The processes which are not involved in a particular view are shaded. Ports which are not involved have been omitted from the figure. The oval indicates the area of interest.

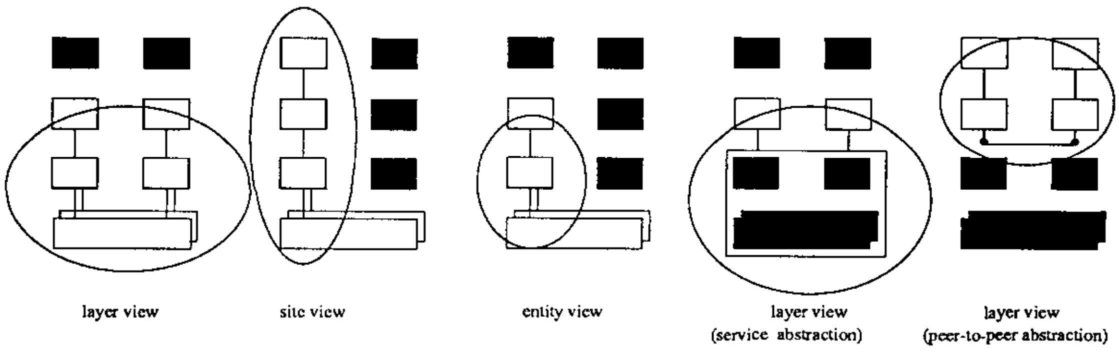

Figure 42: Views obtained by omitting processes and through abstraction.

The main characterisation of the views is based on the main decomposition structures visible in the architecture as presented in Figure 33: decomposition into layers and distribution over sites. By restricting a view to the intersection of a layer and a site we see only a single entity, which for the simple protocols studied means that we see only a single process. Therefore, these are the three main views:

- layer descriptions,
- site descriptions,
- entity descriptions .

For the layer descriptions there are two ways of abstracting away from particular details, either by combining a number of processes into a single service, or by cutting out the service processes of the next layer down. Therefore we regard the following views as more abstract variants of the layer descriptions:

- service descriptions,
- peer-to-peer descriptions.

The layer descriptions for the ABP layer were presented in Section 8. We did not present the layer descriptions for the FTP layer, but they could be made if so desired. A few site descriptions for site 1 and site 2 were presented in Section 10. The entity descriptions for the ABP 1 entity were presented in Section 7. We did not present the entity descriptions for the ABP 2 process, but they could be made if so desired (the same holds for FTP 1 and FTP 2). The service descriptions for the ABP service were presented in Section 6. We did not present the service descriptions for the FTP service, but they too could be made if so desired. The peer-to-peer descriptions for the FTP layer were presented in Section 9. There is no peer-to-peer description for the ABP service, because the mechanism of peer-to-peer abstraction is not directly applicable to an unreliable service.

It is interesting to analyse the way in which some of the descriptions can be obtained from other descriptions. This is shown in Figure 43.

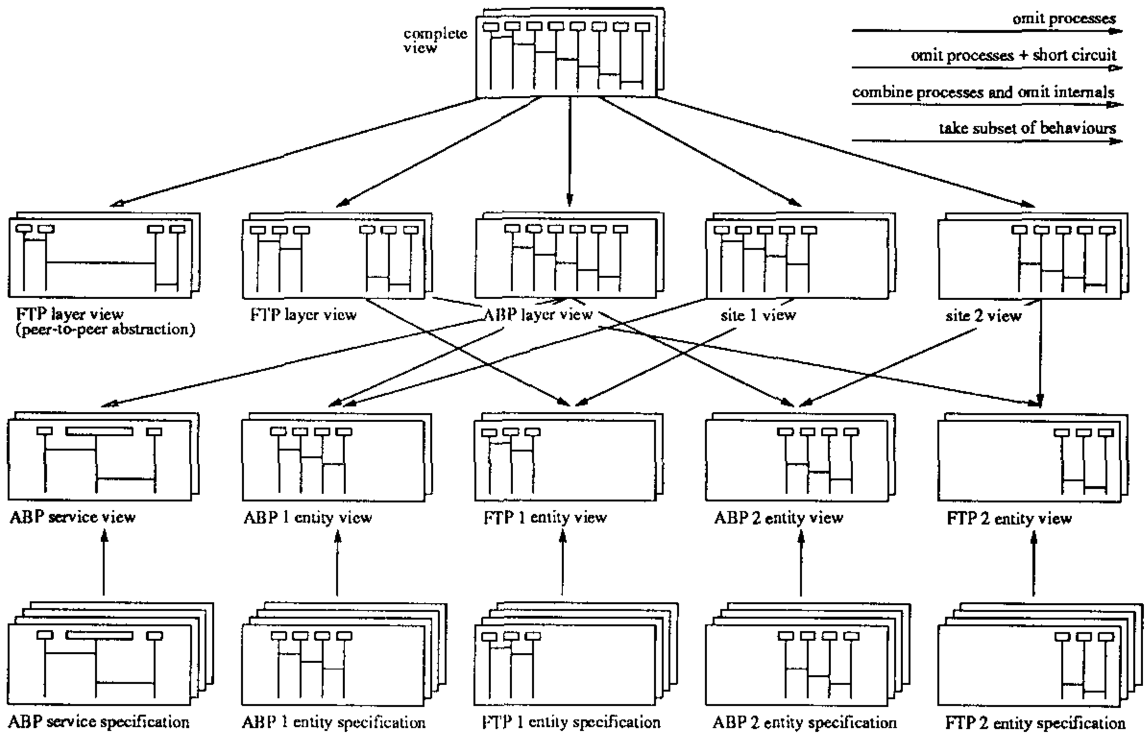
complete view
omit processes
omit processes + short circuit
combine processes and omit internals
take subset of behaviours
FTP layer view (peer-to-peer abstraction)
FTP layer view
ABP layer view
site 1 view
site 2 view
ABP service view
ABP 1 entity view
FTP 1 entity view
ABP 2 entity view
FTP 2 entity view
ABP service specification
ABP 1 entity specification
FTP 1 entity specification
ABP 2 entity specification
FTP 2 entity specification

Figure 43: Relations between the various descriptions.

At the top of this figure we put the description of the complete view; we assume that this is a description containing all traces which may occur in putting all eight processes (Application 1, FTP 1, ABP 1, forward channel, reverse channel, ABP 2, FTP 2 and Application 2) in parallel. Of course this is an infinite set which includes infinite traces, but here we assume that we can succeed in making a finite description of this set, using the operator style, the inductive style, the bounded style, the negative style, or any combination of these. Let us use the term 'view' for such an infinite set of traces (behaviours). The second and the third rows in Figure 43 are obtained by applying three different kinds of forgetful and/or abstraction mappings, as indicated. For example, the 'site 1 view' is the set obtained from the 'complete view' by removing three processes (ABP 2, FTP 2 and Application 2) from each Interworking.

The views in the lowest row in Figure 43 are each related to one other view by means of a subsetting relation. This is because the specification of an entity (or service) concerns all possible behaviours and may hence include behaviours which do not occur when the entity is put in the specific context of the other entities of the ABP and the FTP protocols. For example the ABP service specification must include the behaviour where two successive empty strings are to be transmitted (request(ϵ)), each of which must of course be delivered at site 2 (as an indication(ϵ)). But when used for the particular FTP protocol described in Section 9, this behaviour cannot occur.

The various relations between the distinct views and specifications can be used as a basis for obtaining an understanding of the roles of the distinct checking options, as proposed by Mauw, Reniers, Winter and Van Wijk (see e.g. [2], and [15]). One important kind of check is called 'merge-consistency'. This consistency amounts to checking whether two given Interworkings can be derived from one (unknown) common Interworking by deleting one or more processes. This can be used for example to see if a given Interworking belonging to the 'ABP layer view' and a given Interworking belonging to the 'site 1 view' are merge-consistent with each other. Merge-consistency can be checked by a tool that attempts to combine the two given Interworkings into a single Interworking containing the union of the processes of the individual Interworkings. Of course it is not so that *any* Interworking of the ABP layer view is merge-consistent with any Interworking of the site 1 view.

A second kind of check is related to the concept of 'refinement'. One Interworking is said to be a refinement of another Interworking if the latter is obtained from the former by unifying a number of process-lines, turning them into one process-line and possibly also deleting internal actions (messages which go return to the unified process). This can be used for example to see whether a given Interworking which is supposed to belong to the 'ABP service view' can be refined to a given Interworking which is claimed to belong to the 'ABP layer view'.

12 Discussion

Our analysis of the various styles (operator style, inductive style, bounded style and negative style) show that Interworkings in their purest form, when considered as a specification formalism, lack sufficient expressive power. Even in the case of a small and well-known protocol like the ABP, it turns out that all attempts to arrive at a reasonable coverage of the set of allowed behaviours by means of a sufficiently large number of Interworkings, are problematic. For example, we discovered that the number of Interworkings needed for one process grows exponentially as a function of the number of messages shown in each Interworking.

The use of *generic* Interworkings, where expressive power is added by means of induction principles or operators, allows for a variety of solutions, roughly falling into two categories:

- add composition mechanisms as a separate language level to the Interworking language. This was illustrated by our inductive notation using etc. and etc. and by our use of $*$, $+$ and Σ. It is essentially the approach followed for MSCs in the GEODE toolset. In the context of the ITU MSCs such high-level constructs are called 'road-maps'. The Interworking merge and Interworking sequencing of [2] and [15] also belong to this category.
- extend the Interworking language itself, embedding choice and repetition into the diagrams. We showed this in Figure 5 and Figure 27. The representation of choices or if-then-elses is the hardest problem of this category: they must be put into the two-dimensional structure of the diagrams. See for example [14], where the concept of condition is extended to form both guards and labels to indicate the 'then' and the 'else' part of an if-then-else construct. Also see [14] for sub-MSCs, providing a subroutine mechanism.

So, when trying to use Interworkings as a specification language, some choices have to be made. For other purposes the situation is easier: when analysing and explaining a limited number of interesting cases such as test runs, simulation runs, debug sessions, etc., Interworkings and MSCs are already useful in their pure form.

Our analysis of the various views provided insight into the applicability of Interworkings in the context of the OSI model. The OSI layering and distribution concepts give rise to a number of distinct views, for which we proposed names, viz. layer descriptions, site descriptions, entity descriptions, service descriptions and peer-to-peer descriptions. We showed several examples and made the relations between these views explicit in an informal setting. We claim that this understanding and classification of views is necessary and is complementary with respect to the emerging formal semantics of Interworkings and MSCs and the formal definitions of merge-consistency and refinement. The formal definitions cannot really be exploited if we have no clear view of why merging and refinement are needed in practice. On the other hand, the informal analysis given here can be made precise (but not in the scope of this report) using formal notions of merge-consistency and refinement. Although the definitions of the views are based on observations made during industrial protocol design projects (e.g. we saw site views being written in practice), the work reported here is a fully self-contained rational reconstruction of how Interworkings can and cannot be used in practice.

13 Acknowledgements

The author wishes to thank Ron Koymans, Yat Man Lau, Sjouke Mauw and Michel Reniers for their help and cooperation with respect to the subject of this report.

References

[1] E. Rudolph, P. Graubmann, J. Grabowski. *Towards an SDL design methodology using sequence chart segments.* In: O. Færgemand and R. Reed (Eds.), SDL'91, evolving methods, Elsevier Science Publishers, pp. 237–252 (1991).

[2] S. Mauw, M. Van Wijk, T. Winter. *A formal semantics of synchronous Interworkings.* In: O. Færgemand and A. Sarma, (Eds.), SDL'93 using objects, Elsevier Science Publishers pp. 167-178 (1993).

[3] J.C.M. Baeten, W.P. Weijland. *Process algebra.* Cambridge University Press, Cambridge tracts in theoretical computer science 18 (1990).

[4] ITU-TS. *Draft Recommendation Z.120 Annex B: Algebraic semantics of Message Sequence Charts*, ITU-TS, Geneva (1994).

[5] S. Mauw, M.A. Reniers. *An algebraic semantics of basic message sequence charts.* The Computer Journal 37(4), pp. 269–277 (1994).

[6] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes. *Object-oriented development, the Fusion method*, Prentice-Hall (1994).

[7] I. Ryant. *The correctly analysed system which behaves incorrectly.* ACM SE notes, 20(2), pp. 58–61 (1995).

[8] J. Henshal, S. Shaw. *OSI explained*, Ellis Horwood Limited (1988).

[9] F.W. Vaandrager. *Two simple protocols*, in: J.C.M. Baeten, Applications of process algebra, Cambridge University Press, Cambridge tracts in theoretical computer science 17 (1990).

[10] J.C.M. Baeten and S. Mauw. *Delayed choice: an operator for joining Message Sequence Charts*, in: Formal Description Techniques VII, D. Hogrefe and S. Leue (Eds.), Chapman & Hall (1995).

[11] ITU-TS. *Recommendation Z.100: SDL*, ITU-TS, Geneva (1994).

[12] R.L. Graham, D.E. Knuth, O. Patashnik. *Concrete mathematics*, Addison-Wesley (1990).

[13] G.J. Holzmann. *Design and validation of computer protocols.* Prentice Hall International (1991).

[14] Øystein Haugen. *MSC structural concepts.* Discussion paper, March 1994.

[15] S. Mauw, M.A. Reniers. *Empty Interworkings and refinement, semantics of interworkings revised.* Proceedings of ACP95, Eindhoven (1995), also as Computing Science Report 95/12, Eindhoven University of Technology (1995).

Computing Science Reports | **Department of Mathematics and Computing Science**
Eindhoven University of Technology

In this series appeared:

93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness, p. 24
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Komatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\!\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.

94/39	A. Blokhuis T. Kloks	On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet J.J. Vereijken	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse M. Bijsterveld	Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma R. Gerth W. Janssen S. Katz M. Poel C. Rump J. Davies S. Graf B. Jonsson G. Lowe A. Pnueli J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo F. Kamareddine R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt	On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group	Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten J.A. Bergstra	Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers	A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks D. Kratsch H. Müller	Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek R. Kuiper	Traces and Logic, p. 81
94/53	R. Gerth R. Kuiper D. Peled W. Penczek	A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien	The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem R. Bol J.F. Groote	Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten C. Verhoef	Concrete process algebra, p. 134.
95/04	J. Hidders	An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi	A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen M.G.A. Verhoeven H.M.M. ten Eikelder E.H.L. Aarts	A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn O.S. van Roosmalen	Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo	Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten J.A. Bergstra	Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse R. Verhoeven O. Weber	Math∫pad: A System for On-Line Prepararation of Mathematical Documents, p. 15

//.