SICS Technical Report
T91:05

ISRN : SICS-T—91/05-SE
ISSN : 1100-3154

CCS as a Method of Specification and Verification: Analysis of a Case Study

by

Patrik Ernberg

May 1991

Swedish Institute of Computer Science
Box 1263, S-164 29 KISTA, SWEDEN

CCS as a Method of Specification and Verification: Analysis of a Case Study

Patrik Ernberg *
Swedish Institute of Computer Science (SICS)
Box 1263, S-164 28 Kista, Sweden

May 6, 1991

Abstract

We present a method of specification and verification based on CCS and observation equivalence. The method is applied to an ISDN layer 3 access protocol and an automatic tool is used to verify the correctness of the specification. Finally, we evaluate the method and tools used in the case study.

*Authors' email: pernberg@sics.se

Contents

1 Introduction

This paper reports on a project conducted in cooperation with Ellemtel Utvecklings AB[1] the autumn of 1988. The aim of the project was to determine the applicability of a particular formal description technique(FDT), and a specific method of specification and verification on a realistic example within the field of communications and switching.

Previously, Ellemtel had mostly made use of informal methods of specification and verification when developing new products. Clearly, such methods suffer from a number of drawbacks. Firstly, the specifications are usually subject to (mis)interpretations and secondly, informal specifications do not lend themselves to formal verification. One advantage of formal verification is that one can reason about a specification before actually implementing it and thus detect errors early in the design phase of a product when the cost of correcting errors is cheaper.

A natural course of action to remedy the deficiencies of informal methods of specification is therefore to adopt formal description techniques. Currently, a number of different FDT:s exist. For the purpose of this paper we have concentrated on FDT:s which are based on process algebras. The process algebra CCS (Calculus of Communicating Systems) [Mil89a] was chosen as the specification language, and observation equivalence was chosen as the method of verification. In essence, the method involves specifying a service specification and a protocol specification and proving that the two are observation equivalent. The proof was done using the Concurrency Workbench(CWB) [CPS89, CPS88], a prototype tool for automatic verification.

The ISDN (Integrated Services Digital Network) user-network interface layer 3 protocol (I.451)[CCI84c] was chosen as a realistic example. One reason for choosing this example was that Ellemtel had specified their own version of this protocol modified to be compatible with their own products [Ell87, Ell88]. Furthermore, this specification had been implemented to create an ISDN terminal (ANTON). Related work has later been conducted at LAAS, Toulouse, where I.451 was specified using petri nets [Fau89b, Fau89a].

CCS has previously only been applied to small examples such as the alternating bit and the CSMA/CD protocols [Mil89a, Par88]. In retrospect, choosing CCS as a method of specification for a larger example was a rather naive approach as CCS was not designed with specification in mind; CCS contains a very limited amount of operators and structural aids. A more appropriate FDT could for example have been LOTOS [BB89b] which is based on a process algebra but which has considerably more structural aids. A specification of I.451 in LOTOS is currently being done within the scope of the SIGMA [SIG89] project. However, a number of issues and problems remain regardless of which FDT is chosen. We therefore feel that this paper still contributes a great deal despite the fact that CCS may have been an unsuitable choice at the start of the project.

The contents of this paper are largely based on previous papers [BEH89, BEH90] delivered within the project. In Section 2 we present the method of specification and verification as well as the tool which was used. In Section 3 we sketch the specification and verification of the I.451 protocol. We then analyze the method of specification and verification and evaluate the functionality and efficiency of the Concurrency Workbench

[1] Ellemtel is a subsidiary of the Swedish Telecom (Televerket) and Ericsson and is mainly concerned with research and development of communication and switching equipment.

in section 4. Lastly, we give some concluding remarks and directions for further research.

2 Preliminaries

2.1 CCS

CCS (a Calculus of Communicating Systems) is a process algebra developed at the University of Edinburgh. A process algebra is an algebra composed of communicating processes. CCS can be regarded as an integration of a finite state machine approach into an algebraic framework. A finite state machine is a machine with a finite state space. The machine changes states through atomic events or transitions. It is a trivial task to translate finite state machines into CCS terms. CCS is explained in detail in [Mil89a] and a tutorial on CCS is available [Wal89]. We will nevertheless explain the basic concepts of CCS which are applied when specifying and verifying protocols.

CCS is a linear textual language which is made up of agent expressions. An *agent* can communicate with other agents through input or output *ports*. To distinguish between the two, output ports are generally labelled with an overbar. Complementary output and input ports will have the same name with the distinction that the output port has an overbar.

The simplest CCS-agent is one which can not perform any actions. We call this agent Nil. From Nil we can construct more interesting agents.

2.1.1 Operators

CCS has a number of operators. Each of the operators describes a means of forming a new agent from existing agents producing more complex processes.

Action-prefixing: Action-prefixing is denoted by '.' and is used to model sequential events. Actions are written in small letters while agents are written with a capital letter. A telephone which rings once and then does nothing can be described with the following CCS agent:

$$ring.Nil$$

The meaning here is that the process described by the agent $ring.Nil$ is initially capable of performing the action $ring$ and thereafter behaving as the process described by Nil.

Recursion: We can define recursive agents using definitions of the form

$$Agent = expression$$

where $expression$ may contain $Agent$. Thus, we can model a continuously ringing telephone as follows:

$$Telephone = ring.Telephone$$

Summation: The summation operator '+' is used to denote non-deterministic choice between two processes. The agent $P + Q$ therefore describes a process which can either behave as the process represented by the agent P or behave as the process represented by the agent Q. For example, a telephone which can either engage in a *call* or a *ring* followed by doing nothing could be modeled as follows:

$$Telephone = ring.Nil + call.Nil$$

Composition: The composition operator '|' is used to describe the parallel composition of processes. If P and Q are agents then $P|Q$ is an agent which represents the parallel composition of P and Q in such a way that each of P and Q may proceed independently of the other or communicate with each other. If we have the agents $A = a.A'$ and $B = b.B'$, the composition $A|B$ allows the system to perform either of the following:

- The action a followed by the composition $A'|B$ i.e. $a.(A'|B)$
- The action b followed by the composition $A|B'$ i.e. $b.(A|B')$

If two agents have complementary ports, the same rules which have informally been defined above can be used. If we have the agents $A = a.A'$ and $B = \overline{a}.B'$, the composition $A|B$ allows the system to perform either of the following:

- The action a followed by the composition $A'|B$ i.e. $a.(A'|B)$
- The action $\overline{a}$ followed by the composition $A|B'$ i.e. $\overline{a}.(A|B')$

Both the above actions do not account for any communication between A and B. Both merely represent the case where one process performs an action independently without disturbing the other process. To introduce communication between the processes, we need a third rule representing the internal communication between A and B. If we regard a as input port and $\overline{a}$ as an output port there should be a rule where both A and B change state simultaneously due to the fact that a and $\overline{a}$ represent complementary actions. If we follow the informal notation from above, this rule should be something like this:

- The composition $A|B$ allows the system to perform the action ? followed by the composition $A'|B'$ i.e. $?.(A'|B')$.

The action ? represents an internal communication action by the composite agent $A|B$. Furthermore, this internal action may arise from any pair $(b, \overline{b})$ of complementary actions by the components of a composite agent. It is therefore sufficient to introduce a single silent action, which we denote by the Greek letter τ to represent all communication handshakes. Thus, in our example, the third action which is possible in our composite agent $(A|B)$ is:

- The action τ followed by the composition $A'|B'$ i.e. $\tau.(A'|B')$

The idea is that all internal communication of a system is unobservable for an external observer. Consequently, all internal actions can be abstracted away and represented as an unobservable τ-action.

Restriction: The restriction operator '\' enables us to restrict an agent's behavior by not allowing it to communicate through ports which have been restricted. Given an agent P and a label l, the agent $P\backslash\{l\}$, P restricted on l, is an agent which behaves like P, except that $P\backslash\{l\}$ is unable to communicate through either of the ports l or $\overline{l}$, whereas P may have that capability. Restriction becomes particularly important when considering the communication between two processes i.e. parallel composition of two processes under restriction.

If we impose restriction on port a from our previous example, the two first rules do not apply, since the actions a and $\overline{a}$ can not be performed on their own. Consequently, the only action which can be performed is the internal τ-action. This is in many ways the key to our protocol verification work.

2.1.2 Graphical Syntax

As previously mentioned, CCS is a textual linear language. However, the actual specification work may be more understandable using graphical representations of finite state machines. The translation from these machines to CCS code is trivial. Two kinds of graphs are used to model CCS-agents: *flow graphs* and *transition graphs.*

A flow graph describes how agents are composed with each other. Each node represents an agent. Ports of an agent are represented as dots with a label attached to them. If two agents synchronize on a given port an arc is drawn between the two ports. For the two ports to synchronize the ports must be complementary, i.e. the labels must be an input and an output port with the same name. For technical reasons, we let output port names be underscored. An arrowhead is attached to the arc from the output port to the input port. As the specifications presented in this paper are very sketchy, arcs between nodes will often represent a number of ports that agents synchronize over. We refer to [Mil89b] for a more accurate description of flow graphs. In Figure 1 agents A and B synchronize on the port l. The arc runs from A to B implying that interpretation that A outputs an l to B. Ports which are not attached to other ports are assumed to be external and are available to the environment. In figure 1 port m is an external port.

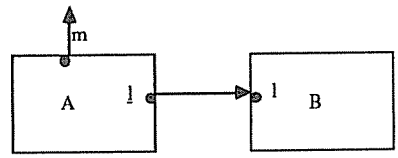

Figure 1: A flow graph

In transition graphs states are represented as circles while transitions are described as arrows. The . and + operators are easily described graphically (see fig. 2). Recursion is implemented by directing an arrow back to an earlier state in the graph.

2.1.3 Expansion Law

Earlier, we described the actions which could be performed as a result of restricted composition. In fact, we were implicitly making use of the expansion law which formally

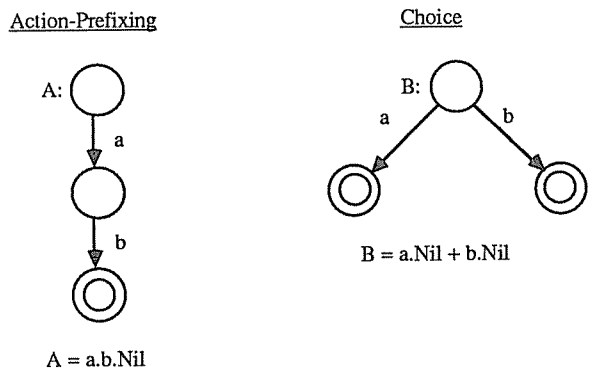

Figure 2: Action prefixing and choice described graphically

states which actions are possible if we have a composite, possibly restricted system. Actions which can be performed are of two types:

- A non-restricted action can be carried out by one process independently without affecting other processes which are part of the composition.
- An internal action can be carried out as a result of a communication between two processes resulting in a τ - action.

Figure 3: The agents A and B

If we apply the expansion law to the composite agent $(A|B)\backslash\{c\}$, where $A = a.c.A$ and $B = \overline{c}.b.B$ (see fig. 3), we get when using the expansion law:

$$\begin{aligned}(A|B)\backslash\{c\} &= a.(c.A|\overline{c}.b.B)\backslash\{c\} \\ &= a.\tau.(A|b.B)\backslash\{c\} \\ &= a.\tau.(b.(A|B)\backslash\{c\} + a.(c.A|b.B)\backslash\{c\}) \\ &= a.\tau.(b.(A|B)\backslash\{c\} + a.b.(c.A|\overline{c}.b.B)\backslash\{c\})\end{aligned}$$

If we substitute $S = (A|B)\backslash\{c\}$ we have

$$\begin{aligned}S &= a.S' \\ S' &= \tau.(b.S + a.S'') \\ S'' &= b.S'\end{aligned}$$

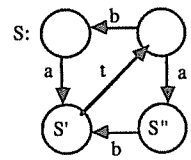

Figure 4: The composite agent obtained using the expansion law

Thus, the intention of the expansion is to produce an interleaving of all actions which can occur in a composite system in terms of only the action prefix and choice operators.

2.2 Observation Equivalence

Several different equivalences exist in the literature. We will initially concentrate on *observation* equivalence which will be used when verifying the I.451 protocol.

Observation equivalence between two agents is proved by the existence of a *bisimulation* relation between the states of the agents. Let l denote a sequence of actions including the internal τ-actions. Then $\hat{l}$ is the sequence gained by deleting all occurences of τ from l. For example, if $l = \tau^n.a.\tau^m$, where a is an observable action and $n, m \geq 0$, then $\hat{l} = a$.

We also introduce the notation $P \xrightarrow{a} P'$, which denotes that the agent P can perform the action a, and reach the agent P'. Furthermore, given the agents P and P', we write $P \stackrel{a}{\Longrightarrow} P'$ if P can be transformed into P' performing the action a preceded and succeeded by any finite number (possibly zero) of τ-actions.

There exists a bisimulation relation $\mathcal{R}$ between two agents P and Q if, for all a,

1. Whenever $P \xrightarrow{a} P'$ then, $\exists Q', Q \stackrel{\hat{a}}{\Longrightarrow} Q'$ and $(P', Q') \epsilon \mathcal{R}$

2. Whenever $Q \xrightarrow{a} Q'$ then, $\exists P', P \stackrel{\hat{a}}{\Longrightarrow} P'$ and $(P', Q') \epsilon \mathcal{R}$

Intuitively, this means that each transition of P can be mimicked by a transition of Q and vice versa. Hence, P and Q can never be distinguished by an external observer.

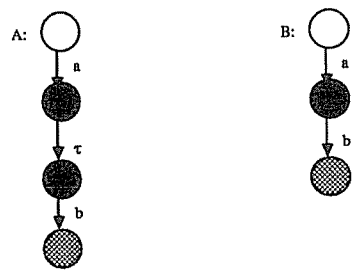

Figure 5: Two observation equivalent agents. The states which bisimulate each other are shaded in the same way.

In figure 5, agents A and B can mimic each other and are therfore observation equivalent, written $A \approx B$. The same cannot be said for agents A and B in figure 6. We observe that B cannot imitate A if both have done an a action. If B chooses the left branch in its action tree, it cannot mimic A if A chooses to perform a c action.

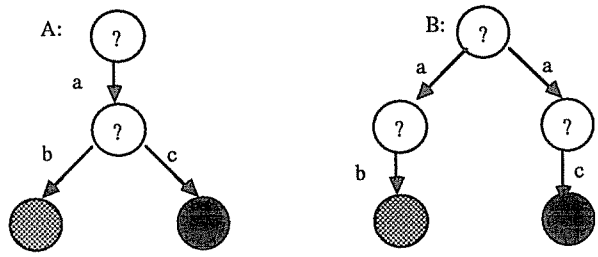

Figure 6: Two agents which are not observation equivalent

Similarly, the maybe more intuitive notion that all τ-actions can simply be 'dropped' when considering observation equivalence is wrong. Consider the two agents in figure 7. These two are not observation equivalent.

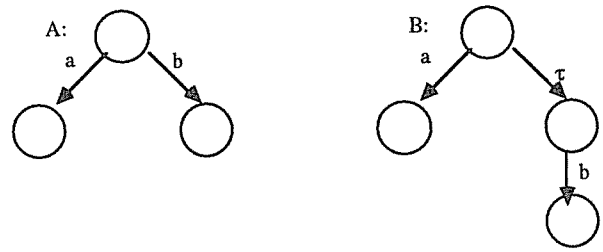

Figure 7: Two agents which are not observation equivalent

This is maybe better understood if we use an analogy. Let the two agents A and B represent two black boxes, each with two buttons a and b which can be pressed at any time. Pressing a button corresponds to performing an observable action. In the above example, box A allows any button a or b to be pressed at any time. Box B, on the other hand, may not allow the a button to be pressed if an internal τ-action has been performed. Consequently, the two boxes are not observation equivalent.

2.3 Specification and Verification of a Protocol using CCS

The OSI model is a layered model where each layer is assigned a particular task. To relate the different layers, a service concept is used. A layer provides a service to the the layer above. The basic idea behind layering is that of abstraction. By using the service concept, the service-user abstracts away the details of the service provider. It is of no interest for the service-user to know how the service is implemented as long as the required service is provided. The service-user executes service primitives at Service Access Points (SAPs). The execution of one service primitive at one SAP may lead to the execution of another service primitive at another SAP.

The service is implemented by the service-provider using a protocol. The protocol is a set of rules which defines how protocol entities should communicate with each other in order to provide the expected service. The protocol entities communicate using messages known as Protocol Data Units (PDUs). The protocol entities can interact with the above layer $N+1$ and the underlying layer $N-1$ through SAPs (see fig 8). Note that the PDUs

are not exchanged directly between protocol entities. In reality, PDUs are exchanged by $N-1$ service primitives.

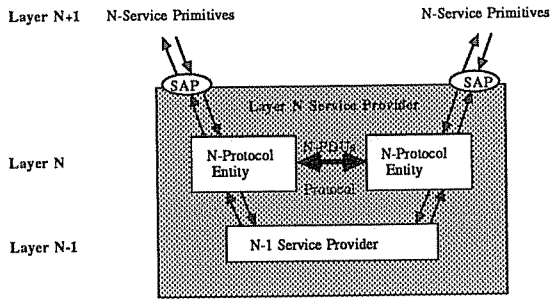

Figure 8: The OSI Service concept

When verifying the protocol using CCS, the basic idea is to prove that the protocol specification provides a service which is observation equivalent with the service specification. The protocol specification is a composition of the two layer N protocol entities in parallel with the $N-1$ layer service, restricted on the $N-1$ service primitives. In short, the methodology for verification of protocols in CCS is (taken from [Par88]):

1. Define a set of actions L_N where each action corresponds to a layer N service primitive. Similarly, define L_{N-1} as the set of layer $N-1$ service primitives. Define the following CCS agents:
   - SS_N The service specification of layer N
   - SS_{N-1} The service specification of layer $N-1$
   - PE_1, PE_2 An agent for each protocol entity
2. Prove in CCS that
$$SS_N \approx (PE_1 | SS_{N-1} | PE_2) \backslash L_{N-1}$$

2.4 CWB

The complexity of the calculations which have to be performed when applying the expansion law and establishing equivalences increases polynomially[Par87] with the size of a system. Manual calculations are therefore only possible for extremely small examples. In order to analyze larger systems, a prototype tool has been developed called The Concurrency Workbench (CWB).

The CWB allows us to analyze CCS agents in a number of different ways. Firstly, it is possible to establish if two given agents are equivalent with respect to several different equivalence criteria. Observation equivalence can, for example, be established between two agents.

Secondly, a number of preorders are defined which correspond to the equivalences defined in the CWB. They allow us to analyze agents which are not equivalent. It is for instance possible to prove that an agent can provide a service which is a subset of another service.

The third method of analysis is based on Hennessey-Milner logic. It is possible to define propositions in a powerful modal language which incorporates both simple propositional connectives, modal operators, and maximum and minimum fixed point operators. A CCS expression can be analyzed to establish if it satisfies a specification formulated in this modal language.

In this paper, we will concentrate our use of the CWB to the equivalence analysis and more specifically to observation equivalence. One command which was used particularly often is *mi(minimize)*. This command computes the agent with the smallest state space that is observation equivalent with a supplied agent. Another command which was frequently used was the *fd(find deadlock)* command which finds deadlocks in an agent.

Operating instructions for the CWB [CPS88] as well as a description of the CWB's architecture [CPS89] are available and will not be discussed in this paper.

3 Applying the Method to I.451

In this section we will briefly present the I.451 protocol and where it fits into the ISDN environment. We will then describe how the service and protocol were defined using the method described in the precious section. Since our specifications are rather incomplete and in many ways incorrect we will not include them as appendices.

3.1 ISDN and the I.451 protocol

This paper concentrates on aspects concerning the actual method of specification and verification. Consequently the details of the ISDN system and in particular I.451 will not be included in this paper. A number of books have been published on ISDN and a simple overview can be found in [Sta88].

3.1.1 Integrated Services Digital Network (ISDN)

In essence, ISDN provides a number of services on top of a digital telephone network. Services which are to be provided range from digital telephony to a variety of other data traffic. The key to ISDN is the small marginal cost for offering data services on the digital telephone network, with no cost or performance penalty for voice services already carried on the integrated digital network (IDN).

The end-user has access the ISDN by means of a local interface which is connected to the ISDN via a "digital pipe". The traffic on the pipe may be composed of a variety of different data up to the pipe's capacity limit. Thus a user may access circuit-switched and packet-switched services, as well as other services, in a dynamic mix of signal types and bit rates. To provide these services, ISDN requires rather complex control signals to instruct it how to sort out the time-multiplexed data and provide the required services. These control signals will also be multiplexed onto the same digital pipe (see fig. 9).

Physically, the digital pipe will carry a number of different communication channels, depending on the capacity of the pipe and the needs of the user. The transmission structure of each access link will be constructed from, among others, the following types of channels:

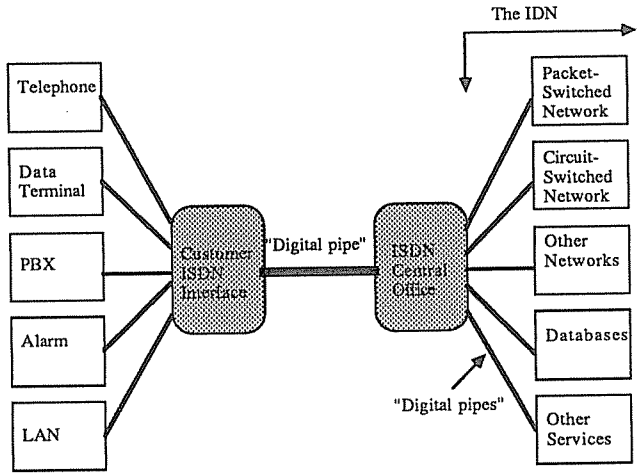

Figure 9: A conceptual view of ISDN from a user point of view

- B channel (64 kbps) : This is a basic user channel and can be used to carry digital data, PCM(Pulse Cose Modulation)-encoded digital voice, or a mixture of lower-rate traffic, including data and digitized voice encoded at a fraction of 64 kbps.
- D channel (16 or 64 Kbps): This is a control channel. It carries signalling information to control circuit-switched calls on associated B-channels at the user interface. When the channel is not used for signalling, it can be used for lower speed data transmissions
- H channel (384, 1536, and 1920 kbps) : This is a channel for high bit rate transmission. It may be used as a high speed trunk or subdivided to cater for a number of different functions with TDM(Time-division multiplexing). Applications which may need high bit rates are fast facsimile, high-quality audio and video.

The above channels are grouped into transmission structures that are offered as a package to the user. The two packages which are best defined and most widely implemented are basic and primary access.

Basic access consists of two full-duplex 64 kbps B-channels and one full-duplex 16 kbps D-channel. With framing, synchronization, and other overhead bits, the total bit rate for basic access is 192 kbps. This type of access is intended to meet the needs of most individual users and very small offices. It would allow the simultaneous use of applications such as packet-switched access, link to a central alarm service, facsimile, and videotex. These services could be accessed from a single multifunction terminal or from several separate terminals. In either case, only a single physical interface is needed.

Primary access is intended for users with greater capacity requirements, such as offices with a local PBX(Private Branch Exchange) or LAN(Local Area Network). Because of differences in digital transmission in different countries, the primary access structure varies. Typically, primary access consists of either 23 or 30 B-channels plus one 64 kbps D-channel.

3.1.2 I451: The layer 3 User-Network Interface Protocol

In CCITT recommendations the digital pipe is referred to as the User-Network interface. The user-network interface comprises the three lower layers of the OSI model: the physical, link, and network layers. Protocols have been developed for each layer. I.451 is CCITT's recommendation for the layer 3 user-network interface protocol [CCI84c]. The protocol is divided into a user-entity and a network-entity which interact using PDUs. The two entities interact with the link layer using link-service primitives and the link layer [CCI84a] is implemented using a LAPD protocol [CCI84b] (see fig. 10).

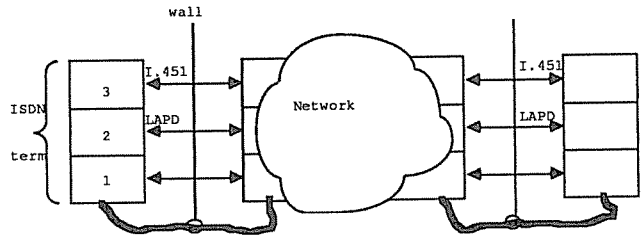

Figure 10: Two User-Network interfaces

The user-entity interacts with layers above the network layer using network-service primitives. Since I.451 is merely an access protocol only one side of a global network service can be represented using one I.451 protocol. To describe the global network service with end-to-end connectivity, two access interfaces have to be defined, where one interface acts as an originating exchange and the other as a destination exchange.

The recommendation specifies the procedures for the establishing, maintaining, and clearing of network connections at the ISDN user-network interface. The procedures are defined in terms of messages exchanged over the D-channel of basic and primary rate interface structures.The messages carry a number of parameters. In the 1984 version of the recommendation, procedures for the control of circuit-switched connections, user-to-user signalling connections, and packet-switched connections were defined. The recommendation has also been described using SDL diagrams. The details of I.451 are described in CCITT's recommendation [CCI84c].

Ellemtel's version of I.451 is a subset of CCITT's version with certain modifications needed to adapt the recommendation to their products . The specification is only described in human language and the two protocol entities have been treated as separate documents. The details can be found in Ellemtel documents [Ell87, Ell88].

We will not describe I.451 in any more detail in this paper. We refer to [SIG90, Fau89b, Fau89a, CCI84a, CCI84b, CCI84c, Ell87, Ell88] for a more thorough description of the protocol.

3.2 Service Specification of I.451

When specifying the service of our protocol we specified two different scenarios. In both cases we specified only the service of the actual protocol and *not* the end to end service between two ISDN end users. Both services were specified using the method described in Section 2.3 i.e. a set of actions were defined where each action corresponded to a service primitive. The difference in the two scenarios lies in what we define as service primitives.

In reality, the network side is connected to the rest of the ISDN network. Consequently, it is in many ways wrong to speak of a network-side service since the true service is the end to end connection between two users. We have nevertheless introduced pseudo-service primitives on the network side in our first specification which correspond to signals which are sent into the network. All service primitives have the form $X\_name\_primitive$, where

$$\begin{aligned} X &= U \text{ or } N \text{ depending on which entity side the primitive is employed} \\ name &= \text{the name of the primitive} \\ primitive &= req \text{ or } ind \text{ depending on whether it is a request or indication primitive.} \end{aligned}$$

3.2.1 Scenario 1: Side to Side Service

The service specification encompasses both the user and network side i.e. the network side primitives are regarded as service primitives. It was nevertheless very difficult to intuitively establish what the global service was supposed to look like. We therefore chose to divide the service into two entities, a user and a network side, which we then combined using parallel composition restricted on some synchronization signals. The synchronization signals were used to guarantee a correct sequence of service primitives:

$$Service = (User\_Service | Network\_Service) \backslash \{Synchronization\ Signals\}$$

3.2.2 Scenario 2: One-Sided Service

The specification of a service according to scenario 1 turned out to be a somewhat artificial method of specification since the synchronization signals used to specify the service corresponded very closely to PDU:s sent between the two protocol entities. We were therefore constantly forced to glance at the protocol in order to establish a correct service which would prove to be equivalent with the protocol specification.

In our second scenario we avoided the problems by only specifying the local user service, i.e. we only allowed User Service Primitives to be observable from the environment.

3.3 Protocol Specification of I.451

We have chosen to specify only a subset of the entire I.451 protocol. Connection establishment and call release requested by either the network or the user have been specified. As a first simplification no attention has been paid to the underlying layers and no link layer service primitives are used by the protocol i.e. we assume an empty medium. This implies that connection establishment of a link as well as coding of network layer primitives into link data are not considered. In other words, we assume a link service which is always willing to establish a connection and which provides reliable data transmission. This simplification also means that signal collisions are excluded due to the fact that no buffers exist between the two protocol entities (see fig. 11).

The protocol was first specified graphically. Service primitives which are offered to environment are written as in the previous section and PDU's have been given the same

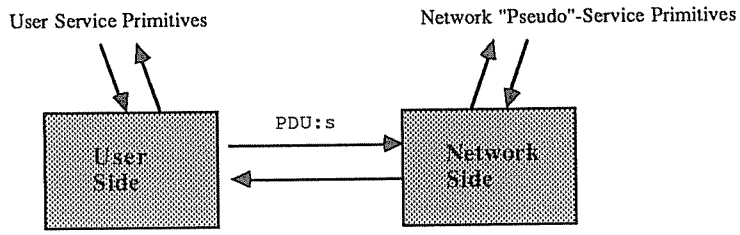

Figure 11: The protocol specification

names as those in [Ell87, Ell88] but a u or an n is appended to each PDU to distinguish which entity has sent the signal.

Timers have been modeled in two different ways in our protocol specification:

1. Only time-out signals are modeled. These are modeled as internal τ -actions.
2. All timer signals for a given timer are modeled as separate signals in a given protocol entity. The entity is then combined with a generic timer using parallel composition (see fig. 12:

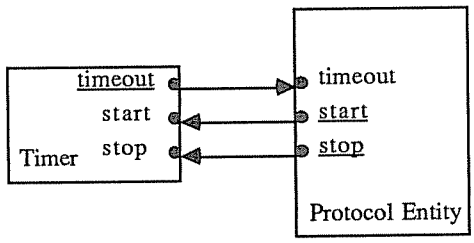

Figure 12: Specifying a Timer

$$New\_Protocol = (Protocol\_entity | Timer) \backslash \{timer\ signals\}$$

where

$$\begin{aligned} Timer &= start.Timer\_1, \\ Timer\_1 &= start.Timer\_1 + stop.Timer + \overline{timeout}.Timer \end{aligned}$$

When the expansion theorem is applied to the composite agent all timer signals are reduced to internal τ-actions.

3.4 Verification of I.451

When verifying the protocol, we first made use of the *fd(find deadlock)* command in the CWB to make sure that the protocol was free of any deadlock. Due to the inefficiency of

this command on a large state space, the *mi(minimize)* command was used to minimize the state space with respect to observation equivalence.

Once the protocol was deadlock free, observation equivalence was established, with difficulty, for the two scenarios. For the second scenario, some method of hiding the network service primitives from the environment had to be used. We chose to provide an agent which we call *Responder* which could complement all the network service primitives to produce a τ-actions (see fig. 13):

$$\begin{aligned} Responder \;=\; & n\_setup\_ind.Responder + \\ & \overline{n\_setup\_req}.Responder + \\ & \overline{n\_alert\_req}.Responder + \\ & n\_alert\_ind.Responder.... \end{aligned}$$

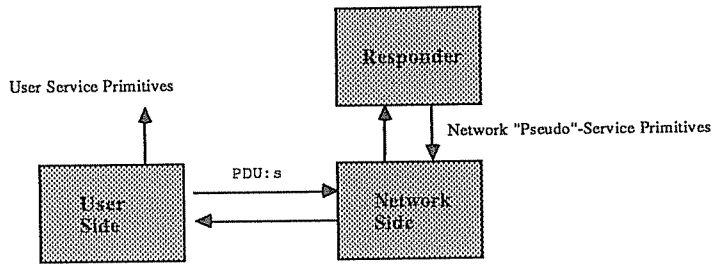

Figure 13: The protocol with attached responder

The protocol is combined with the responder using parallel composition restricted on the network pseudo-service primitives:

$$System = (Protocol|Responder)\backslash\{Network\ pseudo - service\ primitives\}$$

Due to complexity reasons and the poor efficiency of the Concurrency Workbench, the protocol was not specified using an underlying medium. Some work was done using a two one-place buffers in each direction on the call-release phase (see fig. 14) but only limited experiments were conducted.

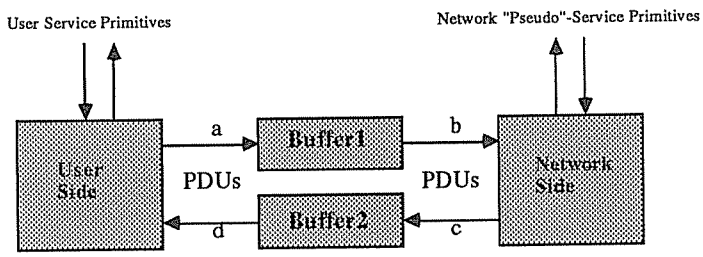

Figure 14: Introduction of a buffer between protocol entities

Since the Concurrency Workbench cannot handle CCS with value-passing, a somewhat primitive buffer had to be modeled which accepted all possible PDU:s at one end and then sent them out again at the other end. Signals also had to be renamed slightly to make sure that all communication was done through buffers instead of directly between the two protocol entities:

$$\begin{aligned} Buffer \;=\; & disconnect\_n.disconnect\_nb.Buffer + \\ & release\_n.release\_nb.Buffer + \\ & release\_complete\_n.release\_complete\_nb.Buffer..... \end{aligned}$$

In CCS, the protocol specification looked like this:

$$Protocol = (User\_Side|Buffer1|Buffer2|Network\_side)\backslash\{a, b, c, d\}$$

where a,b,c,d are PDU's.

4 Analysis of Method and Tools

We have chosen to divide our analysis into three parts. First we anlayse the method of specification, dealing mainly with the problems of specifying the actual protocol. In the second part, we analyse the method of verification, also touching on difficulties that we had when specifying a service. Lastly, we analyse the Concurrency Workbench in more detail.

Ideally, specification should be performed without having to pay attention to the method of verification and tools that are adopted. In practice, however, tools have only reached a maturity where they allow a specific specification language and, furthermore, the method of verification that is used will often influence the method of specification. This was particularly true in our example where observation equivalence turned out to be a very strong equivalence which was hard to establish without being very careful in the specification phase. It is therefore difficult and somewhat artificial to divide our analysis into three parts as they are all closely interrelated. However, we feel that this classification adds some structure to our analysis which would otherwise have been lacking.

4.1 Analysis of the Method of Specification

When specifying large protocols, structural aids and syntactic shorthand are of considerable importance. CCS contains no such aids and was clearly not designed with specification in mind. Looking back, the approach that was taken in this case study was rather naive. Languages more suitable for specification with structural aids and which can handle abstract datatypes exist. Nevertheless we feel that this work still raises a lot of specification issues related to the synchronization properties of a communication protocol, which exist regardless of the structural aids that can be incorporated in a process algebra. We have therefore divided this section into two parts, one which essentially deals with problems related to CCS in particular, and one concerned with more general specification problems which arise when specifying communication protocols in any process algebra.

4.1.1 Problems with CCS

The Parallel Composition Operator: The nature of the CCS parallel composition operator leads to certain specification problems. Firstly, communication in CCS is binary, which means that only two processes can communicate at a given time. Sometimes, it would be advantageous to be able to synchronize more than two processes. Secondly, all communications lead to an internal τ-event. In certain cases, one would like processes to synchronize on a given action but keep the action visible to the environment. This should be compared with the LOTOS [BB89b] parallel composition operator which supports both the above features i.e. explicit hiding and multi-way synchronization.

Structuring Problems: As was previously mentioned, CCS is a very restricted specification language. One way to incorporate more structural aids into the specification is to introduce new operators, which in many cases can be expressed in terms of the simple operators which already exist. Some interesting operators which could improve the structure of a specification are:

- A disrupt operator: A process can be disrupted in any state by a process Q and then continue as defined by Q. In the case of I.451 it could be interesting to specify a *Disconnect* process which disrupts the *Connection_establishment* and *Active* processes.
- A sequential composition operator: A process P terminates and is followed by a process Q.
- An $if - then - else$ operator: Could for example check what value is sent on a port and then perform the appropriate action. In basic CCS, all actions which should not be accepted by the protocol (i.e. simply swallowed) have to be explicitly modeled.

Many interesting operators can be constructed from the basic CCS-operators. In [Mil89a], for example, an imperative language is defined which can be translated into basic CCS-operators. A tool has also been developed, Lunsen [FJP90], which implements this translational semantics and produces CCS-code which can be fed into the Concurrency Workbench for further analysis.

Although what operators are most appropriate for the specification of communication protocols remains an open research question, a more realistic language to use could have been LOTOS [BB89b], which contains a richer set of operators. There are also a number of LOTOS tools available with similar functionality to the one present in the Concurrency Workbench [QPF89, Gar89, Eij89, Tre89]

Data Types: Closely related to the structuring problems mentioned above, is the lack of data types in CCS. Synchronization properties of a protocol can be studied in CCS but clearly a large part of protocol specification is concerned with specification of data types. A tool which is to be used in industry must definitely support abstract data types. Once again, LOTOS seems like a promising candidate and tools which can cope with abstract data types for LOTOS are beginning to emerge. Lunsen also supports the boolean type and restricted types which guarantee a finite state space.

4.1.2 Problems related to Protocol Specification in General

Understanding the Protocol: One of the major problems of specifying an existing protocol is of course that it takes a lot of effort to understand the specification enough to be able to translate it into CCS. The I.451 1984 standard is specified informally in written language, supported by SDL diagrams. However, if there is conflict between the two specifications, the informal text should be chosen as the correct one. Even when one has understood the specification well, simplifications which are made in some part of the specification (particularly for efficiency reasons during verification), may lead to "overspecification" or *redundancy in context* in another part of the specification.

Redundancy in Context: As previously mentioned, redundancy in context has to do with "overspecifying" a protocol. To take a concrete I.451 example, the User side may be in the *Outgoing_Call_Proceeding* state. In this state, a *connect* or an *alerting* signal may be received. When the network side is in the *Outgoing_Call_Proceeding* state, it can receive either a *connect* or an *alerting* signal from the rest of the network which it forwards to the user side. However, we may decide, for efficiency reasons, to only specify a reception of a *connect* signal on the user side:

$$\begin{aligned} U\_Outgoing\_Call\_Proceeding &= connect.U\_Active \\ N\_Outgoing\_Call\_Proceeding &= \overline{connect}.N\_Active + \overline{alerting}.N\_Active \end{aligned}$$

The *alerting* signal on the network side would then be redundant in this context. However, when we place these two entities in parallel composition, restricted on the signals *connect* and *alerting*, and assume an empty medium, the resulting agent will be:

$$Composition = \tau.(U_Active|N_Active)$$

The handshaking mechanism in CCS avoids a deadlock situation and the protocol appears to function the way it should. This form of redundancy in context is not especially dangerous but it does add confusion when trying to understand the specification. Some work has been done on detecting redundancy in context using proof techniques in [Vaa90]. However, this method to our knowledge only produces an agent which is trace equivalent with the original agent.

Modeling the Underlying Service: Similar redundancies in context can also be detected in the protocol specification depending on in what detail the underlying service is modeled. In the *Call Release* phase for example, the protocol can in a worst case scenario have 4 pending signals buffered in the underlying medium. The two entities could perform the following traces:

- User Entity: $\langle disconnect\_u, timeout, release\_u \rangle$
- Network Entity: $\langle disconnect\_n, timeout, release\_n \rangle$

In a specification where an empty medium is assumed these traces could not be performed as the CCS parallel composition operator forces a communication between the

two entities, i.e. the specification is redundant in the empty medium context. Of course, this can be avoided with care and a good understanding of the protocol being specified but some kind of help would be valuable here. Simple reachability analysis could be a possible solution.

In our example, no serious attempt was done to specify the underlying service correctly. A one place buffer was introduced to analyze parts of the call release phase but we were hampered by the inefficiency of the Concurrency Workbench. Naturally, a complete specification of the protocol should include a buffered medium.

When trying to specify the buffer in CCS, there also emerges the need for some kind of value passing mechanism. In our example, all signals which could be sent and received by the medium had to be explicitly modeled. Furthermore, the names of the signals had to be changed slightly to make sure that no communication could be achieved directly between protocol entities without passing through the medium:

$$\begin{aligned} Buffer &= disonnect\_n.diconnect\_nb.Buffer + \\ &\quad release\_n.release\_nb.Buffer + \\ &\quad release\_complete\_n.release\_complete\_nb.Buffer..... \end{aligned}$$

This is again a problem which is specific for basic CCS and which can be remedied by using LOTOS or CCS with value passing.

Modeling Timers: Modeling timers in CCS is a relatively tricky task as there is no notion of real time. Several attempts have been made to incorporate time but no tools dealing with this exist to our knowledge. In communication protocols, timers are almost always used when a message is sent to a peer entity and a response is desired. If the response takes too much time due to a congested medium or the loss of a message, the timer times out. Furthermore, a global timer usually exists which times out if the system is inactive for a certain amount of time. This means that virtually every state in the specification should be equipped with a timer. Two different approaches can be taken when modeling timers.

1. As a spontaneous τ-action or alternatively be modeled as a process interacting with a protocol entity; the end result remains the same:

$$New\_Protocol = (Protocol\_entity | Timer) \backslash \{timersignals\}$$

where

$$\begin{aligned} Timer &= start.Timer\_1, \\ Timer\_1 &= start.Timer\_1 + stop.Timer + timeout.Timer \end{aligned}$$

The problem with the use of such a spontaneous τ-action is that it can fire even if it is possible for the protocol to proceed with another perfectly legitimate action. This corresponds to a timer which times out too early. Thus, such a protocol

specification would also take into account unfriendly timers. The consequence of this when specifying is that signals which arrive after the timer has timed out also have to be taken into account. One problem with our specification is that we account both for the case where the lossy medium exists and the signal is lost, and for the case where the signal is delayed. If we only specify mediums which are not lossy the specification is redundant in this context.

2. As a "friendly" timer where the whole system being specified is informed of a time out. Although this is very unrealistic in practice, this can be useful as a first step in the specification when we are only interested in the "normal" behaviour of the protocol. A similar approach to this is taken in [Fau89b, Fau89a] where "normal" and "abnormal" behaviours of the protocol are specified and verified separately. Alternatively a priority operator may be introduced where timer signals are placed at a lower priority than other signals. Suggestions to how priority operators can be incorporated in the language also exist [CH88, Vaa86]. No observation equivalence based on these semantics has however yet been developed.

4.2 Analysis of the Verification Phase

In this section, we will first discuss the important issues which have to be taken into account when specifying a service. Secondly, we will discuss the problems arising form specifically using observation equivalence in our example.

4.2.1 Issues Related to the Specification of a Service

If we restrict our discussion to communication protocols a service specification can be regarded as a definition of the set of all allowed protocol specifications. When verifying a protocol, we are essentially making sure that our specification satisfies all criteria in the service specification i.e. that the protocol specification is part of the set of allowed specifications. This can be done by establishing a relation between the service and protocol specifications. In our case we have used an equivalence relation, more precisely observation equivalence.

Besides defining the set of allowed specifications, the service specification and relation between service and protocol may enjoy other properties which are important when designing a protocol specification which ultimately should lead to an implementation. Below, we list a few of these properties:

Support Abstraction/Refinement: The design process can seldom be divided into one specific specification phase and one specific verification phase. Most often, a designer will start with a very abstract specification and subsequently refine it into more and more detailed specifications. In each refinement step, verification is performed by establishing a relation between the more refined specifition and the original specification. Abstract specifications will generally be more nondeterministic than their refinements and consequently the relation between them has to account for this. This makes equivalence relations difficult to use since they generally require that the refined specification be as nondeterministic as the original specification. A more natural relation may be some kind of preorder (a preorder is a transitive and reflexive relation). Another observation is that

it seems easier to accomplish a refinement process if a specification and its refinement use the same language.

Refinement is a very broad term which allows many interpretations. A lot of work is being done within the field and all seem to have different definitions of refinement. One such refinement process is that of action refinement; an action can be refined to finer actions. One property which could be desirable in a relation which supports action refinement could be that it preserves the branching structure of processes, i.e. that all branching in an intermediate state of a computation is preserved, even if internal actions are involved.

Support Modularity: Another desired property is that verification should support modularity. By that we mean that it should be possible to specify and verify separate modules and then assume that their composition is correct without verifying the whole composition. Generally, we must require that our relation is a *congruence*, i.e. that the equivalence relation between two specifications is preserved in all algebraic contexts. A motivation for this is, of course, that it is often more efficient and easier to specify and verify partial specifications than large complex ones.

4.2.2 Problems related to verification with Observation Equivalence

Below we will dicuss the problems of observation equivalence in the light of the properties mentioned above. We will also suggest alternative methods of specification and verification which could remedy part of the problems discovered with observation equivalence.

Observation Equivalence is too strong: When we attempted to specify a global service for I.451, we noticed very early in the design process that such a service was hard to define. One reason for this was that observation equivalence is in many ways a very strong equivalence. Observation equivalence is not only sensitive to internal nondeterminism but it is also sensitive to where the internal nondeterminism occurs. As an example: $A = \tau.a.b.nil + \tau.a.c.nil$ is not observation equivalent to $B = a.b.nil + a.c.nil$. Our feeling is that it is very hard for a designer to specify internal nondeterminism with such accuracy. A spectrum of alternative equivalences which treat internal and external nondeterminism differently exist. The most commonly used ones are probably trace, failure [Hoa85], ready[OH84], failure trace [Phi87], ready trace [BBK87], and testing equivalence [Hen88]. The most interesting equivalence in this spectrum seems to be testing equivalence which takes internal nondeterminism into account but is not sensitive to where the internal nondeterminism occurs. The two agents A and B are for example testing equivalent.

One problem which remains regardless of the equivalence used is that it is very hard to specify a global service as was attempted for our example in the the first scenario. Alternatively, a local service can be established as was done in the second scenario. However, this second service is in many ways unsatisfactory since it does not say anything about the relation between two protocol entities. One possible remedy would be to introduce logical formulas to establish a relation between the primitives sent at one entity and the primitives received at the other entity. One such logical formalism is HML [Mil89a]. Walker has for example used HML to verify certain properties of mutual exclusion algrithms [Wal89]. One problem with using HML on CCS-specified protocols is that one cannot

reason easily about internal actions of the system since they are all regarded as equal and inseparable τ-transitions. Walker remedies this by tagging each internal transition with a "probe" action which is visible to the environment. This introduction of an "extra" transitions is in many ways unsatisfactory. An alternative approach could be to adopt the LOTOS [BB89b] parallel composition operator where entities communicating with each other must synchronize on ports with the same name but the resulting action is the port name. Expanding the LOTOS process $A = a; b; stop|[a]|a; c; stop$ would for example yield the process $a; (b; stop|[a]|c; stop)$. If abstraction is desired, the *hide* operator must be explicitely used.

A specification style which in many respects is analogous to specification using logical formulas, is constraint-oriented specification which has been successfully used to specify services in LOTOS[QA89, Man89]. The idea is to force all constraints specified as LOTOS processes to synchronize using the broadcast parallel composition operator. Usually a specification is made up of local constraints and peer-to-peer constraints. One advantage with this method of specifying a service is that it is written in the same language as the protocol; HML is in many ways unintuitive and difficult to use and requires that the specifier learns another language besides the actual specification language CCS.

Constraint-oriented specification seems promising when specifying top-level services. Other specification styles seem more appropriate as the protocol specification becomes more detailed and accurate. A lot of work is being done within the scope of the esprit LOTOSPHERE project to establish what relations are most appropriate for different phases in the design process and what transformations are appropriate between different specification styles.

Maybe a cleaner design process would be to use the same method of specification and verification throughout the whole design process. Preorders seem the most promising candidates when trying to support such a design process. Most equivalences, can be established as kernels of their respective preorders. Thus all equivalences mentioned above have their corresponding preorders. Our very limited experience seems to suggest that these preorders are too weak to be of any greater interest. Kim Larsen and Gérard Boudol av developed an interesting method of specification method based on *modal transition systems* and an accompanying refinement ordering [Lar89]. Here, specifications are described as transitions systems but transitions are also qualified as *necessary* or *admissible*. This method seems to combine the expressive advantages of specifying using simple transition systems with the advantages of logic specifications when dealing with refinement [BL89].

Observation Equivalence is too weak: Above we have given several reasons for why observation equivalence is too strong. However, there are cases where observation equivalence may turn out to be too weak. Observation equivalence is, for example, not a congruence with respect to the + operator. The agent $\tau.b.nil$ is for instance observation equivalent with $b.nil$ but $\tau.b.nil+a.nil$ is not observation equivalent with $b.nil+a.nil$. One way to remedy this is to use observation congruence [Mil89a], which is slightly stronger than the equivalence since it requires that τ-transitions leading from the root of an agent must be mimicked by τ-transitions from the root of the other agent.

Another shortcoming of observation equivalence is that it does not respect branching time. Consider the agent A in figure 15. The agent B in figure 16 is observation equivalent

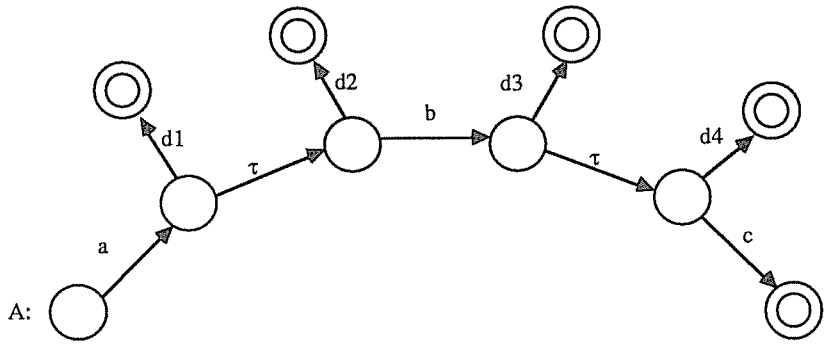

Figure 15: Agent A

with A. The transitions which are more emphasized in B are those which have been added with respect to A.

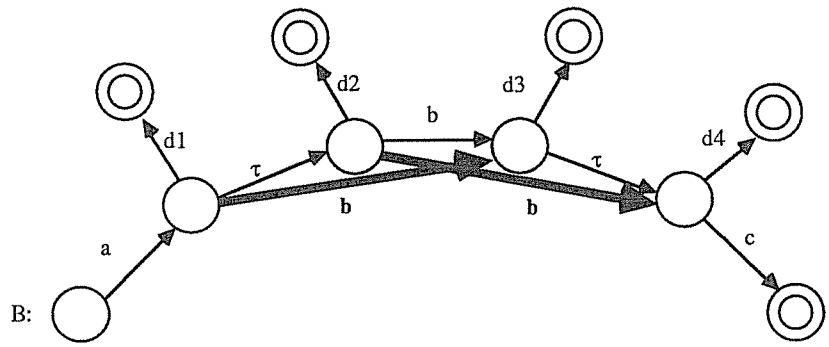

Figure 16: Agent B

Notice that a b-transition has been added which allows the agent B to choose between the transitions $d1$ and b in a state whereas the b-branch is not possible in the corresponding state of A. An alternative equivalence which respects branching has been introduced in [GW89] and a very efficient algorithm to implement the equivalence intoduced in [GV89].

4.3 Analysis of the Tool used

We will divide our analysis of the tool into two sections. First we anlyse the functionality of the Concurrency Workbench(CWB), then we comment on its efficiency.

4.3.1 Functionality of the Concurrency Workbench

Normally, the design process is a very interactive one where the designer alternatively specifies and verifies. A good tool should therefore support both the specification and verification phases and should be able to easily alternate between the two. The CWB is essentially a verification tool which does not provide any specification aids whatsoever; a CCS specification is written in a text editor and then read into the CWB. Syntactic control is done when the specification is read but error messages are fairly ambiguous.

A graphical aid has been designed which allows for specification of CCS in its graphical syntax [BB89a]. This is particularly useful for educational purposes but is somewhat cumbersome to use for the experienced specifier.

Our experience is that specifications often contain a lot of mistakes which are the result of simple typing errors. CCS-actions may for example be wrongly typed resulting in an incorrect specification which will only be discovered in the verification phase. Functions do exist which help resolve such errors and incorporating these functions in the syntactic control part of the CWB would probably help the specifier considerably.

In our example, we have only tried to prove that two CCS specifications are observation equivalent with each other. We have therefore mainly made use of the *fd(find deadlock),eq(equivalence)* and *mi(minimize)* commands. The Workbench contains a lot of other functionality which caters for different methods of verification. We will nevertheless concentrate on functionality related to verification using observation equivalence. Several concrete problems could be identified when dealing with the above-mentioned commands:

1. The minimize command only produces an agent which is minimal with respect to the number of states. The CWB produces a lot of extra transitions when it computes the transitive reflexive closure of all τ-transitions in a specification which makes it difficult to analyse the minimized specification. Other tools such as AUTO [SV89] and Aldébaran [Fer89] produce agents which have a minimal amount of transitions.

2. When two CCS-specifications are proved not to be observation equivalent, the CWB gives the error message *false*. In an interactive design process, it would be useful to have a more informative error message. One approach which has been adopted in TAV [JKZ88] is to produce a distinguishing HML formula which tells the two agents apart.

When specifying we also found it very difficult to gain understanding in how the protocol actually functioned. Often, more understanding is gained through simulation of specifications than through actual verification. Simulation tools exist and the graphical editor (GCW) [BB89a] also contains a simulation function. Our impression is that a user-friendly simulator incorporated in the actual CWB would also improve the tool considerably. A problem related to this which was also mentioned in the previous section is that CCS does not allow us to easily distinguish τ-transitions. A possible extension of the CWB could be to introduce "coloured" τ-transitions.

4.3.2 Efficiency of the Concurrency Workbench

Without an automated tool, it would have been almost impossible to perform any kind of verification. It is however important to bear in mind that the CWB is a prototype which was created essentially to evaluate different theoretical results and methods of analysis on finite state transition systems. Of course, the CWB can be used to demonstrate the applicability of verification methods on small examples but it was not designed with efficiency in mind. When dealing with our second scenario, it took approximately one hour to prove that the protocol with responder was equivalent to the specified service. Experiments with buffers betwen the two protocol entities could only be conducted on part of the protocol for efficiency reasons.

Recently, work has been done to identify the bottlenecks in the *minimize* command [EF90] and improvements have already been made [Dah90]. Nevertheless, the CWB's performance remains largely inferior to tools such as Aldébaran and AUTO.

In general, establishing observation equivalence between two specifications is a problem of polynomial complexity and is consequently very time-consuming for large specifications. A more realistic approach could alternatively be to use testing and modal logic and only attempt partial verification.

5 Conclusion

Specification and verification of complex realistic examples is an inherently difficult task which requires considerable skill and craftsmanship. However, effort and errors can be reduced if good tools and a clear methodology is used. We have presented a method of specification and verification based on CCS and observation equivalence, and a tool, the Concurrency Workbench, which supports this method. Both the method and the tool have some deficiencies. Below, we summarize our main results:

- CCS as a specification language lacks structural constructs such as operators and data types which are of considerable importance when specifying large systems.
- Observation equivalence as a method of verification is deficient in several respects:
  1. Observation equivalence is too strong because it is not only sensitive to internal nondeterminism but also to where the internal nondeterminism occurs.
  2. Observation equivalence does not naturally support a design process based on stepwise refinement.
  3. Observation equivalence is too weak to support modular verification and action refinement.
- The Concurrency Workbench is inefficient and lacks an interactive user interface.

Allusions to other methods of specification and verification have been made in this paper. LOTOS seems to compensate for the structural deficiencies of CCS as a specification language and has been used to specify and standardize large communication protocols. A number of interactive tools for specifying in LOTOS are also emerging. However, many methodology questions still remain unsolved. Our experience suggests that the design process should be an interactive process where specification and verification are alternately performed on increasingly more detailed specifications, i.e. a stepwise refinement process. The methodology should also lend itself to modular verification to make the verification of large complex systems possible. We have mentioned a few different methods which seem potentially promising in different respects. Clearly, their usefulness and applicability can only be determined by performing further case studies. One problem which we have encountered is that specification and in particular verification become difficult for large examples without automated tools. Consequently, the application of these methods relies heavily on the tool support that is available for each method.

References

[BB89a] B. Pehrson B. Backlund, H. Hagsand. Generation of graphic language-oriented design environments. Research Report R89011, SICS, 1989.

[BB89b] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P. van Eijk, C. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 77–82. North-Holand, 1989.

[BBK87] J. Baeten, A. Bergstra, and W. Klop. Ready-trace semantics for concrete process algebra with priority operator. *The Computer Journal*, 30(6), 1987.

[BEH89] L. Beckman, P. Ernberg, and T. Hovander. Utvärdering av en CCS-baserad metod för specifikation och verifiering av protokoll. Result of project performed by SICS in cooperation with Ellemtel(in Swedish), 1989.

[BEH90] L. Beckman, P. Ernberg, and T. Hovander. Statusrapport protokollprojektet. Result of project performed by SICS in cooperation with Ellemtel (in Swedish), 1990.

[BL89] G. Boudol and K. Larsen. Graphical versus logical specifications. Report R89-33, Department of Mathematics and Computer Science, Aalborg University, Aalborg, Denmark, 1989.

[CCI84a] CCITT. CCITT recommendation I.440: ISDN user-network interface data-link layer-specification, 1984.

[CCI84b] CCITT. CCITT recommendation I.441: ISDN user-network interface data-link layer-, 1984.

[CCI84c] CCITT. CCITT recommendation I.451: ISDN user-network interface layer 3 specification, 1984.

[CH88] R. Cleaveland and M. Hennessy. Priorities in process algebras. In *Logic in Computer Science*, pages 193–202. University of Sussex, Coputer Society Press, 1988.

[CPS88] R. Cleaveland, J. Parrow, and B. Steffen. *The Concurrency Workbench: Operating Instructions*, 1988.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. A semantics-based verification tool for finite-state systems. In *Protocol Specification, Testing, and Verifiation, IX*, 1989.

[Dah90] M. Dahlberg. Efficient algorithms for computing transitive closure in CWB. Technical Report (To be published), Swedish Institute of Computer Science, Kista, 1990.

[EF90] P. Ernberg and L. Fredlund. Identifying some bottlenecks of the concurrency workbench. Technical Report T90002, SICS, Kista, 1990.

[Eij89] P. Eijk. Lotos tools based on the cornell synthesizer. In E. Brinksma, G. Scollo, and C. Vissers, editors, *Proceedings of IFIP IX*, 1989. to be published.

[Ell87] Ellemtel Utvecklings AB. ISDN basic access, layer 3 protocol specification. Internal Report, 1987.

[Ell88] Ellemtel Utvecklings AB. Layer 3 protocol for the D-channel, anton terminal. Internal Report, 1988.

[Fau89a] C. Faure. Modelisation des couches 3 et 2 du RNIS pour un observateur. Technical Report 89390, LAAS, Toulouse, France, 1989.

[Fau89b] C. Faure. Protocole de signalisation du reseau numerique a integration de services. Technical Report 89126, LAAS, Toulouse, France, 1989.

[Fer89] J-C. Fernandez. Aldébaran: A tool for verification of communicating processes. Technical Report RTC 14, IMAG, Grenoble, 1989.

[FJP90] L. Fredlund, B. Jonsson, and J. Parrow. "an implemntation of a translational semantics for an imperative language". submitted CONCUR90, 1990.

[Gar89] H. Garavel. *Caesar 3.2 Reference Manual.* IMAG - LGI, Grenoble, France, 1989.

[GV89] J. Groote and F Vaandrager. An efficient algorithm for branching bisumulation and stuttering equivalence. Technical Report (draft full paper), Centre for Mathematics and Computer Science, Amsterdam, 1989.

[GW89] R. van Glabbeek and P. Weijland. Branching time and abstraction in bisimulation semantics. Report CS-R-89-11, CWI, 1989. (extended abstract).

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[JKZ88] J.Godskesen, K.Larsen, and M. Zeeberg. *TAV (Tools for Automatic Verification) Users Manual.* Aalborg University Center, Aalborg, Denmark, 1988.

[Lar89] K. Larsen. Modal specifications. Report 89-9, Department of Mathematics and Computer Science, Aalborg University, Aalborg, Denmark, 1989.

[Man89] J. Manas. Dining philosophers: A constraint-oriented specification. In P. van Eijk, C. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 439–451. North-Holand, 1989.

[Mil89a] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Mil89b] R. Milner. *Communication and Concurrency*, chapter 3.4. Prentice Hall, 1989.

[OH84] E R Olderog and C A R Hoare. Specification oriented semantics for communicating processes. Technical Report PRG-37, Programming Research Group, Oxford University, February 1984.

[Par88] J. Parrow. Verifying a CSMA/CD protocol with CCS. In S. Aggrawal and K. Sabnani, editors, *PSTV VIII*, pages 373–384. North-Holland, 1988.

[Phi87] I. Phillips. Refusal testing. *Theoretical Computer Science*, (50):241–284, 1987.

[QA89] J. Quemada and A. Azcorra. A constraint-oriented specification of al's node. In P. van Eijk, C. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 83–88. North-Holand, 1989.

[QPF89] J. Quemada, S. Pavon, and A. Fernandez. State exploration by transformation with LOLA. In *Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[SIG89] SIGMA. Technical annex, specification of IBC systems, generation of a methodology and architectural guidelines (SIGMA). Technical report, DIT,ELLEMTEL,INTELSA and SICS, 1989.

[SIG90] SIGMA. Deliverable 2: Characterization of target systems (selection of a case study). Technical report, DIT, 1990.

[Sta88] W. Stallings. *Data and Computer Communications*, pages 585–618. MacMillan Publishing Company, second edition, 1988.

[SV89] R. de Simone and D. Vergamini. *Aboard Auto.* INRIA, Sophia-Antipolis, 1989.

[Tre89] J. Tretmans. Hippo: A lotos simulator. In P. Eijk, C. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 391–396. North-Holland, 1989.

[Vaa86] F.W. Vaandrager. Verification of two communication protocols by means of a process algebra. Technical Report CS-R8608, Department of Software Technology, CWI, 1986.

[Vaa90] F.W. Vaandrager. Some observations on redundancy in context. In J.C.M. Baeten, editor, *Applications of Process Algebra*, number (to be published), pages 237–260. Cambridge University Press, 1990.

[Wal89] D.J. Walker. Automated analysis of mutual exclusion algorithms using CCS. Technical Report ECS-LFCS-89-91, Universisy of Edinburgh, Dept. of Computer Science, 1989.