Refining Interfaces of Communicating Systems *

Ed Brinksma
Dept. of Informatics, University of Twente †

Bengt Jonsson
Swedish Institute of Computer Science and Dept. of Computer Systems, Uppsala University

Fredrik Orava
Swedish Institute of Computer Science ‡

Abstract

There are now several theories for describing and reasoning about the behavior of communicating systems, where the behavior of a communicating system is described in terms of its capabilities to perform communication actions in cooperation with its environment. In such theories, preorders or equivalences are defined as criteria for when one system is an acceptable substitute or implementation of another. Existing theories of communicating systems define preorders or equivalence relations only between systems with identical sets of communication actions. In many practical design situations, however, it may be desirable to refine a system by changing its set of communication actions. We present a simple method for carrying out such refinements. The method is first formulated in a general setting, and then elaborated in more detail in the trace model and a simple version of the failure model. We illustrate the usefulness of our method by an application to I.451, an ISDN access protocol.

1 Introduction

A commonly used idealization of the program development process is that from an initial high-level specification one performs a sequence of refinement steps, at each of which details are added, until the specification attains a form which can be realized in software and/or hardware. A theory which supports such a stepwise refinement process should provide a correctness criterion for refinements and methods for verifying them. For communicating systems, there are several theories which define preorders that can be used as criteria for refinement. Examples are the testing preorders of Hennessy and de Nicola

*This work was partially supported by the Swedish Board for Technical Development (ESPRIT/BRA project 3096 SPEC)

†Address: P.O. Box 217, 7500 AE Enschede, The Netherlands. E-mail: brinksma@cs.utwente.nl

‡Address: P.O. Box 1263, S-164 28 Kista, Sweden. E-mail: bengt@sics.se, fredrik@sics.se

[dNH84], the failure model [BHR84], the readiness model [OH86], and refinements between I/O-automata [LT87, Jon87]. Common to these preorders is that they compare two systems only if both systems communicate with their environment over the same set of communication actions.

The ability to compare only systems with the same set of communication actions can in some situations be too restrictive. Assume for instance that during the development of a particular program the specification has been divided into a part which contains a database and a part which uses this database. Furthermore, the division has been made at a high level of abstraction, where it makes most sense to describe the interaction between the database and its user by a single action, called *update*. The database can perform many other actions, that e.g. query the database. After several refinement steps of the database it may become sensible to replace the single action *update* by the two actions *update.req* and *update.conf*. It is then necessary to use a criterion for correctness of refinements which can compare two systems that communicate over different sets of actions. The standard refinement notions, e.g. trace or failure containment, are no longer adequate.

In this paper, we consider the problem of refining communicating systems by changing their sets of communication actions. Our approach is partly motivated by the design scenario outlined in the preceding paragraph. We will limit ourselves to the case where the replaced set of actions is used by only two of the components in a system. This implies that the refinement should be carried out simultaneously in the two components that are concerned. Furthermore, the joint behavior of the two systems after the refinement should be a correct refinement of the two systems before the refinement. This last observation enables us to relate our approach to existing preorders between communicating systems.

We will define a simple way of describing certain types of refinement that change the set of actions, and prove that it is correct in the sense of the preceding paragraph. The main idea is that the refinement is described by a shift, or displacement, of the interface. This is a simple idea which is independent of the particular model of communicating systems (trace model, failure model, bisimulation semantics, etc.) that one starts with. We will thereafter discuss various aspects of this idea in more specific contexts, such as the trace model and the failure model. For the failure model, we will define a specific extension in which shifts of interfaces can be described.

We intend to show the usefulness of our ideas through applications to nontrivial examples. In this paper we include an application to I.451, which is a layer three protocol used by an ISDN-terminal to access the ISDN network.

In summary, the main contributions of this paper are the following:

1. We define a notion of interface refinement where the refinement concerns both the process and its environment. This is in contrast to [AH88, GW89, GMM88] where actions of a process are refined without taking the environment into account.

2. We define a simple method for describing this kind of interface refinement together with a simple rule for proving the refinement correct.

3. We define an extension of the failure model for describing shifts of interfaces.
4. As a case study we provide an application of the method to a nontrivial example.

The paper is organized as follows: In the next section, we present the framework of communicating systems and the operations that we assume. Section 3 describes the method of refining an interface by displacing it, and illustrates it by a simple example. The method is specialized to the trace model in Section 4, and to an extended version of the failure model in Section 5. In Section 6 we apply the method described in Section 3 to the I.451 protocol. Finally, Section 7 contains conclusions and some comparisons with related work.

2 Communicating systems

Our method for refining interfaces of communicating systems will first be formulated in a general setting. The method assumes a certain structure on the theory of communicating systems, which is met by most existing theories. In this section, we present the general operations and assumptions on communicating systems that will subsequently be needed.

We assume a set of *actions*, which does not include the silent action τ. We assume a set of *processes*. With each process P is associated a set αP of actions, called the *sort* of P. We assume that there is a refinement relation between processes, denoted $\lhd$, which for each set L of actions is a preorder on the set of processes with sort L. The relation $\lhd$ could for instance be trace inclusion, the ordering in the failure model, or observation equivalence. We define $P \simeq Q$ to denote $P \lhd Q \lhd P$.

We assume the following operations on processes:

- If P and Q are processes and L is a set of actions such that both $L \subseteq \alpha P$ and $L \subseteq \alpha Q$, there is an operation, called the *parallel composition* of P and Q synchronizing over L, which is denoted $P \|_L Q$. This operation is used e.g. in LOTOS [vEVD89], and can also be defined in CSP [Hoa85]. Intuitively, the process $P \|_L Q$ denotes the result of putting P and Q in parallel and synchronizing over the actions in L.
- If P is a processes such that $L \subseteq \alpha P$, then the *hiding* of L in P, denoted $P \setminus L$, intuitively makes the actions in the set L invisible.
- As a shorthand, we will use $P|_L Q$ to denote $(P \|_L Q) \setminus L$.

The operations can be defined for various semantic models of processes. For the moment, we will not be specific about the model used. However, we need to assume certain basic properties of the operations, which will be satisfied by most "reasonable" models in the literature. The properties in question are:

(A1) All operations are assumed to be monotonic with respect to the refinement relation $\lhd$

(A2) For arbitrary sets L_1, L_2 of actions $(P \setminus L_1) \setminus L_2 \;\simeq\; (P \setminus L_2) \setminus L_1$

(A3) If L_1 is disjoint from both L_2 and the sort of Q, then $(P \setminus L_1) \|_{L_2} Q \;\simeq\; (P \|_{L_2} Q) \setminus L_1$

(A4) If $L_1 \cap L_2 = \emptyset$, and $L_1 \cap \alpha R = \emptyset$, and $L_2 \cap \alpha P = \emptyset$, then

$$(P \|_{L_1} Q) \|_{L_2} R \;\simeq\; P \|_{L_1} (Q \|_{L_2} R)$$

(A5) $P \|_L Q \;\simeq\; Q \|_L P$

From the above axioms we can infer that the combination of parallel composition and hiding is associative in the following sense.

(A6) If $L_1 \cap L_2 = \emptyset$, and $L_1 \cap \alpha R = \emptyset$, and $L_2 \cap \alpha P = \emptyset$, then

$$(P |_{L_1} Q) |_{L_2} R \;\simeq\; P |_{L_1} (Q |_{L_2} R)$$

The proof of (A6) is a simple application of the definition of $|_L$ and of the properties (A1) - (A5).

$$\begin{aligned} & (P|_{L_1}Q)|_{L_2}R \simeq (((P\|_{L_1}Q) \setminus L_1)\|_{L_2}R) \setminus L_2 \simeq ((P\|_{L_1}Q)\|_{L_2}R) \setminus L_1 \setminus L_2 \simeq \\ \simeq\; & (P\|_{L_1}(Q\|_{L_2}R)) \setminus L_2 \setminus L_1 \simeq (P\|_{L_2}((Q\|_{L_2}R) \setminus L_2)) \setminus L_1 \simeq P|_{L_1}(Q|_{L_2}R) \end{aligned}$$

3 Interface Refinement

In this subsection, we describe a method for refining the interface between two processes within the general framework outlined in Section 2. As indicated in the introduction, the scenario is that we are given two processes P and Q that both communicate over a given set of abstract actions, L_A. It is required to refine P into P' and Q into Q' such that both P' and Q' communicate over a new set of concrete actions, L_C, and not over L_A. In order for this refinement to make sense when the two processes are put together, the joint behavior of P' and Q' should refine the joint behavior of P and Q, i.e.,

$$P'|_{L_C}Q' \lhd P|_{L_A}Q \quad .$$

In general, there may be many ways of carrying out such a refinement. If we want to cover all possible ways of refining a pair P, Q into P', Q', we cannot be more specific about the refinement than to state the above criterion. However, our goal is to present a more specific method in which we only refine a part of P (Q), namely the part which is close to the interface L_A. In such a method, we should be able to describe the relationship between P and P' (and analogously between Q and Q') by some relation between their behaviors with respect to the actions L_A and L_C. In the following, we describe one way of carrying out such a refinement. We first define notation for describing the interface change, and thereafter describe the method.

For any process I and set L of actions with $L \subseteq \alpha I$, define the mapping $\widehat{I}_L$ from the set of processes whose sort contains L by $\widehat{I}_L(P) = I|_L P$, i.e., $\widehat{I}_L$ changes P by adding I,

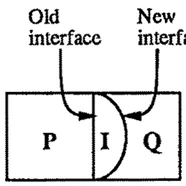

Figure 1: Simple interface refinement

with which it communicates via the synchronous interface L. The interface L thereafter becomes hidden. The idea of defining this notation, is that we can then define the inverse mapping $(\widehat{I}_L)^{-1}$ on the set of processes whose sort is disjoint from L. If $I|_L R \simeq Q$ then we put $(\widehat{I}_L)^{-1}(Q) = R$. In general, $(\widehat{I}_L)^{-1}(Q)$ need not exist, and if it exists it need not be a unique process. If $(\widehat{I}_L)^{-1}(Q)$ exists, then we have

$$Q' \lhd (\widehat{I}_L)^{-1}(Q) \qquad \Longrightarrow \qquad \widehat{I}_L(Q') \lhd Q$$

by the monotonicity of $\lhd$. We remark that when the preorder $\lhd$ has certain nice properties, which is the case in for example the failure model, then there is a unique "maximal" process Q' which satisfies the above definition.

We can now present our method for refinement of interfaces.

Definition 3.1 Let P and Q be processes both of whose sorts contain L_A and are disjoint from L_C. Let I be a process whose sort is $L_A \cup L_C$. A *simple interface refinement* which adds I to P and subtracts I from Q results in processes P' and Q' such that

1. $P' \lhd \widehat{I}_{L_A}(P)$, and
2. $Q' \lhd (\widehat{I}_{L_C})^{-1}(Q)$.

and $L_A \cap \alpha Q' = \emptyset$. □

Intuitively, the operation in the previous definition adds the process I to P and cuts the process I out of Q. It can be illustrated graphically as in Figure 1. Intuitively, the refinement has been carried out by a displacement of the interface. The process I describes the "area" over which the displacement is performed. The soundness of the method follows from the following proposition.

Proposition 3.2 *If P, Q, L_A, L_C, I, P', and Q' satisfy the conditions of Definition 3.1, then*

$$P'|_{L_C} Q' \lhd P|_{L_A} Q$$

□

Proof: The proof is a sequence of simple applications of the definition of simple interface refinement and the property (A6).

$$P'|_{L_C}Q' \quad \lhd \quad (P|_{L_A}I)|_{L_C}Q' \quad \lhd \quad P|_{L_A}(I|_{L_C}Q') \quad \lhd \quad P|_{L_A}Q$$

□

4 Refinement in the Trace Model

In this section, we indicate how the method of Section 3 could appear in a simple trace model. The trace model is very simple and cannot represent deadlock or liveness properties of systems.

In the trace model, we associate (in addition to αP) with each process P a set tP of *traces*. The set tP is a prefixed-closed subset of $(\alpha P)^*$. i.e., a prefixed-closed subset of sequences of actions in αP. Intuitively, a trace of P is a sequence of actions that may occur up to some point in a computation. As an example, a one-place buffer can be represented as the process P, where αP is the set of actions $\{in, out\}$, and tP is the set $(in\ out)^* \cup (in\ out)^*\ in$ (where we borrow the notation for regular expressions).

For a sequence σ and set of actions L, let $\sigma\lceil_L$ denote the projection of the sequence σ onto the communication events of L, i.e., the subsequence of σ consisting of the actions in L. For a set of actions L, let $\sigma \setminus L$ denote σ with all occurrences of actions in L removed.

In this paper we shall for simplicity consider the parallel composition $P\|_L Q$ only in the case that $L = \alpha P \cap \alpha Q$. The operations in the trace model can then be defined as follows:

- The process $P\|_L Q$ is defined by

$$\alpha(P\|_L Q) = \alpha P \cup \alpha Q$$
$$t(P\|_L Q) = \{\sigma \in (\alpha P \cup \alpha Q)^* \mid \sigma\lceil_{\alpha P} \in tP \quad \text{and} \quad \sigma\lceil_{\alpha Q} \in tQ\}$$

- The process $P \setminus L$ is defined by

$$\alpha(P \setminus L) = \alpha P \setminus L$$
$$t(P \setminus L) = \{\sigma \setminus L \mid \sigma \in tP\}$$

- The refinement relation $\lhd$ is defined as

$$P \lhd Q \equiv \alpha P = \alpha Q \ \text{ and } \ tP \subseteq tQ$$

Example 4.1 As an example we shall carry out the refinement alluded to in the introduction of the paper, where an action *update* is refined by the two actions *update.req* and *update.con*. Let L_A be $\{update\}$, and let L_C be $\{update.req, update.conf\}$. The process I is defined by $\alpha I = L_C \cup L_A$, and tI is the set of prefixes of sequences in $(update.req\ \ update\ \ update.conf)^*$.

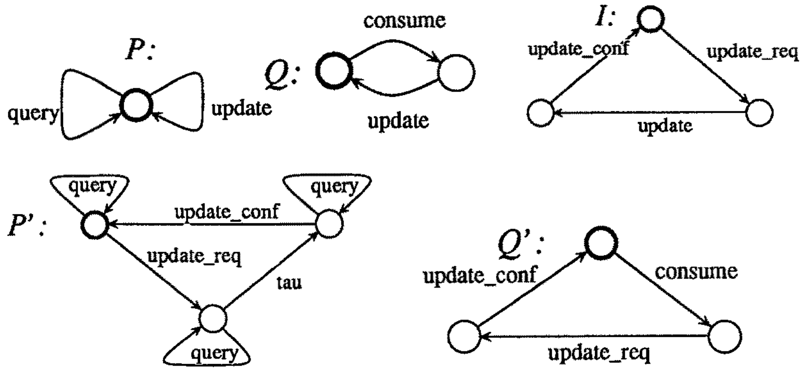

Figure 2: Abstract specifications of the data base P (upper left), and the user Q (upper middle). Specification of I (upper right). Concrete specifications of the data base P' (lower left) and the user Q' (lower right).

We next describe how a refinement is carried out for two particular processes P and Q. Intuitively, P is a database, and Q is some process that uses the database. The process P can perform the actions *query* and *update* in any order. The process P', defined by $\widehat{I}_{L_A}(P)$, contains all traces in which each *update.conf* is preceded by a corresponding *update.req*. If the process Q is a user that alternates the actions *consume* and *update*, then Q', defined by $(\widehat{I}_{L_C})^{-1}(Q)$, must perform the actions *consume*, *update.req*, and *update.conf* strictly cyclically in this order. The processes can be described by the transition diagrams in Figure 2 that generate the traces of each process.

□

5 Refinement in the Failure Model

In this section, we indicate how the method of Section 3 could appear in a simple variant of the failure model that does not consider divergence. The particular form of the failure model is a simplification of the model in [BHR84], in that the hiding operation does not add divergence. This simplification is mainly for purposes of presentation. There are also other versions of the failure model with an improved treatment of divergence [BR85], but we will not consider them in this paper.

In the failure model, we associate with each process P a set fP of *failures* (in addition to αP). A failure in fP is a pair $\langle \sigma, X \rangle$, where $\sigma \in (\alpha P)^*$ and $X \subseteq \alpha P$. Intuitively, a failure $\langle \sigma, X \rangle$ is in fP if P can perform the sequence σ of actions and thereafter refuse to perform any of the actions in X, i.e., the process may deadlock if only the actions in X are offered for communication. The first component σ of a failure is often referred to as a *trace* and the second component X as a *refusal*. We require that the set fP satisfy the following closure properties:

i) $Y \subseteq X \wedge \langle \sigma, X \rangle \in fP \Longrightarrow \langle \sigma, Y \rangle \in fP$

ii) $\langle \sigma, X\rangle \in fP \wedge \langle \sigma\langle c\rangle, \emptyset\rangle \notin fP \Longrightarrow \langle \sigma, X \cup \{c\}\rangle \in fP$

Intuitively, requirement i) says that if a process can refuse the actions in X, then it can refuse any smaller set of actions. Requirement ii) says that the impossible actions can always be refused.

The operations in the failure model are defined as follows:

- The process $P\|_L Q$ is defined by

$$\begin{aligned} &\alpha(P\|_L Q) = \alpha P \cup \alpha Q \\ &f(P\|_L Q) = \{\langle \sigma, X_P \cup X_Q\rangle \mid \langle \sigma\lceil_{\alpha P}, X_P\rangle \in fP \quad \text{and} \quad \langle \sigma\lceil_{\alpha Q}, X_Q\rangle \in fQ\} \end{aligned}$$

- The process $P \setminus L$ is defined by

$$\begin{aligned} &\alpha(P \setminus L) = \alpha P \setminus L \\ &f(P \setminus L) = \{\langle \sigma \setminus L, X\rangle \mid \langle \sigma, X \cup L\rangle \in fP\} \end{aligned}$$

- The refinement relation $\lhd$ is defined as

$$P \lhd Q \equiv \alpha P = \alpha Q \text{ and } fP \subseteq fQ$$

As an illustration, we could carry out Example 4.1 using the failure model, and the result would be analogous. The transition diagrams in Figure 2 can be interpreted as denotations in the failure model: $\langle \sigma, X\rangle$ is a failure if after the trace σ there are no transitions in the diagram labeled with elements from X. Definition 3.1 works nicely also in the failure model.

However, assume P is the process $STOP$ which never performs any action (we can e.g. think of $STOP$ as a dead-locked database). The process $\widehat{I}_{L_A}(STOP)$ will then be able to perform the action *update.req* although $STOP$ will never perform the action *update* and $\widehat{I}_{L_A}(STOP)$ will never perform *update.conf*. Thus, performing *update.req* serves no real purpose, and it would be more sensible to refuse to perform the action *update.req* in the first place. However, in the current setting, $\widehat{I}_{L_A}(P)$ never refuses *update.req* for any process P, since the process I cannot use information about future refusals of P to produce refusals of P'. However, since I will never appear as a process by itself, we can develop a theory of processes that can indeed transfer information about refusals about other processes, and apply this theory when defining the interface changes of our method. In the rest of this section, we shall briefly develop such a theory.

In the *conditional failure model*, we associate with each processes P a set cP of *conditional failures.* A conditional failure in cP is a triple $\langle \sigma, X, C\rangle$, where $\sigma \in (\alpha P)^*$ and X and C are subsets of αP. Intuitively, a conditional failure $\langle \sigma, X, C\rangle$ is in cP if P can perform the sequence σ of actions and thereafter refuse to perform any of the actions in X in a situation where the environment refuses to perform the actions in C. The first component σ of a failure is often referred to as a *trace* and the pair $\langle X, C\rangle$ as a *conditional refusal.* We require that the set cP in addition to properties corresponding to i) and ii) above also satisfy the additional closure properties

iii) $C \subseteq D \wedge \langle \sigma, X, C \rangle \in cP \Longrightarrow \langle \sigma, X, D \rangle \in cP$

iv) $\langle \sigma, X, C \rangle \in cP \Longrightarrow \langle \sigma, X, C \setminus X \rangle \in cP$.

Intuitively, requirement iii) says that if P can refuse the actions in X under the condition that the environment refuses the actions in C, then P refuses the actions in X also if the environment refuses any larger set. Requirement iv) says that the total refusal $X \cup C$ can always be factored into disjoint contributions X of the process and C of the environment.

The operations in the conditional failure model are defined as follows:

- The process $P \|_L Q$ is defined by

$$\begin{aligned} \alpha(P\|_L Q) &= \alpha P \cup \alpha Q \\ c(P\|_L Q) &= \left\{ \langle \sigma, X_P \cup X_Q, C \rangle \;\middle|\; \begin{array}{l} \langle \sigma\lceil_{\alpha P}, X_P, (C \cup X_Q) \cap \alpha P \rangle \in cP \quad \wedge \\ \langle \sigma\lceil_{\alpha Q}, X_Q, (C \cup X_P) \cap \alpha Q \rangle \in cQ \end{array} \right\} \end{aligned}$$

- The process $P \setminus L$ is defined by

$$\begin{aligned} \alpha(P \setminus L) &= \alpha P \setminus L \\ c(P \setminus L) &= \{ \langle \sigma \setminus L, X, C \rangle \mid \langle \sigma, X \cup L, C \rangle \in cP \} \end{aligned}$$

- The refinement relation $\lhd$ is defined as

$$P \lhd Q \equiv \alpha P = \alpha Q \quad \text{and} \quad cP \subseteq cQ$$

When we apply the conditional failure model in the simple method of Definition 3.1, we will require that P, Q, P' and Q' are processes in the failure model. A process in the failure model can be identified with a process in the conditional failure model by the identification

$$cP = \{ \langle \sigma, X, C \rangle \mid \langle \sigma, X \rangle \in fP \ \wedge \ C \subseteq \alpha P \}$$

i.e. X does not depend at all on C, in particular $\langle \sigma, X, \emptyset \rangle \in cP$, from which cP can be obtained via iii) above. The process I can be a process in the conditional failure model.

6 Example

In this section we present an application of the method proposed in Section 3. As an example we have chosen a layer 3 access protocol of ISDN, as specified in CCITT's recommendation I.451 [CCI85]. This protocol specifies procedures for establishing, maintaining and clearing of network connections at the ISDN user-network interface. ISDN provides a number of services on top of a digital telephone network, ranging from digital telephony to a variety of data services e.g. file transfer. The user-network interface comprises the

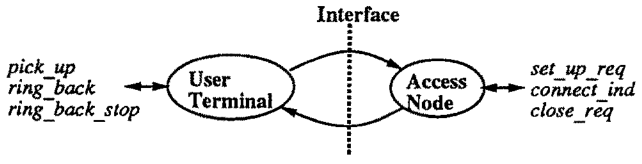

Figure 3: The architecture of the user-network interface

three lower layers of the OSI model: the network, link, and physical layers. The CCITT recommendation I.451 consists of the access protocol for layer 3.

The architecture of the ISDN user-network interface is shown in Figure 3. The entities which interact via the interface are: at the user side a user-terminal, and at the network side an access-node. We have chosen to specify a subset of the I.451 protocol, where a call establishment can only be requested by the user and a call release can only be requested by the called user or the ISDN network.

In this section we will present specifications of the I.451 protocol at two levels of detail. First we give abstract specifications of the user-terminal and the access-node entities. We thereafter refine the interface between the entities using the method described in Section 3. The purpose is that the abstract specification should be easily understandable, whereas the refined specifications should correspond to the I.451 protocol. We will in this example use the trace model of Section 4 and indicate how the refinement can be performed in the conditional failure model of Section 5. As in Example 4.1, processes will be given as transition diagrams, which can be regarded as generators of sets of traces (failures/conditional failures).

6.1 Abstract Specification

The abstract specifications of the user-terminal and the access-node are given in Figure 4. As indicated in Figure 3 the user-terminal communicates with a user (a telephone subscriber) via the actions *pick_up*, *ring_back* and *ring_back_stop*. The abstract interface between the user-terminal and the access-node, corresponding to L_A, consists of $\{start\_estab, end\_estab, close\}$. The access-node communicates with the rest of the ISDN network via actions *set_up_req*, *connect_ind* and *close_req*. These actions may or may not be present in a real implementation, since the I.451 recommendation does not specify the communication between the access-node and the rest of the ISDN network.

We first describe the operation of the user-terminal. A *pick_up*-action is interpreted as the user picking up the handset and dialing a subscriber number. The user-terminal then requests a connection to the called end-user by the action *start_estab*, after which the user is notified, by a *ring_back*-action, that the ISDN network is trying to establish a connection. When the connection is established the user-terminal receives an *end_estab*-action from the access-node and the user is notified that a connection exists by a *ring_back_stop*-action. Finally, at any point after the occurrence of the *start_estab*-action, the access-node may issue a *close*-action to the user-terminal, indicating that the connection has been closed. Observe that, according to [CCI85], the user is not explicitly notified that the connection

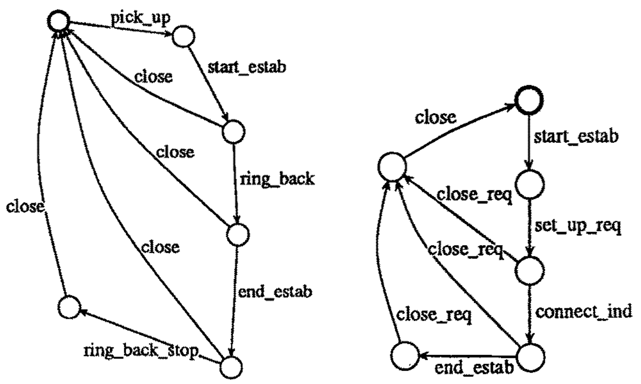

Figure 4: Abstract specifications of the user-terminal (left) and the access-node (right)

has been closed.

The access-node awaits a *start_estab*-action from the user-terminal. Thereafter a *set_up_req*-action is transmitted to the rest of the ISDN network indicating that a connection should be established to the called end-user. The reception of a *connect_ind*-action signals that the requested connection has been established. The access-node then notifies the user-terminal that a connection exists by initiating an *end_estab*-action. At any point after the *set_up_req*-action the access-node may receive a *close_req*-action indicating that ISDN network or the called end-user wants to close the connection. The access-node then closes the connection to the user-terminal by using a *close*-action.

6.2 Specification of the Refined Interface

We now use the method of Definition 3.1 to refine the interface between the user-terminal and access-node in Figure 4. The change of interface is described by the process I in Figure 5, where the actions of the abstract interface, L_A, are capitalized. The refinement will be carried out by adding I to the access-node and subtracting it from the user-terminal. We first describe the operation of the concrete interface, and then how it is related to the abstract interface. The connection is initiated by the user-terminal sending a *set_up*-action. The access-node responds with a *set_up_ack*-action. We assume that the whole subscriber number cannot be transmitted in one action, but is transmitted transmitted digit by digit in *info*-actions. When the access-node recognizes a complete subscriber number, a *call_proc*-action is transmitted to the user-terminal. Thereafter the access-node informs the user-terminal when the connection has reached the called user by an *alerting*-action. When the connection is established, the user-terminal is informed by the access-node sending a *connect*-action. This action may transfer information about the connection (e.g., quality of service). If the connection is acceptable, the user-terminal responds with a *connect_ack*-action. We have only considered the case when the connection is acceptable. The access-node initiates the closing of the connection by sending a *disc*-action. If the connection should be completely released, the user-terminal responds by sending a *rel*-action (we have only considered this case, another possibility is to keep the

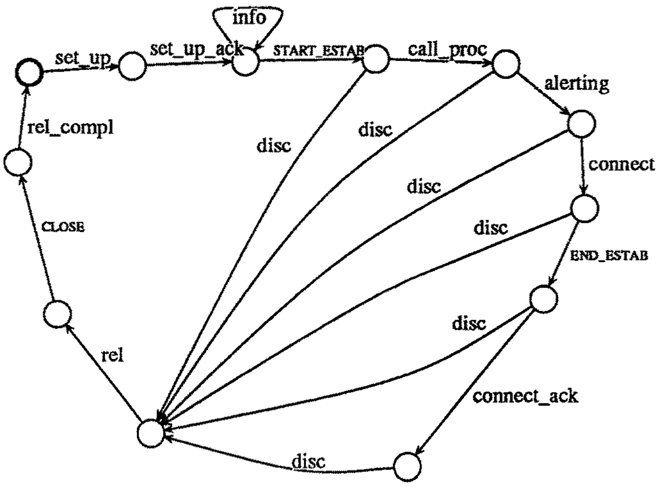

Figure 5: A process describing the change of interface

connection for future use). The closing is completed by the access-node releasing the connection and sending a *rel_compl*-action to the user-terminal.

The relation between the concrete and the abstract interface is as follows. We consider the abstract action *start_estab* to take place when the access-node has all information necessary to reach the called user, since the access-node then can initiate the call in the rest of the ISDN network. The *end_estab* is considered to take place when the connection is established, i.e. when the access-node has informed the user-terminal of the existence of the connection. Finally, the *close* is considered to take place when the connection is actually closed down, i.e. after the access-node has received the *rel*-action.

6.3 Refined Specification

In this subsection we present the refined specifications *UserTerm'* and *AccessNode'*. These processes should satisfy the following properties:

$$AccessNode' \lhd \hat{I}_{L_A}(AccessNode) \qquad \text{and} \qquad UserTerm' \lhd (\hat{I}_{L_C})^{-1}(UserTerm)$$

The process *AccessNode'* is shown in Figure 6. It is equal to $\hat{I}_{L_A}(AccessNode)$ with the exception that all traces where *disc* occurs before the corresponding *close_req* have been removed. The motivation behind this simplification is the following. In $\hat{I}_{L_A}(AccessNode)$ a *disc*-action can occur without any preceding *close_req*, i.e. the access-node may initiate the closing of the connection without the called user or the ISDN network requesting it. This violates our informal assumption about the protocol that the access-node should never spontaneously close the connection. In the trace model, we are free to refine processes by removing traces, so the removal is correct. Note however that even without the removal, the parallel composition of $\hat{I}_{L_A}(AccessNode)$ and *UserTerm'* (defined below) is a correct refinement of $(AccessNode|_{L_A} UserTerm)$.

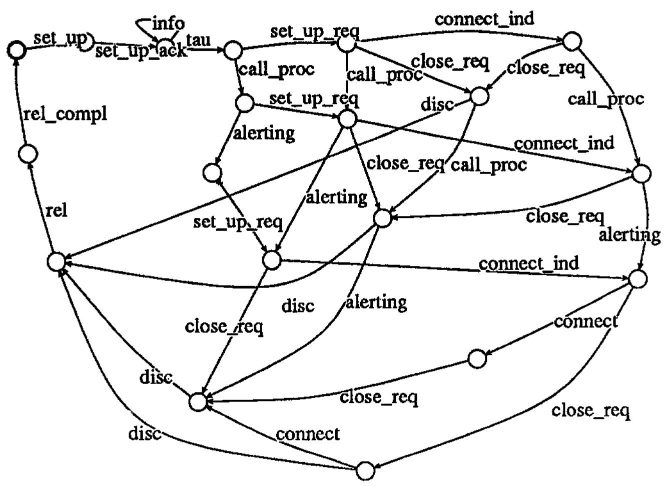

Figure 6: The refined access-node

In the failure model, the above removal is not correct, but can be made correct by considering the conditional failure model. The reason why the removal is not correct is that after the initial actions of the protocol, $\hat{I}_{L_A}(AccessNode)$ may not refuse a *disc*-action, whereas *AccessNode'* may refuse a *disc*-action if no *close_req*-action has previously occurred. Clearly, *AccessNode'* does not refine $\hat{I}_{L_A}(AccessNode)$ in the failure model. This situation is analogous with the situation explained in the example in Section 5. To use the conditional failure model we interpret I as a denotation in the failure model as explained in Section 5. Then we identify this process with a process in the conditional failure model using the identification defined in Section 5. Finally, we add the conditional failure $\langle\sigma, \{disc\}, \{close\}\rangle$ to cI if $\langle\sigma, X, C\rangle \in cI$ and $disc \notin X$. With this addition, the removal above is correct also in the conditional failure model.

The refined user-terminal, *UserTerm'* is shown in Figure 7. The behavior of this process defines a strict cyclic order between the actions in L_C, with the exception that *disc* (followed by *rel* and *rel_compl*) can occur at any time after a *set_up_ack*-action is received. This process satisfies the requirement $\hat{I}_{L_C}(UserTerm') \simeq UserTerm$ and thus by the definition of the inverse mapping, $(\hat{I}_L)^{-1}$, in Section 3 we can choose $UserTerm' = (\hat{I}_{L_C})^{-1}(UserTerm)$.

We have now motivated that *UserTerm*, *AccessNode*, L_A, *UserTerm'*, *AccessNode'*, L_C and I satisfy the conditions in Definition 3.1. From Proposition 3.2 we conclude that

$$(UserTerm'|_{L_C}AccessNode') \quad \lhd \quad (UserTerm|_{L_A}AccessNode)$$

7 Conclusions and Related Work

We have presented a method for refining communicating systems by changing their sort. The method was first presented in a general setting, and then elaborated in the trace

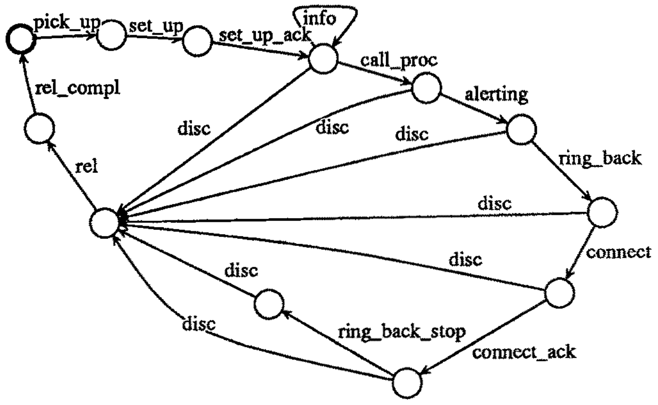

Figure 7: The refined user-terminal

model and a simple variant of the failure model. To show the usefulness of the method, we applied it to an existing access protocol in an ISDN network. In the example, we first presented a simple and abstract version of the protocol, and then used our method to refine the protocol into a less abstract protocol.

Our method could be said to serve two purposes. It enables a designer to use an abstract version of an interface between two processes during the design process, thus alleviating him from the burden of representing all existing interfaces in complete detail. It also gives a way of assigning precise meanings to abstract descriptions of interfaces. The idea behind a design can then be described abstractly at a level of detail appropriate to the situation at hand, and still retain preciseness.

In our work, we have also considered more general methods which are also based on the idea of displacing the interface, but where the displacement is done symmetrically, in both directions at the same time. However, this paper suggests that the simple method of Section 3 is adequate in many practical situations.

A problem related to refinement of interfaces is *action refinement.* Action refinement is usually understood as the replacement of a single action by some behavior with several actions. The problem of finding equivalences between communicating systems which are preserved under action refinement has been studied by Aceto and Hennessy [AH88] and by van Glabbeek and Weijland [GW89]. In contrast to our work, these works consider refinements that are performed without any assumptions about the environment of the process. Back [Bac89] and Lamport [Lam90] propose methods for verifying that the refinement of an action preserves correctness properties of a system. The main issue involved is that the replacing actions are not atomic in the same way as the replaced actions. In [Lam86], Lamport considers refinement of atomic actions in a partial order setting.

Another approach to describing protocols abstractly is to divide them into phases. Stomp and de Roever [SdR89] present a design principle where a distributed algorithm is first decomposed into phases, whereafter each phase is refined separately. It is then important to control the interaction between phases. In our example in Section 6 it might be possible to identify three phases corresponding to the three abstract actions *start_estab*, *end_estab*, *close*, instead of considering them together in the more abstract specification. Chow, Gouda and Lam [CGL85] propose a methodology for constructing multiphase communication protocols from a set of single phase protocols. If each phase possesses certain nice properties, such as deadlock freedom and proper termination, then the whole protocol possesses these properties.

Acknowledgment

We wish to thank members of the ESPRIT/BRA project SPEC for comments on this work.

References

[AH88] L. Aceto and M. Hennessy. Towards action-refinement in process algebras. Technical Report 3/88, Department of Computer Science, University of Sussex, 1988.

[Bac89] R.J.R. Back. A method for refining atomicity in parallel algorithms. In *Proc. PARLE 89*, volume 365 of *Lecture Notes in Computer Science*, pages 199–216. Springer Verlag, 1989.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[BR85] S.D. Brookes and A.W. Roscoe. An improved failures model for communicating processes. In Brookes, Roscoe, and Winskel, editors, *Proc. Seminar on Concurrency, 1984*, volume 197 of *Lecture Notes in Computer Science*, pages 268–280. Springer Verlag, 1985.

[CCI85] CCITT. *CCITT Recomendation I.451:ISDN user-network interface layer 3 specification*, 1985.

[CGL85] C.-H. Chow, M. Gouda, and S. Lam. A discipline for constructing multiphase communication protocols. *ACM Transactions on Computer Systems*, 3(4):315–343, 1985.

[dNH84] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[GMM88] R. Gorrieri, S. Marchetti, and U. Montanari. A^2CCS: A simple extension of CCS for handling atomic actions. In *CAAP '88*, volume 299 of *Lecture Notes in Computer Science*, pages 258–270. Springer Verlag, 1988.

[GW89] R.J. Glabbeek and W.P. Weijland. Refinement in branching time semantics. Technical Report CS-R8922, CWI, 1989.

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

[Jon87] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Dept. of Computer Systems, Uppsala University, Sweden, Uppsala, Sweden, 1987. Available as report DoCS 87/09.

[Lam86] L. Lamport. On interprocess communication Part I: Basic formalism. *Distributed Computing*, 1(2):77–85, 1986.

[Lam90] L. Lamport. A theorem on atomicity in distributed algorithms. *Distributed Computing*, 4(2):59–68, 1990.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6:th ACM Symp. on Principles of Distributed Computing, Vancouver, Canada*, pages 137–151, 1987.

[OH86] E.R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.

[SdR89] F.A. Stomp and W.P de Roever. Designing distributed algorithms by means of sequentially phased reasoning. Technical Report 89-8, University of Nijmegen, 1989.

[vEVD89] P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors. *The Formal Specification Language LOTOS.* North-Holland, 1989.