7

Using Formal Methods in the Development of Protocols for Multi-User Multimedia Systems*

Stephan Kleuker
University of Oldenburg, Department of Computer Science
P.O. Box 2503, 26111 Oldenburg, Germany
E-mail: kleuker@informatik.uni-oldenburg.de

Abstract

This paper presents the results of applying a formal method in an industrial feasability study. The chosen case study is a platform for the development of services for multi-user multimedia systems with the desired property that the functionality of the system can be extended step by step in the future.

The aim of this paper is to present a formal approach in which such a complex distributed system is developed step by step through different abstraction levels. It is shown that formal methods can be combined with existing standards which contain mainly textual descriptions of the protocols. A bottom-up approach is presented in which more abstract elements for the description of services are based on verified lower level protocols.

1 BACKGROUND

Telecommunication systems are complex distributed systems with the property that their development never terminates. Typically, new functionalities or services have to be added to already running software. However, the process of extending an existing system is not treated by formal methods so far. Only a top-down development from requirements to programs is supported.

Such a top-down development is e.g. formalized in the ESPRIT Basic Research Action ProCoS [2, 3, 4, 19] (Provably Correct Systems). ProCoS is a wide-spectrum verification project where embedded communicating systems are studied at various levels of abstraction ranging from requirements" capture over specification language and programming language down to the machine language. It emphasizes a constructive approach to correctness, using stepwise transformations between specifications, designs, programs, compilers and hardware.

In the research project *Provably Correct Communication Networks* [11, 12] — abbreviated as CoCoN — the ideas of ProCoS are extended with a method for the development of *extensible* systems. Extensibility means that *new functionality* can be added step by step to an existing system. CoCoN was carried out in close cooperation between Philips Research Laboratories

*This research was partially supported by the Philips Research Laboratories Aachen as part of the project CoCoN (Provably Correct Communication Networks) and partially supported by the Leibniz Programme of the Deutsche Forschungsgemeinschaft (DFG) under grant No. Ol 98/1-1

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

Aachen and the Department of Computer Science at the University of Oldenburg from 1993 to 1996.

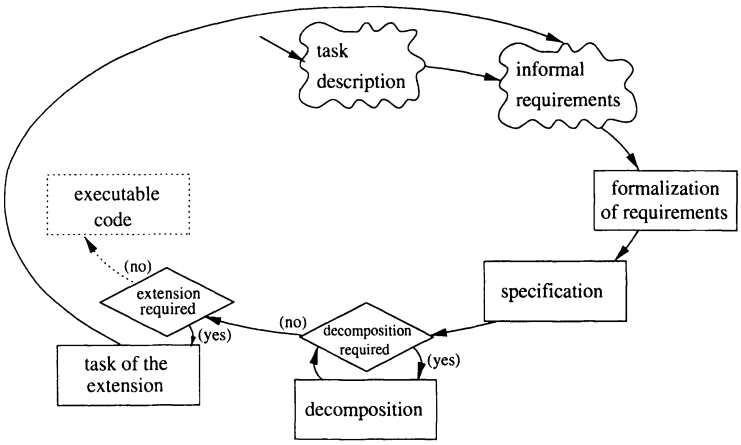


Figure 1 Summary of the development steps

The CoCoN method for the development of distributed systems is sketched in figure 1 and can be described as follows: The complete development begins with describing the main task of the desired system. This task is analyzed and split into subtasks. Tasks can be described in natural language. These tasks are structured as a set of informal requirements. Subsequently, informal requirements are translated into formal ones. A system (or specification) is provably correct if and only if the system fulfils these formal requirements.

The next step is the development of a first specification which already takes into account the architectural idea, i.e. it specifies the components and the interfaces between them. It is then proven that this specification fulfils the requirements and is therefore provably correct.

At this point two cases have to be considered. Either this specification is the final desired result and no changes are needed. Then this specification can be transformed into correct code. Or the specification is an intermediate result. Then, next possible steps are a decomposition of the components into sets of smaller ones or an extension of the functionality. An extension begins with an informal description of the changed behaviour of the system or of one of its components. Typical extension tasks have the form "The following sequence of actions shall be possible, too". Here, the loop of the development from informal requirements to a verified specification begins again. This loop leads to an incremental design in which it is possible to start with the development of a very simple system and to finish with a complex specification of a distributed system. An overview of the method is presented in [13].

Another aim of this paper is to present how abstraction can be used to come from a formal support of a lower level protocol to a support for a more abstract protocol layer.

After introducing the case study of this paper and the related standards which have to be supported in section 2, we fix the essential requirements of our specification in section 3. Section 4 subsequently illustrates the development of a small initial specification of the system in the

ProCoS specification language SL using a lower level protocol which in section 5 is stepwise extended to a more realistic system. The sections 6 and 7 extend the protocol specified so far with two higher abstraction levels. Section 7 includes the specification of an abstract conference creation element which can be reused in several applications. Some conclusions and final remarks are stated in section 8.

This paper is related to work on synthesis of systems [20] (see also the introduction of [5] for an overview). These approaches mainly support the development of asynchronous systems with their related problems like calculations w.r.t. the size of buffers. By contrast, the approach used here supports systems with synchronous communication.

2 THE CASE STUDY

We focus on the incremental development of a specification of a multi-user multimedia system in which several sites can be connected in one conference to exchange video, audio and data information. The necessary protocols are standardized by the ITU (International Telecommunication Union), but most parts of the protocols are only described in natural language. Some parts are formalized in SDL [1] but these specifications do not form an integral part of the standards. Therefore it is a central goal of this paper to demonstrate how formal approaches can be combined with existing standards. It is shown in a bottom-up approach how the development of more abstract elements for the management of multi-user multimedia sessions can be developed on top of verified, more detailed protocols that are developed earlier. A bottom-up approach is chosen because usually industrial partners have a certain knowledge about the lower level protocols and this knowledge should be used as the initial state.

2.1 A short introduction into the ITU T.120 standards

At a slightly simplified abstract level a multipoint communication system can be considered as a system of n terminals connected by a distinguished component, a so-called *Multipoint Communication Unit* (abbreviated *MCU*). A set of physical point-to-point connections between the terminals and the *MCU* is mapped together in the *MCU* to form one logical multipoint connection. The *MCU* serves as an audio bridge, video- and data switch. Furthermore the *MCU* maintains the status of the multipoint call, i.e. the participants and the rights of the involved sites.

The protocols which are necessary for communications between the components have to be standardized to guarantee interoperability between different providers. Standardized interfaces are also necessary for applications to guarantee that they can be used in different environments. A practically relevant proposal for such protocols is presented by the ITU-T in the T.120 series [6, 10]. An overview of the protocol stack is presented in figure 2.

The T.120-protocols are applicable to a wide range of different networks, e.g. ISDN, PSDN or PSTN. Work is under way to support these protocols also for B-ISDN over ATM. The basic idea is always that a set of point-to-point transport connections is seamlessly mapped together to establish a (virtual) multipoint environment.

The lowest multipoint layer (in which sets of point-to-point transport connections are mapped together) is the "Multipoint Communication Services" (MCS) layer defined in T.122/T.125

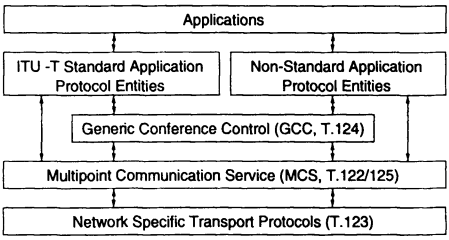


Figure 2 Model of ITU-T T.120 series recommendations

which offers services primarily for the so called *domain, channel* and *token management*. A conference (or domain) consists of a set of multipoint channels for the exchange of data between the conference members. Each conference member can in general send and receive information on channels that he/she has joined. Each conference has a certain set of tokens which are used for the management of resources (e.g. "who is allowed to do what"). The MCS offers so-called *MCS primitives*, abbreviated as MCSPs, to use the services.

On top of MCS resides the "Generic Conference Control" (GCC, defined in T.124) which offers services for conference establishment and termination, conductorship, bandwidth control and so on. The GCC offers so-called *GCC primitives*, abbreviated as GCCPs, to use the services. GCCPs are mapped in the standards onto sequences of MCSPs. Certain other protocol packages that are of general interest – like packages for "Still Imaging" (SI) and "Multipoint Binary File Transfer" (MBFT) – are standardized. But it is also possible for users to develop their own non-standard application protocol entities that reside on top of MCS and/or GCC. End-user applications can be developed which only use services that are defined with these specific protocol entities.

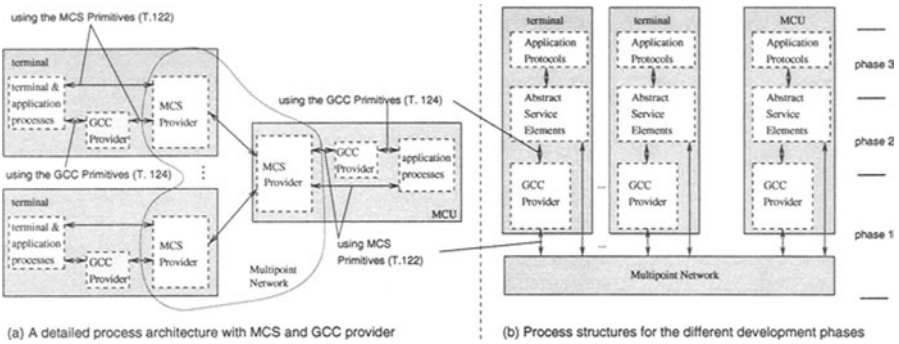


Figure 3 Terminals are connected with a multipoint control unit

A detailed process structure of the terminals and the *MCU* with providers for MCS and GCC primitives is presented in figure 3(a). Note that it is possible to connect several *MCU*s to describe one network. Such a description is omitted here because a decomposition of the *MCU* in a network of $MCUs$ is possible in later steps. Terminal and application processes have to use the

MCSPs and GCCPs in order to communicate with other processes as it is illustrated in figure 3. Processes have to use GCCPs for any action w.r.t. the conference control. For other activities, they have to use MCSPs.

Our aim is a specification of a communication platform for the development of reliable applications which are built by using the protocols explained above. Therefore it is first necessary to develop a correct multipoint communication system based on MCSPs in a first phase and then to extend the system in a second phase in such a way that GCCPs are also supported. Finally, it should be possible to summarize GCCPs and MCSPs to so called *Service Elements* (abbreviated as SEs) [8] which can be used for the development of applications. These SEs can be used to construct applications in a more abstract way than with the primitives of GCC and MCS.

The process structure and the three phases of the development which are sketched in this paper are described in figure 3(b). The MCS providers are combined in 3(b) to a single multipoint network which is responsible for the transmission of the MCSPs.

3 FROM INFORMAL REQUIREMENTS TO FORMAL ONES

A system or specification is called *correct* with respect to certain requirements, if it is *verified* that each requirement is satisfied. So, the most important part in a formal system development is the question of how to get the right initial set of formal requirements.

The process of finding requirements, called *requirement engineering* (see also [17]), starts with an informal description of the structure and the functionality of the desired system. Any kind of description of the desired system ranging from oral descriptions to documents from related projects can be important. An intensive discussion is needed to come from an informal description to informal requirements. Informal requirements are simple sentences in a reduced natural language that can be understood by both customers and developers. These requirements shall give a description of the initial system that we have in mind as precisely as possible. They are developed by customer and developer together.

For our example, a first architectural concept is fixed (see figure 3(b)). Then we choose a first set of requirements which focus on a single conference scenario. Requirements for other types of conferences can be added later.

Examples for typical informal requirements are:

- It is guaranteed that each site in the system can do its desired next action in a limited number of steps, i.e. the system is deadlock-free.
- The conference can only be terminated by the convenor (i.e. the initiating site).
- Each site can leave a conference at any time.
- Each member of the conference has the possibility to invite other sites to join the conference.
- Each site can create new conferences until a certain limit is reached.
- Each member can convene new channels in active conferences until a certain limit is reached. These new channels can be used for subconferences.

These requirements are formalized in our approach in trace logic [21]. Such a formalization and the verification that the requirements are fulfilled are omitted here and can be found in [14].

4 INITIAL SPECIFICATION

Initially, we specify distributed systems using a parallel composition of non-terminating finite automata. Each component is given by one automaton with a designated initial state. A communication can only happen if it is possible as the next communication for both the sender component and the receiver component (fully synchronized communication). The automaton changes its state to the next state after performing a communication.

The automata are composed with the following operators of process algebra (see e.g. CSP [9]):

- alternative composition (+): a *trace* (a sequence of communications) is possible in an alternative composition iff it is possible in one of the composed automata,
- parallel composition (||): the possibility of a trace in a parallel composition of two or more automata requires synchronization on common symbols, e.g. $a.b \parallel b.c = a.b.c$,
- interleaving ($|||$): a trace is possible in an interleaving of n automata iff it is a shuffle of traces each of which is possible in one of the automata. The difference between interleaving and parallel operator is that no synchronization has to take place, e.g. $a.b|||b.c = a.b.b.c + a.b.c.b + b.a.b.c + b.a.c.b + b.c.a.b$.

The development of the specification for the exchange of MCSPs is divided into the following steps for the incremental development:

1. A specification is developed which describes only the creation and termination of the conference. This specification can be seen as a skeleton which is extended in the following steps.
2. The allowed sequences for the conference extension are added. It will be possible to invite other sites to join the conference.
3. The creation of subconferences with the introduction of new channels to the conference will be possible.
4. Data can be sent and received on all conference channels.
5. Tokens are introduced to the conference for the management of resources.
6. The result of previous steps is specification in which each desired action (e.g. creation of a conference, convening a new channel, grabbing a token) is successful. This specification is then extended for unsuccessful actions.

The first automaton resulting from step 1 is presented in figure 4 and describes a simple creation and termination of a conference for a terminal. Communications $> c$ and $c <$ denote *inputs*, communications $c >$ and $< c$ denote *outputs*. The other automata for the network and the *MCU* can be found in [14]. Note that this automaton describes only the possibility to connect terminals T_i to the *MCU* and to terminate this connection without any further action. The suffixes of the communication denote the following: $\_rq$ for request, $\_id$ for indication, $\_rp$ for response, $\_cf$ for confirmation.

The *boxes* in the figure denote the possibility of presenting complex automata in a more structured way. The contents of the boxes which is also presented in the figure have to be substituted in the main automaton. This structured presentation of automata is e.g. related to state-charts [7]. The different numbers of parameters of the box in the main automaton and in the detailed description of the box is explained later in this section.

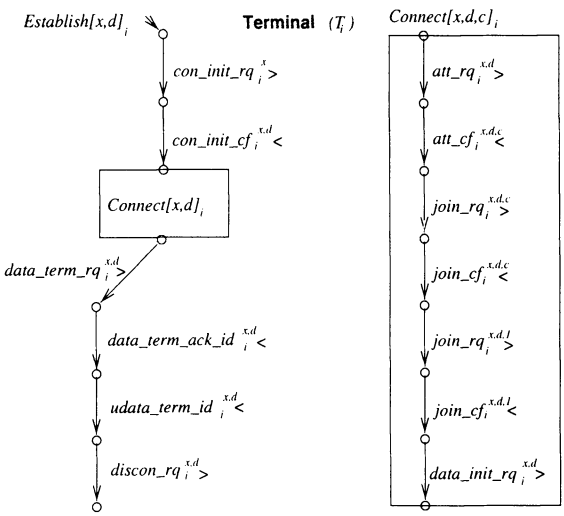


Figure 4 Initial specification of a convener of a conference

The automaton called $Establish[x,d]_i$ denotes the main automaton of a terminal which establishes a conference. The index i is used to describe that this automaton belongs to T_i. Some additional variables are added for each automaton in square brackets to increase the expressive power of the automata. The names of the communications are extended with the following upper indices:

- x denotes the x-th active process of T_i. A site can participate in more than one conference at a time. The variable x is used to distinguish the different conferences in T_i. The range of x is $0 \leq x \leq max_{proc}$ where $max_{proc} + 1$ is the maximal number of active processes.
- d denotes the d-th active conference of the network. The range of d is $0 \leq d \leq max_{conf}$ where $max_{conf} + 1$ is the maximal number of active conferences. The variable d is used to distinguish the different conferences in the network.
- c denotes an MCS channel of the conference. Therefore (d, c) can be seen as a unique identifier of a conference channel. There is a fixed maximal number of channels for each conference, the range of c is $0 \leq c \leq max_{chan}$, where $max_{chan} + 1$ is the maximal number of available channels in the conference d.

The automaton which describes each possible way of establishing a conference for T_i is given as follows:

$$T_i = \underset{x=0}{\overset{max_{proc}}{|||}} \; \underset{d=0}{\overset{max_{conf}}{|||}} \; (Establish[x,d]_i^*)$$

The initial communication for a new conference is *con_init_rq* which is sent to the *Net*. This communication is forwarded to the *MCU* (*con_init_id*, not visible here). Then the *MCU* is connected to the *Net*, a conference is created and a confirm communication (*con_init_cf*) is sent to T_i. The subautomaton (or box) $Connect[x,d,c]_i$ describes that T_i asks (*att_rq*) for a so

called *nodeID channel* which is a local channel that can only be joined by T_i. This channel is used for the conference management w.r.t. T_i. Terminal T_i joins its nodeID channel c and the conference broadcast channel 1 (this channel is joined by each conference member, the name 1 is given in the T.120 standard) such that it is possible for T_i to receive data on these channels. The name of the nodeID channel c is chosen by the *Net*. $Connecting[x,d]_i$ can be described as the alternative composition of getting any possible nodeID channel.

$$Connect[x,d]_i \;=\; \overset{max_{chan}}{\underset{c=0}{+}}\; Connect[x,d,c]_i$$

The conference termination is initiated with a $data\_term\_rq$ communication of the conference convener. This communication is acknowledged by the network ($data\_term\_ack\_id$). The network sends a $udata\_term\_id$ to indicate that the conference is really terminated. This information is acknowledged by T_i with a final $discon\_rq$.

Although finite automata are often powerful enough to describe the typical (i.e. regular) behaviour of many communication protocols, in general complex distributed systems have to store and manipulate data. Therefore finite automata have to be extended with variables to get the expressive power of Turing machines.

The specification language SL developed in the project ProCoS [15] can be seen as such a language based on finite automata and extended with variables. These variables are manipulated by the execution of communications. A communication can happen only if a certain condition over the variables (the *enable predicate*) is fulfilled. After the execution of a communication a new condition (the *effect predicate*) is established. A communication can happen if and only if it is a next possible communication of the automata to which it belongs, and the enable predicate of this communication is fulfilled in each automaton to which it belongs. Thus, the automata describe supersets of possible traces which are further restricted by the enable predicates.

The enable and effect predicates of a communication are summarized in a *communication assertion* of the following form, the part $< comm\_type >$ denotes whether the communication is an input or an output and the datatype of the transmitted value.

com $< communication\_name >$: $< comm\_type >$
when $< enable\_predicate >$ then $< effect\_predicate >$

Two typical examples of communication assertions are presented now. A variable $proc\_ID$ is declared which contains all possible conference identifiers that can be chosen for new conferences.

var $proc\_id$ of set of $0..max_{proc}$ init $\{0,\ldots,max_{proc}\}$

The initial communication is only possible if there is a free process identifier x. This identifier is then removed from the set of usable identifiers. Primed variables refer to the values of the variables after the execution (like in Z [18]), e.g. the effect predicate $(y \neq 0 \rightarrow x' = x + 2) \wedge (y = 0 \rightarrow x' = x)$ specifies that the value of x is incremented by 2 if y is not equal to 0, otherwise the value of x is not changed.

com $con\_init\_rq_i^x$: output of signal
when $x \in proc\_id$
then $proc\_id' = proc\_id - \{x\}$

Enable and effect predicates can be abbreviated as so called *state transformers*. An example is $remove\_ProcessIdentifier(x) \;\equiv\; proc\_id' = proc\_id - \{x\}$. The advantages of such a

structured representation of the communication assertions are that the state transformers can be reused in other communication assertions and that the names can present an intuitive description of the intended behaviour.

The process identifier returns to the set of free process identifiers after the last communication of the conference. The information for this conference is then erased.

com $discon\_rq_i^{x,d}$: output of signal
then $add\_ProcessIdentifier(x) \wedge reset\_conference(x)$

It is then verified for the parallel composition of the first specifications by a combination of a model checking algorithm and invariant proofs for SL that some of the formalized requirements like deadlock-freedom are fulfilled [14]. The specification has to be extended such that all requirements are fulfilled.

5 STEPWISE EXTENSION OF THE INITIAL SPECIFICATION

As mentioned in the introduction, the basic idea of an incremental development technique is to come from small systems by the application of extension rules to larger systems. By *extension* we mean that new functionalities (e.g. new services) are added to the system. We stipulate that each new functionality can be described by a trace which denotes the sequence of communications that shall also be possible in a certain state of the system. The application of extension rules guarantee that this trace is indeed possible and that no deadlock occurs in the extended system

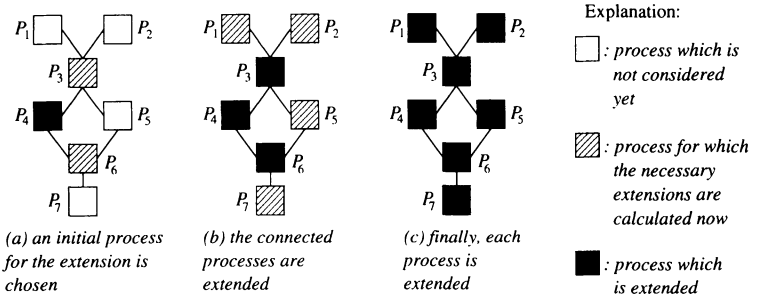


Figure 5 Stepwise extension of a distributed system with new functionality

The extension rules are based on an *extension algorithm* which works as follows. A deadlock-free system, a sequence of communications which shall be possible and an *initial* state in a component (e.g. P_4 in figure 5(a)) in which the sequence shall start are given. It is calculated in the next step which states are *influenced* in each component that is connected via a channel with the first extended component (figure 5(a), P_3 and P_6). If a state of a component can be reached in the parallel composition together with the initial state then such a state is called influenced. In the following steps it is calculated which states are influenced in the other components that are connected with the already observed components (figure 5(b)). Finally (figure 5(c)), the influenced states are calculated for each component and the parts of the trace which are relevant

for each component are added in these states. The final state of the extensions can be any reachable global state of the parallel composition.

It is shown in [12] that a system extended in such a way is deadlock-free again and that the added trace is possible when the initial state of the extension is reached. Note that the sketched extension steps need no knowledge about the global state space, only informations w.r.t. pairs of connected components are required.

The property of deadlock-freedom is emphasized for the extension rules because it is in general difficult to prove and, once it is guaranteed the developer can concentrate on the real tasks, i.e. obtaining the desired functionality of the system.

The MCSPs for the extension of a conference (inviting other sites), sites leaving a conference, and the management of conference channels are added now. A new subautomaton $ActiveE$ is added in the automaton. This subautomaton formalizes all activities during an active conference. The resulting automata are presented in figure 6.

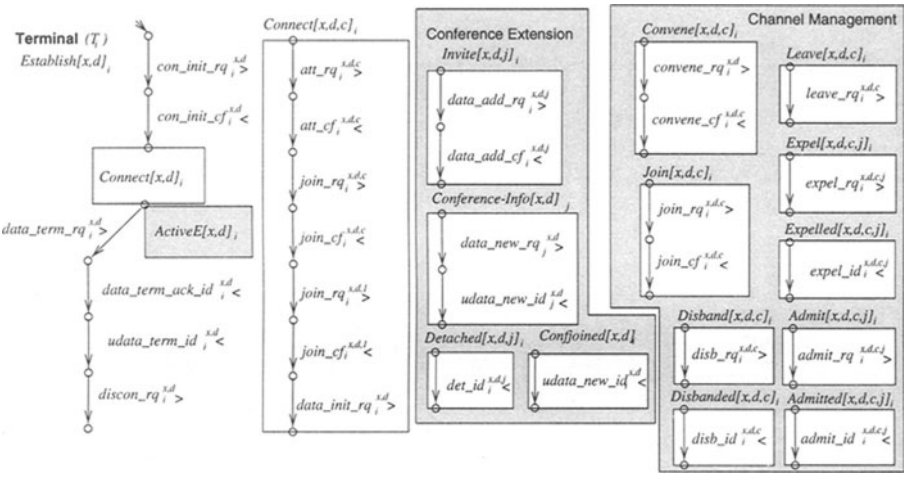


Figure 6 Extension of the specification with the possibility to invite other sites to join the conference and the management of conference channels

The most interesting extension is the possibility for each conference member to invite other parties to join the conference. Such an invitation is started with an add-request ($data\_add\_rq$) which is sent through the *Net* to the *MCU*. Then, the *MCU* invites the desired site to join the conference. Finally, a *data_add_cf* is sent to the inviting site.

A communication $data\_new\_rq$ is used to announce that something has changed at the site (e.g. an additional medium is available). This information is received by all conference members ($udata\_new\_id$) including the original sender. If a previously invited site leaves a conference then each member of the conference is informed by the *Net* (det_id).

During an active conference it is possible for each member to create new channels. The creator of a channel can ask other conference members to join the channel, can expel members

and can disband the channel. Members which have joined the channel can leave this channel at any time. It is possible to create subconferences in such a way.

The automata are composed in the following way. The set I denotes the set of indices of all possible members including the *MCU* ($I = \{1, \ldots, n\} \cup \{mcu\}$).

$$ActiveE[x,d]_i = (\overset{n}{\underset{j=1,j\neq i}{+}} Invite[x,d,j]_i) + Confjoined[x,d,j]_i$$
$$+(\overset{n}{\underset{j=1,j\neq i}{+}} Detached[x,d,j]_i) + Conference\_Info[x,d]_i + (\overset{max_{chan}}{\underset{c=0}{+}} Convene[x,d,c]_i)$$
$$+(\overset{max_{chan}}{\underset{c=0}{+}} Join[x,d,c]_i) + (\overset{max_{chan}}{\underset{c=0}{+}} Leave[x,d,c]_i) + (\overset{max_{chan}}{\underset{c=0}{+}} \overset{}{\underset{j\in I,j\neq i}{+}} Expel[x,d,c,j]_i)$$
$$+(\overset{max_{chan}}{\underset{c=0}{+}} \overset{}{\underset{j\in I,j\neq i}{+}} Expelled[x,d,c,j]_i) + (\overset{max_{chan}}{\underset{c=0}{+}} Disband[x,d,c]_i)$$
$$+(\overset{max_{chan}}{\underset{c=0}{+}} Disbanded[x,d,c]_i) + (\overset{max_{chan}}{\underset{c=0}{+}} \overset{}{\underset{j\in I,j\neq i}{+}} Admit[x,d,c,j]_i)$$
$$+(\overset{max_{chan}}{\underset{c=0}{+}} \overset{}{\underset{j\in I,j\neq i}{+}} Admitted[x,d,c,j]_i)$$

The new possible traces are added step by step using the extension approach explained before. Therefore it is guaranteed that the new specification is also deadlock free. It is shown in [14] that the proofs for unchanged requirements can be extended, it is e.g. possible to reuse information which was produced by the mentioned model checking algorithm. New proofs have to be done only for added requirements.

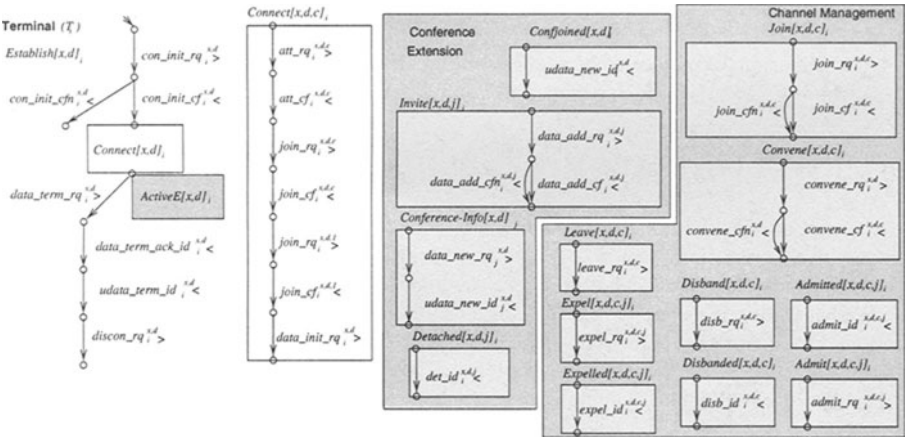


Figure 7 Extension of the specification with possible negative responses

The extensions of the specification with the MCSPs for the token management and the transmission of data are performed in a similar way. They can be found in [14].

It is assumed for the previous specifications that each action is successful. This assumption is dropped now. Negative responses and confirmation messages with the suffixes $\_rpn$ and $\_cfn$ shall be possible. Suffixes $\_rp$ and $\_cf$ belong to positive actions. New requirements are added

Table 1 Supported GCC primitives

name of the GCC Primitive	*abbreviation*	*request*	*indication*	*response*	*confirm*
	informal semantics				
GCC-Conference-Create	GCC_conf_create	+	+	+	+
	used to create new conferences				
GCC-Conference-Add	GCC_conf_add	+	+	+	+
	used to to ask the *MCU* to invite another site				
GCC-Conference-Invite	GCC_conf_invite	+	+	+	+
	used by the *MCU* to invite other sites				
GCC-Conference-Disconnect	GCC_conf_discon	+	+		+
	used to leave an ongoing conference				
GCC-Conference-Terminate	GCC_conf_term	+	+		+
	used to terminate a conference				

that denote in which cases negative responses are possible. Figure 7 presents the resulting extensions. An example of an added communication assertion is the following assertion which describes the behaviour in the case where the conference creation was rejected (enable predicates *true* can be be omitted).

com $con\_init\_cfn_i^{x,d}$: input of reason
then $add\_ProcessIdentifier(x)$

The extension technique can be applied for each desired extension. Therefore it is possible to come to a specification which includes all desired traces w.r.t. the MCSPs.

6 INTRODUCTION OF MORE ABSTRACT PRIMITIVES

Until now each activity is formalized in terms of sequences of MCSPs. Many MCSPs are necessary to describe simple actions like the creation of a conference and many of these activities happen again and again. Therefore it is a good idea to introduce more abstract service primitives that comprise sets of MCSPs. This is done in the GCC protocol defined in T.124 with the GCCPs. It encompasses functions such as conference establishment and termination, managing the roster of nodes participating in a conference, managing the roster of applications and their capabilities, coordination of conference conductorship, as well as other functions.

We use the informal textual descriptions of the GCCPs in the standard T.124 to identify the states in which extensions are needed to support the GCC primitives. The supported GCC primitives and their abbreviations are presented in table 1. It is also shown what kinds of primitives (request, indication, response, confirm) are supported. We follow the approach to the decomposition of a specification as described in [11, 13] and introduce local communications, i.e. communications which only belong to one automaton and do note need synchronization with other automata. These communications will be the interface to the new processes.

The GCC primitives of table 1 are added in figure 8. The subautomata w.r.t. the channel man-

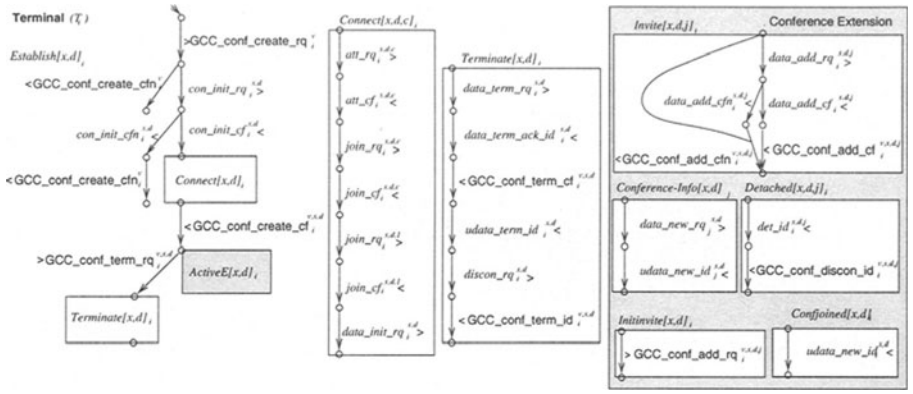


Figure 8 Terminal automaton extended with GCCPs

agement are omitted because they are not changed. Another extension rule which is explained in [11, 13] and preserves deadlock freedom is applied several times.

7 SERVICE DEVELOPMENT WITH ABSTRACT PRIMITIVES

The aim of this section is to develop a first *service element* (abbreviated SE) for application protocols based on GCC and MCS primitives. SEs summarize sequences of GCC and MCS primitives to more abstract actions which can be used in application protocols. The motivation is that it should not be necessary for a developer of new services to describe similar activities again and again. SEs describe e.g. the creation of a new subconference with a fixed number of channels or guarantee that certain actions at certain sites are synchronized. Therefore it is also possible to call an SE in another sense a module or procedure of a programming language which can be reused (called several times) in applications.

The idea of summarizing possible actions of a system at the lower level of abstraction to one action at the higher level is established in many different areas like the ISO-OSI reference model.

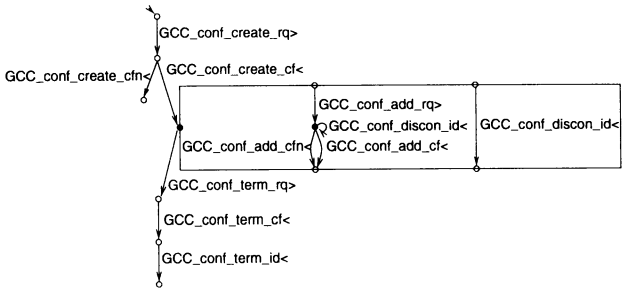


Figure 9 Possible sequences of GCCPs for terminals

To obtain the information about possible sequences of GCC primitives, the information about possible MCS primitives is hidden in the automata presented so far. The final result for a terminal is described in figure 9. There are special black states "•" in the figure that denote states in which it is possible to send and receive MCS primitives.

We can use the summary presented in figure 9 to determine the possible sequences of GCC primitives. It is e.g. possible to calculate whether a SE describes incorrect (impossible) sequences of GCC primitives. One example for an impossible trace is GCC_conf_create_rq.GCC_conf_term_rq because if we look at the automaton in figure 9 then we can observe that there has to be at least a communication GCC_conf_create_cf between these two communications.

The automata can also be used to observe that GCC primitives which belong to the same "action" (e.g.: GCC_conf_add_rq, GCC_conf_add_cf) can be interrupted by GCC primitives from other actions like GCC_conf_discon_id.

The automaton in figure 9 can be used for the development of an SE in the following way:

1. An informal description of the new SE is given. This informal description is translated into a first set of informal requirements.
2. These informal requirements are formalized.
3. A first specification of the new SE is written. It is checked whether the described sequences are possible in the automata presented in figure 9. If impossible traces are specified then a new specification has to be developed.
4. Figure 9 is also used to determine whether the specification is complete or not. It is checked whether all possible inputs in all possible states are taken into account.
   If this is not the case then the informal requirements are extended and the next loop of the incremental development begins in step 1. If all inputs are considered then the following step is executed.
5. Verify that the specified SE fulfils the formal requirements.

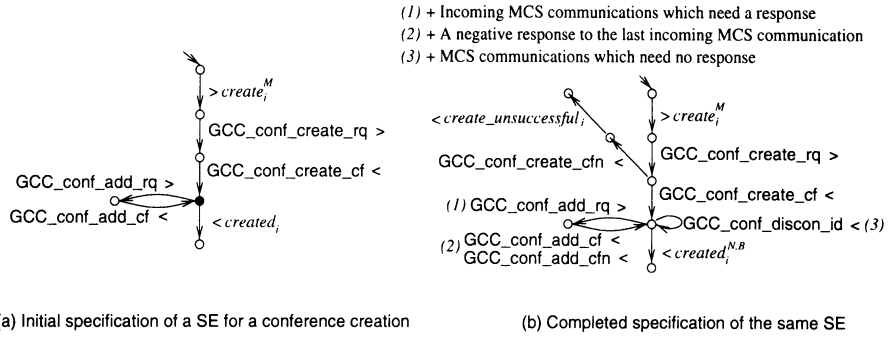
7.1 A service element for the creation of a new conference

The following text describes the development of an SE for the creation of a new conference: It is assumed that the new SE is initiated with a communication $create_i^M$ which is sent by an application at site i to create a conference with the initial set M of conference members. A communication $created_i$ is used to indicate that the SE is terminated.

The automaton for the first specification of the SE for the conference creation is given in figure 10(a) (some parameters are omitted, they can be taken from the communication assertions in [14]).

If we look at the automaton in figure 10(a) and at the automaton in figure 9 then we can observe that in many states other communications are possible and therefore more design decisions have to be made. The following questions have to be answered:

- What happens if an invited site is not reachable?
- What happens if an already connected site disconnects during the execution of the SE?
- What happens if an already connected site sends data or other information during the execution of the SE (note that black states in which MCS primitives may arrive are visited in figure 10(a))?



(a) Initial specification of a SE for a conference creation

(b) Completed specification of the same SE

Figure 10 Development of a SE for the creation of a conference

We choose the following solution. The first two questions are answered in the following way: there is an output parameter N for the SE which describes the set of all sites that are really in the conference after the execution of the SE. Information which arrives during the execution of the SE is stored in a buffer which is the second output parameter B. Incoming communications which need an answer get automatically a negative response. The extended automaton of the specification is presented in figure 10(b).

8 CONCLUSIONS AND FINAL REMARKS

A quite complex layered specification of a protocol for a multi-user multimedia system was developed in the previous sections. The incremental development technique explained in [11, 12, 13] is applied to expand a small initial specification to a complex specification in the final section. It is shown in [14] how informal requirements are formalized and how it is possible to change and add requirements in later development steps. The incremental development supports the idea that not all requirements can be known at the initial state of the development. Because of the formal background each step leads to a verified specification. It is possible to reuse specifications and parts of the verification of requirements that are changed only slightly or not changed at all in the development.

Another result of this case study is that protocols described in natural language in the standards are formalized. Several cases were detected in which the informal standards do not offer a unique description of the desired sequences of possible communications. The problem is that in the standards only typical sequences of communications w.r.t. one action like the conference creation are described and that the interaction of several actions is not explicitly solved.

Our case studies show that formal methods of ProCoS and CoCoN are suitable for problems from the telecommunications area. Experiences of academic case studies [16] can be scaled up to industrial-size problems. Nevertheless further research is needed to complete each part of our method, e.g. the idea of reusing proofs which are produced by a combination of a model checking algorithm and an interactive theorem prover has to be studied in more detail. The development of tools to support the verification that requirements are fulfilled and for the support of the incremental development by a designer is in progress. Case studies in other application areas are under development.

Acknowledgements. The author thanks H. Tjabben of Philips Research Laboratories Aachen and E.-R. Olderog and the other members of the "semantics" group in Oldenburg for helpful discussions.

References

[1] F. Belina, D. Hogrefe, The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems 16 (1988/89) 311-341, North-Holland
[2] D. Bjørner, H. Langmaack, C.A.R. Hoare, ProCoS I Final Deliverable, ProCoS Technical Report ID/DTH db 13/1, January 1993
[3] D. Bjørner et al., A ProCoS project description: ESPRIT BRA 3104, Bulletin of the EATCS, 39, 1989
[4] J. Bowen et al., Developing Correct Systems, Bulletin of the EATCS, June 1993
[5] D. Y. Chao, D. T. Wang, An Interactive Tool for Design, Simulation, Verification, and Synthesis of Protocols, Software - Practice and Experience, Vol. 24(8), 1994
[6] W. J. Clark, J. Boucher, Multipoint communications - the key to groupworking, BT Technol J, Vol 12, No 3, July 1994
[7] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming No. 8, 1987
[8] G.J. Heijenk, X. Hou, I.G. Niemegeers ,Communication Systems Supporting Multimedia Multi-User Applications, IEEE Network Jan./Feb. 1994
[9] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, London, 1985
[10] ITU-T Recommendation T.120 Series, Transmission Protocols for Multimedia Data, Geneva, 1994, ITU General Secretariat
[11] S. Kleuker, A. Kehne, H. Tjabben, Provably Correct Communication Networks (CoCoN), Philips Research Laboratories Aachen, Technical Report, 1123/95, 1995 ftp://ftp.informatik.uni-oldenburg.de/pub/procos/cocon/lab1123.ps.Z
[12] S. Kleuker, A Gentle Introduction to Specification Engineering Using a Case Study in Telecommunications, in P.D. Mosses, M. Nielsen, M.I. Schwartzbach (eds.), Proc. TAPSOFT "95, LNCS 915 (Springer), 1995
[13] S. Kleuker, H. Tjabben, The Incremental Development of Correct Specifications for Distributed Systems, in M.-C. Gaudel, J. Woodcock (eds.), Proc. FME "96, LNCS 1051 (Springer), 1996
[14] S. Kleuker, H. Tjabben, A Formal Approach to the Development of Reliable Multi-User Multimedia Applications, Philips Research Laboratories Aachen, Technical Report, 1168/96, ftp://ftp.informatik.uni-oldenburg.de/pub/procos/cocon/mumu.ps.Z
[15] E.-R. Olderog et al., ProCoS at Oldenburg: The Interface between Specification Language and OCCAM-like Programming Language. Technical Report, Bericht 3/92, Univ. Oldenburg, Fachbereich Informatik, 1992
[16] E.-R. Olderog, S. Rössig, A Case Study in Transformational Design on Concurrent Systems, in M.-C. Gaudel, J.-P. Jouannaud (eds.), Proc. TAPSOFT "93, LNCS 668 (Springer), 1993
[17] H.A. Partsch, Specification and Transformation of Programs, Springer, 1990
[18] B. Potter, J. Sinclair, D. Till, An Introduction to Formal Specification and Z, Prentice-Hall, 1991
[19] S. Rössig, A Transformational Approach to the Design of Communicating Systems, PhD thesis, University of Oldenburg, 1994
[20] P. Zafiropulo et al., Towards Analyzing and Synthesizing Protocols, IEEE Transactions on Communications, Vol COM-28, No. 4, April 1980
[21] J. Zwiers, Compositionality, Concurrency and Partial Correctness - Proof Theories for Networks of Processes and Their Relationship, LNCS 321 (Springer), 1989