Telecommunication Systems 9 (1998) 207–221

Modular specification and verification of XTP

Peter Herrmann and Heiko Krumm

Department of Computer Science, University of Dortmund, D-44221 Dortmund, Germany
E-mail: {herrmann,krumm}@ls4.informatik.uni-dortmund.de

Received January 1997; in final form February 1998

The transfer protocol framework supports the formal specification and verification of data transfer protocols. It consists of generic specification modules and theorems. Compositions of specification module instances result in well-structured specifications which describe a protocol, the medium used, and the service provided by means of TLA formulas. The protocol verification is based on the proof of the logical implication between protocol and service specification. Due to the modular structuring of the specifications, this proof can be decomposed into a set of subimplications which correspond directly to theorems of the framework. Therefore, the development of formal specifications as well as the protocol verification can be reduced to the instantiation and arrangement of framework elements. The flexibility of the framework opens its application for a broad spectrum of data transfer protocols. We outline the principles of the framework and concentrate on its application to the high-speed transfer protocol XTP. Because of the framework support, the formal modeling and analysis of this modern and function-rich protocol was manageable and identifies deficiencies of the current protocol definition clearly.

1. Introduction

The formal verification of complex real-life protocols is a time-consuming task in general. Very often suitable formal specifications of the protocol, of the service provided, and of the service used do not exist and have to be developed from informal descriptions. Furthermore, considerable work is necessary to perform the verification itself, even if automated tools are applied. Mostly fully automated reachability-analysis and model-checking tools run out of memory due to the large number of reachable states. Thus, simplifications and abstractions have to be developed in order to reduce the set of relevant states. If theorem-proving-based verification methods are used, practitioners have carefully to design the outlines, strategies, and lemmas (particularly invariants) of complex deduction processes. Therefore, many approaches exist which concentrate on the enhancement of tools and methods.

In addition to methods and tools, a third type of support is possible: domain-specific frameworks can be provided, which document the results of general investigations of the principal objects and interdependences of specific application domains. We analyzed the domain of data transfer protocols in order to identify the relevant

protocol mechanisms, service properties, and logical relationships developing libraries of re-usable specification modules and theorems. The specification modules facilitate the development of formal specifications substantially. Protocol and service specifications can be composed of instances of library modules. Moreover, the structure of these compositions already outlines the structure of proofs. The proof that a protocol as a whole provides for a certain service can be decomposed into a series of proofs, which consider that certain service properties are provided by certain subsystems of the protocol. Last, these subsystem proofs can be infered directly from instances of the theorems. Since the theorems are already proven, the developer of a particular protocol has to ensure only, that subsystems, proof goals, and theorems are configured consistently. We call this collection of generic specification modules "Transfer Protocol Framework" since it supports the formal modeling and verification of data transfer protocols in general. While it may advantageously be applied to traditional link, network, and transport layer protocols, it is of special interest for the design of modern high-speed communication protocols which focus on efficient and flexible combinations of transfer protocol mechanisms (e.g., XTP [15,19], MSP [9,11]).

Due to the framework support, manual and well-understood proofs of complex (real-life) protocols are possible. The approach, however, does not exclude the use of tools. Specialized tool support is provided by a theorem manager which facilitates the selection of theorems and performs the necessary consistency checks [3]. Moreover, the theorem manager has an interface to the theorem prover OTTER [14] which is used to verify predicate-logic-based obligations of consistency checks.

The complete transfer protocol framework is available via WWW (`http://ls4-www.informatik.uni-dortmund.de/RVS/P-TPM`). A detailed description is contained in [4]. The principles are outlined in [6].

The specification module and theorems of the framework are based on TLA [8], a linear-time temporal logic which is primarily devoted to the description of state transition systems and the reasoning with their safety and liveness properties. We apply a special compositional TLA specification style which we call cTLA [5,13]. In particular, cTLA supports the structuring of systems into concurrent and interacting processes.

The paper gives a short introduction into the transfer protocol framework and the high-speed transfer protocol XTP [15,19] first. Thereafter it reports on the application of the framework which supports a manageable analysis of the unicast services of XTP. In more detail, the service provided by XTP, the protocol XTP, and the service used by XTP have been described formally by compositions of instances of specification modules of the framework. The protocol verification is performed by combining theorems of the framework to form the proof of the overall implication of protocol and service used implying that the appropriate services are provided. Due to the framework support, the tasks of formal specification and verification could be performed efficiently within three weeks. The verification showed XTP to be correct with respect to our formal specifications. Moreover, we identified some deficiencies of the (informal) XTP definition which can result in faulty implementations.

2. The transfer protocol framework

The transfer protocol framework is a collection of generic specification modules and theorems supporting the specification and verification of transfer protocols by reusable elements. The specification elements are modular definitions of process types. The verification elements are implications between process systems and processes. Process types and theorems are defined using cTLA [5,13]. This TLA style supports the composition of systems from resource-oriented processes as well as from constraint-oriented processes (cf. [17]). As in LOTOS [7] or in I/O automata [12], the processes of a system interact via joint actions.

Primarily, the framework supports the specification of the service provided, the specification of the service used, and the specification of the protocol. The service provided is described by a composition of service constraint processes (SCs), the service used is described by a composition of medium constraint processes (MCs), and the protocol is described by a composition of protocol mechanism processes (PMs). The protocol verification proves that the conjunction of PMs and MCs implies the set of SCs. This proof can be decomposed into proofs of PM/MC-subsystems implying single

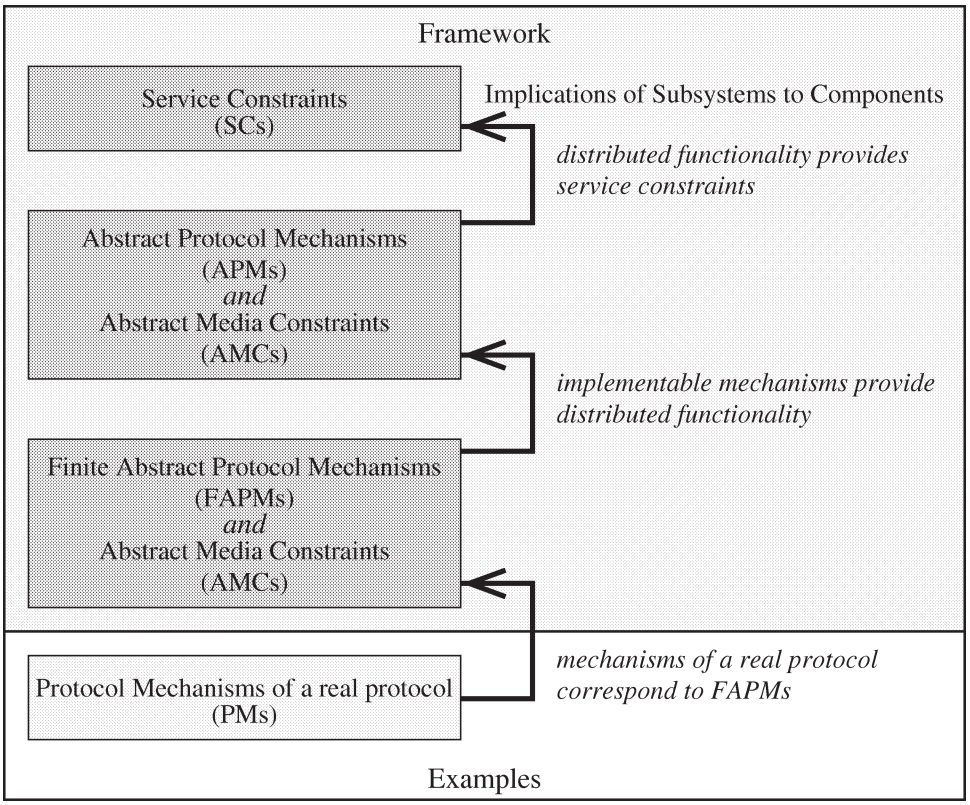

Figure 1. Framework and layers.

SCs. The framework should support these proofs by the provision of corresponding theorems.

The framework, however, does not provide for direct implications between PM/MC-subsystems and SCs since a large set of specialized theorems would be necessary to cover the wide variety of protocol mechanism combinations. Actually, the gap between protocol and service is bridged by intermediate abstractions of protocol mechanisms and medium constraints which are grouped into layers. Figure 1 gives an overview of the layers of the framework. At the top of the framework are process types which correspond to SCs. At the bottom, the framework supplies finite abstract protocol mechanisms (FAPMs) and abstract medium constraints (AMCs). They consist of generic process types, instantiations of which directly correspond to PMs and MCs. In the middle, abstract protocol mechanisms (APMs) form abstractions of FAPMs. The theorems of the framework are implications between subsystems of a layer and processes of the next higher layer. Thus, the implications of a protocol verification can be proven by transitively combining theorems.

Presently, the framework consists of 28 SCs, 44 APMs, 14 AMCs, and 47 FAPMs. There are 44 theorems linking FAPM/AMC systems with APMs. 31 theorem link APM/AMC systems with SCs.

3. XTP

The high-speed transport protocol XTP (eXpress Transfer Protocol) [15,19] provides a broad spectrum of transport services supporting the data transfer requirements of various distributed applications. Therefore it is designed in a modular way. It consists of protocol functions (cf. [2,10]) which are mostly asynchronous to each other. This facilitates the use of concurrent systems (e.g., transputers) for implementation (cf. [1]). Furthermore, XTP facilitates the dynamic adaption of protocol functions depending on the particular needs of the protocol user (cf. [18]), who can select different techniques for connection management, error control, and flow control. Virtual unicast and multicast connections are supported as well as datagram-oriented data transfer. While connections are opened implicitly without confirmations in general, the user can select different closing modes including implicit disconnects, disconnects caused by timeouts, and graceful releases. Detection of corrupted user data can be switched off as well as the remedy for lost data. Further, different modes for triggering error reports and the repetition of data transfer are available. XTP is the first protocol offering a twofold flow control. End-to-end flow control is supported by a sliding window credit mechanism while the flow control of the basic service is realized by interpacket gaps. Both flow control mechanisms can be switched off, too.

The protocol data formats of XTP facilitate hardware realizations. In the current version 4.0 [19] an XTP packet consists of a packet header with a fixed structure and length (32 bytes) and a payload segment, the structure and length of which depend on the packet type. The header contains relevant information, i.e., a connection identifier, connection mode data, a sequence number of the data unit to be sent next, a sequence

number of a connection mode negotiation handshake, a packet priority indicator, a frame check sequence, and a packet length indicator. Seven different types of XTP packets are defined. The control of a connection is provided by three packet types which are used to acknowledge delivered data, to report transfer errors, and to negotiate a new mode of the connection. The other four types support the transfer of information. Packet types are available to open new connections (connection packets may contain user data, too), to join an existing multicast connection, to send user data, and to report severe protocol errors.

4. Arrangement of XTP specifications and verifications

The formal specification of XTP had to consider the informal XTP description [19], which does not contain descriptions of the communication service supplied. In the first step we examined the XTP description. We identified service constraints and modeled them by instantiations of SCs of the framework. Altogether, 27 SC instances were derived, the composition of which forms the service specification. In the same way we developed the protocol specification by arranging and composing 60 Finite Abstract Protocol Mechanisms (FAPMs) in accordance with the XTP description. Further, we designed an intermediate abstract protocol specification by arranging and composing 58 Abstract Protocol Mechanisms (APMs). In the third step we proved that the protocol specification implements the service specification. We structured this proof into 85 lemmas. By means of 58 lemmas we proved that each APM of the intermediate specification is fulfilled by a subsystem of the protocol specification. Each of the remaining 27 lemmas verifies that an SC is implied by a subsystem of APMs. The verifications of the lemmas were performed utilizing theorems of the framework which could easily be selected depending on the structure and the parametrization of the subsystems. Further, we proved the consistency of the different process compositions.

The process types forming the XTP protocol specification are arranged and composed according to figure 2. The protocol specification consists of an infinite set of descriptions modeling single connections. A connection description is composed of specifications of various data channels, each modeling a different quality of service, and of a connection management specification. The data channels are specified by the data transfer FAPMs and by Abstract Media Constraints (AMCs). The data transfer FAPMs describe protocol functions which are necessary to provide correct data transfer between distributed sites.

The connection management consists of two parts modeling the transfer of signals and the coordination of the channels. The transfer of signals is described by a set of signaling channels which are modeled by data transfer FAPMs and AMCs as well. Since XTP handles the signal transfer by means of own acknowledgement and remedy mechanisms, we had to separate the modeling of the signal transfer from that of the data transfer and therefore introduced independent signaling channels. The protocol functions necessary to coordinate the signaling and data channels in the dif-

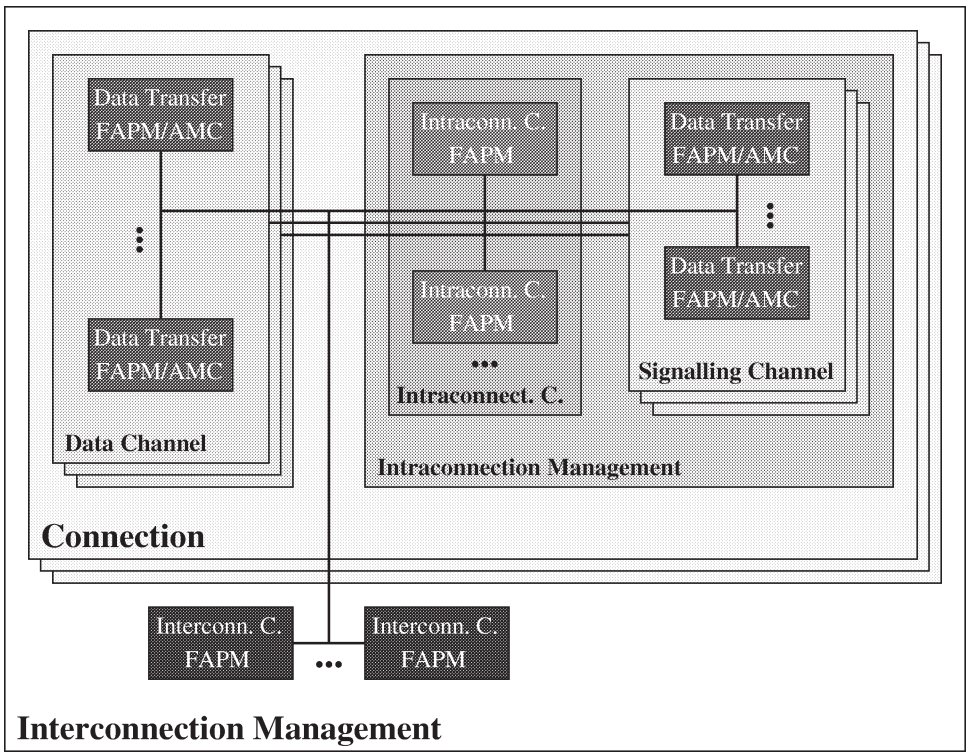

Figure 2. Organization of a protocol specification.

ferent connection states are specified by a set of so-called intraconnection coordination FAPMs.

Since during the lifetime of the XTP protocol an arbitrary number of connections may be used, the protocol specification contains an infinite number of connection specification instances. The interaction of different connections is modeled by a third group of FAPMs, the interconnection coordination FAPMs.

5. Service specification

As an example, we will sketch the arrangement of the SCs regarding the transfer of signals used by the connection management. Since the transfer requirements vary within different service elements, we defined four different channel specifications. The channels `'First'` and `'Last'` are used to transmit the first and last signal of a connection only. Intermediate signals (e.g., negotiations to change the quality of the data channels) are transmitted via the channel `'Active'`. The signal transfering a connection abort triggered by the protocol entities is transfered via the channel `'ProviderAbort'`. Table 1 lists the SCs of which the four channel specifications are composed.

Table 1
SCs to model service specifications of signal transfer channels.

SC	'First'	'Active'	'Last'	'ProviderAbort'
SDUId	⋆		⋆	⋆
SDUIdSynch		⋆		
Corruptions	⋆		⋆	
CorruptionsSynch		⋆		
Duplicates	⋆	⋆	⋆	
Reorderings	⋆	⋆	⋆	
Phantoms	⋆	⋆	⋆	⋆
Capacity	⋆		⋆	⋆
CapacitySynch		⋆		
ShowDeliverySynch		⋆		
LiveIn				⋆

SC Corruptions

```
PROCESS Corruptions ( usd : Any ;
                          Set of data units to be transmitted
                      tc :  SUBSET(usd × usd) )
                          Relation of corruptions which are tolerable
IMPORT Constants;
BODY
  VARIABLES
    buf : SUBSET(key × usd); Buffer of transfered data units
  INIT ≙ buf = ∅;
  ACTIONS
    Rq (krq : key; d : usd) ≙ Data transfer request
      buf′ = buf ∪ {(krq,d)} ;

    In (krq : key; d : usd) ≙ Data transfer indication
            1.) Delivery of phantoms
      ( krq = "notsent" ∨ ∀ e ∈ usd :: ((krq,e) ∉ buf) ∨
            2.) Delivery of data units at most corrupted within limits of tc
        ∃ e ∈ usd :: ((krq,e) ∈ buf ∧ (e,d) ∈ tc ) ) ∧
      buf′ = buf ;
  END
```

Figure 3. The SC *Corruptions*.

The service constraints *SDUId* and *SDUIdSynch* are basic constructs supplying the ordered stream of signal data. The SCs *Corruptions*, *CorruptionsSynch*, *Duplicates*, *Reorderings* and *Phantoms* model certain classes of data transfer errors. The capacity of the channels is described by *Capacity* and *CapacitySynch*. The SC *ShowDelivery-Synch* specifies the occurrence of implicit data confirmations. *LiveIn* guarantees that

XTP service constraint XTPCorruptions

```
DWord ≙ { x | x ∈ Nat ∧ x < 2^64 };
SigdTypes = { "FirstDuplex", "FirstSimplex", "NewQ", "WClose",
              "Final", "UAbort" };

Structure of signals to be transmitted
sigd ≙ [ sTYPE : SigdTypes, sNOERR : {0, 1}, sNOCHECK : {0, 1},
         sNOFLOW : {0, 1}, sRES : {0, 1}, sFASTNAK : {0, 1},
         sSEQ : DWord ];
Identity relation
id ≙ { (x, x) | x ∈ sigd };
```

```
Adaption of SC Corruptions to XTP
XTPCorruptions : Corruptions ( sigd, id );
```

Figure 4. The XTP service constraint *XTPCorruptions*.

transmitted data will be delivered eventually. The specifications of the four channels are derived by composing subsets of these SCs (see table 1).

In figure 3, we outline the arrangement of the SC *Corruptions*, which models the restriction of corrupted data units or signals. The type `usd` denotes the set of data units or signals to be transfered. During the data transfer each data unit is accompanied by a unique sequence number describing the transmission order. `key` identifies the set of available sequence numbers. Since in service specifications aspects of distribution are transparent, *Corruptions* uses only one global set variable `buf`, which models the set of data transfered by the service.

The transmission of new data units is modeled by the action `Rq`. This action contains two action parameters `d` and `krq` describing the value and the sequence number of the sent data unit. `Rq` models the storage of the data unit in the buffer `buf`.

The delivery of a data unit to the receiver is modeled by the action `In`. This action may only be executed if the delivered data unit, identified also by `krq` and `d`, either is a phantom (represented by the special sequence number `'notsent'`) or corresponds with a data unit in `buf` the value of which is related to the delivered unit in accordance with the relation `tc`. By `tc` classes of tolerable corruptions are described. Thus, the SC states that delivered data are either phantoms or are at most corrupted within the limits of `tc`.

The SC *Corruptions* is adapted to the XTP service constraint *XTPCorruptions* by the parametrization listed in figure 4. The formal parameter `usd` describes the set of data units to be transmitted. It is instantiated by the type `sigd` of XTP signals. `sigd` is a record containing a service element identifier and various quality of service parameters. Since corruptions of signals cannot be tolerated at all, `tc` is replaced by the identity relation $\{(x, x) \mid x \in \texttt{sigd}\}$.

6. Protocol specification

Again, we will use specifications of the signal transfer channels to outline the arrangement of the XTP protocol mechanism specifications. The FAPMs used to model the protocol mechanisms realizing the four channels `'First'`, `'Active'`, `'Last'`, and `'ProviderAbort'` are described in table 2. The FAPMs *SBufferKey*, *SBufferUsd*, *RBufferKey*, and *RBufferUsd* model the handling of sequence numbers and the buffering of user data in the transmitter and receiver protocol entities. The notification of the data transmitter by the receiver about data received properly or corrupted is described by the FAPMs *SAcknowledge*, *SReject*, *RAcknowledge*, and *RReject*. The remedy for data duplications, reorderings, and phantoms by refusing the delivery of defective data units is modeled by *RDuplicates*, *RReorderings*, and *RPhantoms*. *SSendInOrder* is used to describe the transmission of data in a certain order. The notification of the transmitter about data transfered properly is modeled by the FAPM *SShowDelivery* while *SSynchronize* is used to adapt different sequence number assignment methods. The FAPMs *SLiveMRq*, *RLiveARq*, and *RLiveIn* guarantee that data units are eventually transmitted, notified, and delivered. These FAPMs are composed in order to specify the four signaling channel classes of XTP (table 2).

As an example we sketch the parametrization of the FAPM *SBufferKey* (figure 5) for the channels `'First'`, `'Active'`, and `'Last'`, which models the handling of sequence numbers of user data and signaling information and the segmentation and blocking of data in the transmitter. The set variable `sbk` models the buffer of sequence

Table 2
FAPMs to model protocol specifications of signal transfer channels.

FAPM	'First'	'Active'	'Last'	'ProviderAbort'
SBufferKey	⋆	⋆	⋆	⋆
RBufferKey	⋆	⋆	⋆	⋆
SBufferUsd	⋆	⋆	⋆	
RBufferUsd	⋆	⋆	⋆	
SAcknowledge	⋆	⋆		
RAcknowledge	⋆	⋆		
SReject			⋆	⋆
RReject			⋆	⋆
RDuplicates	⋆	⋆	⋆	
RReorderings	⋆	⋆	⋆	
RPhantoms	⋆	⋆	⋆	
SSendInOrder	⋆	⋆	⋆	
SShowDelivery		⋆		
SSynchronize		⋆		
SLiveMRq				⋆
RLiveARq				⋆
RLiveIn				⋆

numbers of data segments in the transmitter buffer. `cRq` contains the sequence number which will be assigned to the data unit to be sent next. `fkey` is the finite set of data unit sequence numbers.

The action `Rq` models a data transfer request. A data unit `d`, sent by the service user, is provided with the sequence number contained in `cRq`. `cRq` is incremented. The segmentation of user data is modeled by the functions `skk`, `skn`, `skm`, and `usdsize`. The sequence numbers of the segments are added to the set `sbk`. The old sequence number $(\texttt{cRq} + 1 - \texttt{ms})$ mod `ws`, which is no longer relevant for the transmission, is deleted. The action `MRq` models the transfer request of a Protocol Data Unit (PDU) to the medium. The action parameter `p` is the PDU transmitted by the transfer request. The parameter `krq` is a set of sequence numbers which are contained in the PDU. A PDU can contain only data units, the sequence numbers of which are elements of `sbk`. The action `MRq` states further that the frame check sequence of `p` must be initiated properly and a block may contain at most `mb` segments.

In dependency of the channels `'First'`, `'Active'`, and `'Last'` the process type parameters of *SBufferKey* are instantiated to develop the XTP protocol mechanism specification. `pdu` and `pci`, which describe the sets of PDUs and their Protocol Control Information (PCI), are replaced by the datatype `XTPpdu` defined previously which models the structure of the PDUs and PCIs in XTP. As in the SC *Corruptions* (see section 5) the parameter `usd` is replaced by the signal type `sigd`. Since, for simplicity, we model PDUs and their PCIs by the same data type, the function `spci` describing the position of the PCI in a PDU is instantiated by the identity function. By the parameter `encpdu` frame checking is modeled. It refers to the set of PDUs which pass a frame check. Due to the corresponding XTP protocol functions this set contains all PDUs, the frame check sequence of which are identical to the result of the XTP frame checking function.

The instantiation of the parameter `skey` modeling the set of sequence numbers of signaling data transfered in the PDU is a little more sophisticated and depends on the channel type. In the type `'Active'` it is replaced by the sequence number of a connection mode negotiation handshake which is a structure in the PDU header. Since in channels of the other types at most one PDU is sent (except for retransmissions), 0 is the only sequence number used, since that is the sequence number which is always assigned to the first data unit transmitted. XTP uses four bytes to represent sequence numbers of connection mode negotiations. Thus, we set the parameter `ir` denoting the total number of sequence numbers to 2^{32}. In the sliding window protocol mechanism with selective repeat, one can simultaneously use all but one of the available sequence numbers (cf. [16]). Therefore the parameter `ms` specifying the number of active sequence numbers is set to $2^{32} - 1$. Finally, the parameter `mb` specifies the maximum number of data units transfered in a PDU. Its value depends on the particular protocol implementation and on the properties of the basic service used.

7. Verification

After creating service and protocol specifications we have to prove that the protocol implements the service. This task is performed utilizing theorems of the framework. As explained in section 2, the refinement proofs are performed in two steps. First, we prove that the service specification is implemented by an intermediate specification consisting of Abstract Protocol Mechanisms (APMs). Second, we verify the refinement between the protocol specification and the intermediate specification. Therefore, we have to create the intermediate specification by arranging APMs. This task is relatively easy since the APMs use process type parameters similar to the FAPMs. For example, the parameters of the APM *Corruptions* are instantiated by the same values as the FAPM (figure 5) except for `mseg` and `is`, which are not used in the APM.

We will now outline the proof that the protocol implies the service specification of the signaling channels which is formed by SCs (table 1). We have to prove that the SCs of all four channel classes `'First'`, `'Active'`, `'Last'`, and `'ProviderAbort'` are implied by subsystems of the intermediate specification. We sketch the proof of the SC *Corruptions* (figure 3). This proof is performed using the theorem shown in figure 6. The theorem states that a subsystem `Sys` of APM processes *SBufferKey*, *SBufferUsd*, *RBufferKey*, and *RBufferUsd*, as well as the AMCs *MSDUId*, *MCorruptions*, and *MPhantoms* implies the service constraint *Corruptions*. The subsystem `Sys` is defined by a conjunction of process instances and a coupling constraint $\mathtt{CC}_{\mathtt{Corruptions}}$. $\mathtt{CC}_{\mathtt{Corruptions}}$ describes which actions of the processes of `Sys` are performed jointly. The implication depends on the consistent parametrizations of the APMs, AMCs, and the SC which is described by the condition `Pars`.

For the application of the theorem we have to check first that `Sys` is a subsystem of our intermediate system, i.e., each process of `Sys` has to occur in the intermediate system. Furthermore, the coupling constraint $\mathtt{CC}_{\mathtt{Corruptions}}$ has to be implied by the intermediate system.

Second, we have to check that the parameters of the process instances of the intermediate system meet the parameter condition `Pars` of the theorem. In our example, the parameter `mtc` of the AMC *MCorruptions* is instantiated by the set $\{(\mathtt{p},\mathtt{q}) \mid \mathtt{p} \notin \mathtt{encpdu} \vee (\mathtt{p} = \mathtt{q})\}$, guaranteeing that corruptions can be detected by the receiver entity. Thus, the instantiation of `Pars` is

$$\begin{aligned}&\{(\mathtt{p},\mathtt{q}) \mid \mathtt{p} \notin \mathtt{encpdu} \ \vee \ (\mathtt{p} = \mathtt{q})\}\\ &\quad \subseteq \{(\mathtt{p},\mathtt{q}) \mid \mathtt{p} \notin \mathtt{encpdu} \ \vee \ (\mathtt{skey}[\mathtt{spci}[\mathtt{p}]] = \mathtt{skey}[\mathtt{spci}[\mathtt{q}]]\\ &\qquad\qquad \wedge \ \forall \mathtt{n} \in \mathtt{skey}[\mathtt{spci}[\mathtt{p}]]\colon \ \mathtt{susd}[\mathtt{p},\mathtt{n}] = \mathtt{susd}[\mathtt{q},\mathtt{n}])\}\end{aligned}$$

which can be proved easily.

Since the theorem is already proven, these checks are sufficient to infer that the SC *Corruptions* is implemented by the subsystem `Sys` of the intermediate specification consisting of APMs and AMCs. In proofs of liveness processes (e.g., the SC *LiveIn*),

```
FAPM SBufferKey
Handling of sequence numbers of data units in the transmitter entity
PROCESS SBufferKey
          ( pdu, pci, usd : Any ;  Set of PDUs, PCIs, and user data
            encpdu : SUBSET (pdu) ;  Set of PDUs accepted by FC function
            spci : SET[pdu → pci] ;  Reference to the PCI within a PDU
            skey : SET[pci → ((SUBSET (fkey) ∪ {"≪≫"})] ;
            Reference to sequence numbers within a PCI
            mb : Nat;    Max. no. of data units transfered within a PDU
            ir : fkey;   number of available sequence numbers
            ms : fkey )  maximum number of active sequence numbers
IMPORT Constants;  Import data types (fi. fkey) and functions / operators (fi. +, inc)
BODY
  VARIABLES
    cRq : fkey;  Sequence number to be assigned to the next data unit sent
    sbk : SUBSET (fkey);  Buffer of data unit sequence numbers to be transmitted
  INIT ≙ cRq = 0 ∧ sbk = ∅;
  ACTIONS
    Rq (krq : fkey) ≙  Data transfer request
        1.) Removing old sequence number and
            adding sequence number of the newly submitted data unit
      sbk′ = (sbk \ {(cRq - ms) mod ir}) ∪ {krq} ∧
        2.) The current value of cRq is exported to other processes by
            means of the parameter krq. cRq is incremented
      krq = cRq ∧ cRq′ = inc(cRq);
    MRq ( krq : ((SUBSET(fkey)) ∪ {"≪≫"}); p : pdu ) ≙
      Request data transfer to the receiver via the basic entity
        1.) Only PDUs p can be sent which are accepted by the frame check function
      p ∈ encpdu ∧
        2.) The PDU p contains only data, the sequence numbers of which are in sbk.
            At most mb-many data segments are transmitted within a pdu.
      ( krq = "≪≫" ∨ ( krq ⊆ sbk ∧ card(krq) ≤ mb ) ) ∧
        3.) The selected sequence numbers (krq) are stored in the field
            identified by skey[spci[p] ] of the pdu p
      krq = skey[spci[p] ] ∧
        4.) The variables cRq and sbk are not changed
      cRq′ = cRq ∧ sbk′ = sbk;
END
```

Figure 5. The FAPM *SBufferKey* (for the sake of clarity we omitted mechanisms to segment data units).

a fourth check has to be performed to guarantee that the intermediate specification does not consist of APMs or AMCs spoiling the liveness of the subsystem Sys. This check profits from the fact that protocol and service specifications are composed from framework processes. For all processes of the framework we already analyzed and

```
┌──────────────── Theorem to prove SC Corruptions ────────────────┐
 Definitions:
   LET Pars ≙ Parameter condition
                  The basic service guarantees :
         mtc ⊆ { (p, q) |
                  Either delivered PDU q is not accepted by the FC function
                 q ∉ encpdu ∨
                   Or 1.) The segment numbers are not corrupted
                 ( skey[spci[p] ] = skey[spci[q] ] ∧
                       2.) The user data of the segments is corrupted at most within
                           the limit of tc
                   ∀ l ∈ usd ∃ m ∈ usd ∀ n ∈ pkey :
                     ( (l, m) ∈ tc ∧ n ∈ skey[spci[p] ] ∧
                       ( susd[p, n] = usdsplit[skn[n], l] ⇒
                         susd[q, n] = usdsplit[skn[n], m] ) ) ) };
       Sys ≙ Subsystem which implements the SC Corruptions
       APMs:
          SBufferKey (pdu, pci, usd, encpdu, spci, skey, skk, skn,
                      skm, usdsize, mb) ∧
          SBufferUsd (pdu, usd, usd, skk, skn, usdsize, usdsplit) ∧
          RBufferKey (pdu, pci, usd, encpdu, spci, skey, skk, skn,
                      skm, usdsize, rc, tg) ∧
          RBufferUsd (pdu, usd, usd, skk, skn, usdsize, usdsplit) ∧
       AMCs:
          MSDUId ∧
          MCorruptions (pdu, mtc) ∧
          MPhantoms (pdu,encpdu) ∧
       Coupling constraint:
          CC_Corruptions;
├──────────────────────────────────────────────────────────────────┤
 Theorem:
   IN Pars ∧ Sys ⇒ Corruptions(usd, tc)
└──────────────────────────────────────────────────────────────────┘
```

Figure 6. A theorem to prove the SC *Corruptions*.

documented, which processes might block others and therefore spoil the liveness of Sys. Thus, the check is reduced to the comparison of the process types of the protocol specification with the list of the incompatible process types of the liveness theorem. The theorem manager [3] performs this task automatically.

In this way the refinements of all SCs of the four channels were verified. Since all SCs of the service specification are implemented by the intermediate specification, we can complete the proof. We simply have to examine that the coupling constraints of the service specification and the intermediate specification are consistent. This is guaranteed by the structure of the framework processes. The second proof step, verifying that the XTP protocol specifications refine the intermediate specifications, was performed analogously.

8. Summary of the XTP specification and proof

By developing specifications of the XTP service and protocol and performing the refinement proof, we checked the functional correctness of the XTP unicast functions. All in all, we can state that the protocol mechanisms of XTP provide the service specified by the service specification (section 5). Nevertheless, we detected some ambiguities in the informal description that might lead to defective implementations. In particular, the functionality of the mechanisms to report lost data in the selective repeat retransmission mode is described incompletely: the case of reporting the loss of the last data unit of a data stream is not mentioned explicitly. Thus, in our opinion, the XTP description allows two different solutions. If two cooperating implementations realize different solutions, loss of data can occur. According to the XTP developers, only one of the two solutions is valid and the next version of the XTP definition will state this clearly.

Besides this safety problem, we detected two weaknesses with regard to liveness. The first point depends on the XTP priority mechanism which enables the prioritization of certain connections. If the PDU selection algorithm is too rigid, a connection of high priority can suppress a connection with lower priority. Second, a station can paralyse its peer by abusing a protocol function which is included to enable it to discover the actual turn-around-time. If it requests immediate reports of its peer's connection state continuously, the peer is not allowed to deliver any received user data to its user. These problems are not errors of XTP but may occur in implementations.

9. Concluding remarks

The transfer protocol framework was established in order to support the practical design of high-speed transfer protocols. In contrast to most other protocol verification approaches, our work did not concentrate on automated tools and general verification methods, but aimed at a direct support of the understanding of services, protocols, their logical constituents and structuring by means of the provision of process types and theorems. Since the framework follows a new approach – and since it explicitly aims to support practical protocol designs – experiences of its application to complex real-life protocols are of interest. Therefore we applied the framework to the protocol XTP and accomplished a complete analysis of the functionality of the unicast services of XTP. In our opinion, the work was performed very efficiently since we were able to complete the verification within a time period of three weeks. We were able to construct all formal specifications needed by composing instantiations of framework modules. All necessary proofs of the protocol verification were directly supported by the theorems of the framework. Therefore, we profited from the comprehensive work, we spent for the development of the framework and the proof of its theorems. Moreover, we took advantage of the logical orientation of the framework which provides for a close relationship between formal verification steps and protocol properties. This facilitated the interpretation of results and supported a clear structuring of the work.

References

[1] T. Braun, B. Stiller and M. Zitterbart, XTP and VMTP on multiprocessor architectures, in: *Proceedings of the International Workshop on Advanced Communications and Applications for High-Speed Networks* (1992).

[2] W.A. Doeringer, D. Dykeman, M. Kaiserswerth, W. Meister, H. Rudin and R. Williamson, A survey of light-weight transport protocols for high-speed networks, IEEE Transactions on Communications 11 (1990) 2025–2039.

[3] O. Drögehorn, Ein Werkzeug zum formal basierten Entwurf von Hochleistungsprotokollen, Diploma thesis, University of Dortmund, 1996 (in German).

[4] P. Herrmann, Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen, Ph.D. dissertation, Universität Dortmund (November 1997). To appear in Deutscher Universitätsverlag 1998 (in German).

[5] P. Herrmann and H. Krumm, Compositional specification and verification of high-speed transfer protocols, in: *Protocol Specification, Testing, and Verification XIV*, eds. S.T. Vuong and S.T. Chanson (Chapman & Hall, 1994).

[6] P. Herrmann and H. Krumm, Re-usable verification elements for high-speed transfer protocol configurations, in: *Protocol Specification, Testing, and Verification XV*, eds. P. Dembiński and M. Średniawa (Chapman & Hall, 1995).

[7] ISO, LOTOS: Language for the temporal ordering specification of observational behaviour, International Standard ISO/IS 8807 (1989).

[8] L. Lamport, The temporal logic of actions, ACM Transactions on Programming Languages and Systems 3 (1994) 872–923.

[9] T.F. La Porta, A feature rich transport protocol: Functionality and performance, Ph.D. dissertation, Columbia University, New York (May 1992).

[10] T.F. La Porta and M. Schwartz, Architectures, features, and implementation of high-speed transport protocols, IEEE Network Magazine (1991) 14–22.

[11] T.F. La Porta and M. Schwartz, The MultiStream Protocol: A highly flexible high-speed transport protocol, IEEE Journal on Selected Areas in Communications 11(4) (May 1993).

[12] N. Lynch and M. Tuttle, An introduction to input/output automata, CWI Quarterly 2(3) (September 1989) 219–246.

[13] A. Mester and H. Krumm, Composition and refinement mapping based construction of distributed applications, in: *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (BRICS, Denmark, 1995).

[14] W.W. McCune, OTTER 3.0 Reference Manual and Guide, Research Report ANL-94/6, Argonne National Laboratory, Argonne, IL (January 1994).

[15] Protocol Engines, Incorporated, XTP protocol definition revision 3.4 (1989).

[16] A.S. Tanenbaum, *Computer Networks*, 3rd edn. (Prentice-Hall, Englewood Cliffs, NJ, 1996).

[17] C.A. Vissers, G. Scollo and M. van Sinderen, Architecture and specification style in formal descriptions of distributed systems, in: *Protocol Specification, Testing and Verification VIII*, eds. S. Agarwal and K. Sabnani (Elsevier, 1988).

[18] A.C. Weaver, The Xpress transfer protocol, Computer Communications 1 (1994) 46–52.

[19] XTP transport protocol specification, revision 4.0, XTP Forum, Santa Barbara, CA (1995).