Telecommunication Systems 20:3,4, 291–310, 2002

Framework and Tool Support for Formal Verification of Highspeed Transfer Protocol Designs

PETER HERRMANN and HEIKO KRUMM {peter.herrmann;krumm}@cs.uni-dortmund.de
University of Dortmund, Department of Computer Science, Dortmund, Germany

OLAF DRÖGEHORN olaf@uet.e-technik.uni-kassel.de
University of Kassel, Department of Electrical Engineering, Kassel, Germany

WALTER GEISSELHARDT gd@uni-duisburg.de
Gerhard-Mercator-University Duisburg, Department of Electrical Engineering, Duisburg, Germany

Abstract. Formal description techniques, verification methods, and their tool-based automated application meanwhile provide valuable support for the formal analysis of communication protocol designs. Nevertheless the practical analysis of modern protocols still requires relatively great efforts and therefore many protocol developments do not employ formal methods. In that context the transfer protocol framework aims to complementary support. It supplies a rich collection of specification modules and guides their efficient composition to service and protocol specifications. Moreover the functional relations between service properties and implementing protocol mechanisms have been investigated systematically. The framework provides a collection of corresponding theorems to be applied to protocol correctness proofs. In result protocol verification can be reduced to the selection, instantiation, and proper arrangement of framework theorems. The verification process can further be supported by special tool-assistance. The tool COAST identifies the compositional structure of a protocol specification mechanically and selects according framework theorems. It splits service property proofs into arrangements of subproofs where the subproofs can mainly be accomplished by application of the selected framework theorems. After outlining the general transfer protocol framework approach we concentrate on the introduction of the tool COAST. We describe its functions and clarify its application by means of the verification of the complex real-life high-speed data transfer protocol XTP.

Keywords: composition, framework, protocol synthesis, TLA, automated verification, XTP

Introduction

Due to the high performance demands of modern highspeed and multimedia applications many new data transfer protocols and protocol variants were recently developed. Since most of these protocols are very complex, one should support their design by formal methods (cf. [Gibbs, 17]). In reality, however, protocols are frequently developed without any formal support, although standardized formal description techniques (ISO/OSI: ESTELLE [13] and LOTOS [41], ITU: SDL [49]) are available. Therefore protocols are often designed by means of incomplete and ambiguous protocol descriptions. Furthermore, many protocol developers omit the development of abstract service specifications, which describe the communication services to be provided by the protocols. If, however, a service specification is lacking, there is no way to check the correctness of the protocol with respect to the services to be provided. Thus, design errors are often detected in late development phases and lead to costly delays.

While many protocol developers acknowledge the benefits of formal modeling and analysis in principle, the methods are seldom put into real practice since they are difficult to apply and mostly require considerable efforts of well-educated experts. Therefore most verification approaches are meanwhile accompanied by tool-assistance in order to facilitate the application and reduce its costs. Mainly two major approaches to tool-supported formal protocol verification were developed, model-checking and theorem-proving.

Model-checking relies on a finite model of the protocol (cf. [Clarke et al., 6]). Mostly a finite state transition system is used which models the composition of underlying services and all protocol entities. The service to be provided is either also defined by a finite state machine or by a temporal logic formula which describes the properties of the service. Due to the finiteness of the models, the reachable states and transitions of the protocol model can be explored exhaustively and the model-checker tool can in principle fully automatically decide if the protocol is a correct model of the service or not. In practice, however, one major problem exists, the so-called state explosion. Since protocol models can have very large sets of reachable states, verifications often fail because of their enormous time and space requirements.

Model-checking based systems include CADP [Garavel et al., 15], Concurrency Workbench [Cleaveland et al., 7], COSPAN [Kurshan, 35], HyTech [Alur et al., 2], MurΦ [Dill et al., 8], SMV [McMillan, 44], SPIN [Holzmann, 30], TLC [Yu et al., 53], and UPPAAL [Larsen et al., 39]. Many of these tools apply special algorithms supporting the efficient representation of large state sets (e.g., supertracing [Holzmann, 29], "on-the-fly" model-checking [Fernandez and Mounier, 14], and reachability graph projections [Ip and Dill, 32; Krumm, 33]). Thus, relatively complex communication protocols can be verified by model checking nowadays. Nevertheless, high-speed transfer protocols like XTP [52] and MSP [La Porta and Schwartz, 38] support a wide spectrum of quality of service demands and therefore comprise rich sets of interacting protocol mechanisms. Resulting protocol models suffer from state space explosion and model-checking can only be applied to simplified models.

The second major protocol verification approach is based on symbolic logical deduction and applies mechanical theorem provers. Here, both the protocol and the provided service are specified by logical formulas (e.g., predicate logic, higher order logic, temporal logic). Based on the axioms and inference rules of the logic a theorem-prover tool tries to find a proof that the protocol formula implies the service formula. Theorem-prover based approaches include EHDM [Rushby et al., 48], HOL [Chou, 5], Isabelle [Paulson, 47], Larch [Guttag and Horning, 19], Nqthm [Boyer et al., 3], OTTER [McCune, 43], PVS [Owre et al., 46], and TLP [Engberg et al., 12].

Due to the application of symbolic reasoning the state space explosion problem does not exist directly, even protocols and services with infinite state spaces can be handled. Indirectly, however, it is present, too, since automated theorem provers have to manage search spaces in order to find proving deduction chains. The size of the search spaces and the time of searches again exceeds practical limits if abstract service properties shall directly be inferred from detailed protocol descriptions. Since many logics are undecidable, even in principle there is no guarantee that proof searches will terminate in finite time. Therefore theorem provers usually require intensive human guiding. The user has to split and structure the verification by explicit introduction of lemmas to be proven in the course of separate proof sessions. Moreover in many cases even a lemma proof can be accomplished only if the user guides the proof interactively and controls the activation of suitable axiom sets and deduction strategies. Additionally the efforts for the special design of suitable lemmas have to be considered. Very often lemmas are needed which express subtle invariant properties. They have to be developed in a longer process of alternating proof trial and lemma strengthening steps. In summary theorem-prover based protocol verifications tend to be time-consuming and expensive (fi., a theorem-prover based verification of the relatively simple Bounded Retransmission Protocol, an Alternating Bit Protocol extension which tackles data losses by time-out triggered retransmissions and provider aborts, needed an effort of three man-months [Havelund and Shankar, 20]).

During formal verification processes one moreover experiences a further problem which leads to considerable additional efforts. Usually the verification not only exposes errors of the logical protocol design stemming from wrong design decisions. Quite often the verification also uncovers several specification bugs which result from wrong formal modeling of abstractly correct ideas. Thus verification processes are stamped by time-consuming iterations of proof trials, bug detection, and correction.

We were aware that most existing approaches for the formal design of communication systems mainly concentrated on two topics, on tool-support and on general fundamentals of suitable specification and verification techniques, and considered a third topic complementarily, the support of users by means of application-domain specific theories [Herrmann and Krumm, 26,27]. The theories can provide conceptual frameworks and can supply predefined specification and verification elements. They can prepare and facilitate the formal modeling and reasoning tasks and can directly support the systematic understanding of application problems and methods.

For the domain of communication protocol design we developed the so-called transfer protocol framework [Herrmann and Krumm, 24,26]. It defines a basic architecture for formal compositional specifications and supplies corresponding generic reusable specification modules. Thus it supports the efficient development of bug-free protocol and service specifications. Moreover the framework supplies theorems. Each theorem describes an implication between a certain protocol mechanism subsystem and a certain service property. The theorem states that the protocol mechanisms correctly implement the service property. These theorems act as re-usable verification elements. The proof that a certain protocol as a whole correctly implements all required properties of the service to be provided, can be structured into subproofs where each subproof corresponds to a theorem instance. Thus the theorems substitute subproofs and reduce verification efforts substantially. Furthermore the design of the structuring of the main proof is also supported by the framework since that structuring corresponds well with the compositional structure of the protocol specification.

In consequence, the transfer protocol framework reduces the task of protocol verification to a series of subtasks:

– identification of the compositional structure of the specifications,
– design of a corresponding structuring of the proof into subproofs,
– design of a theorem instance for each subproof.

The design of a theorem instance consists of the selection of a generic theorem from the set of theorems of the framework followed by the choice of a suitable theorem parameter setting. The parameter setting of a theorem has to correspond with the parameter settings of the specification modules of the compositional protocol specification. The theorems are accompanied by conditions which constrain the parameter settings accordingly. Therefore, in fact, still logical proofs are necessary. Those parameter condition proofs, however, are relatively simple and can usually be accomplished without problems.

In order to enhance the verification support of the transfer protocol framework further, we developed the special tool COAST [Drögehorn, 10]. COAST is supplied with the compositional specification of the protocol and with the specification of the service to be provided. It relies on a database holding all framework theorems and their descriptions. COAST analyzes the compositional structure of the protocol specification and automatically selects the theorems which are needed for the protocol verification. It deduces the parameter settings of the theorems from the parameter settings of the protocol specification components. Moreover it prepares the parameter condition proofs. Finally, COAST translates the condition formulas into the syntax of a frontend tool [Geist, 16] for the theorem prover OTTER [McCune, 43]. Because of that preparation the theorem prover front-end tool as well as the theorem prover can be engaged without any additional interaction. OTTER usually accomplishes the proofs without interactive guidance due to the simplicity of the parameter conditions.

In the sequel we introduce the verification tool COAST in more detail. We describe its structure and functionality. Moreover we clarify its application by outlining the COAST-based verification of the high-speed transfer protocol XTP [52]. Additionally we give a concise overview over the context of COAST. Our conception of framework-based specification and verification and the transfer protocol framework employ the formal specification technique cTLA [Herrmann and Krumm, 23; Mester and Krumm, 45]. cTLA is based on Lamport's TLA (Temporal Logic of Actions) [Lamport, 36] and supports the modular description of process systems. Similar to LOTOS [41] and to DisCo [Kurki-Suonio, 34], systems are composed from processes which interact via synchronous joint actions. The process composition of cTLA, however, is of particular interest since it has the character of superposition which passes safety and liveness properties of subsystems over to embedding systems.

1. Framework-based formal modeling and cTLA

A formal modeling framework supplies re-usable elements and provides architecture guidelines in order to support the efficient development of specifications and correctness proofs.

The supply of re-usable building blocks for specifications depends on a suitable conception for specification modules. On the one hand a module shall represent typical elements of protocols and services in a way that supports their direct integration into specifications. On the other hand the elements shall be really re-usable. Therefore they shall be employable in a wide range of applications. Under these requirements we follow the argumentation of Vissers et al. [50] pleading for the structuring of protocol and service specifications into processes where two sorts of processes are of interest. So-called resource-oriented processes represent active components of a system as they are parts of implementations or implementation-near logical structures. So-called constraint-oriented processes correspond to more abstract views and represent logical conditions for behaviors occurring within the system. Constraint processes can refer to certain parts of a system and to special points of view and thus can be devoted to separated aspects of a system. Furthermore, a notion of process types supports re-usability. Specification modules do not define processes directly. Instead they define process types which can depend on generic parameters and can be employed for the instantiation of a wide range of process instances. The design of the specification language cTLA considers those requirements. In particular, cTLA supports the free combination of resource-oriented and constraint-oriented structures and therefore can be applied on different levels of abstraction.

The notion of theorems is the well-known general conception for re-usable proof building blocks. A theorem is a proven formula and can be used as axiom in the proof of other formulas. Protocol verifications, however, usually need quite special theorems reflecting particular combinations of protocol mechanisms and mechanism parameter settings. Therefore the practical re-usability is restricted and further means are of interest. With respect to special mechanism parameter settings, theorems can also have parameters enhancing their range of application. With respect to particular combinations of mechanisms, means are of interest which support the separation of aspects in a way that protocol correctness proofs can be split into a series of separated proofs. Then the theorems can also be devoted to single aspects each and the theorems can have a wider range of application.

The solution to that structuring problem of verifications and theorems is based on the special character of process composition in cTLA. Here, process specifications correspond to temporal logic formulas and the specification of a system which is composed from processes corresponds to the logical conjunction of the process formulas. Syntactical restrictions provide for the consistency of the conjunction. Therefore the specification of a system implies all specifications of the constituting processes. All relevant properties of processes are also properties of a system as a whole. Thus process composition has the character of superposition (cf. [Chandy and Misra, 4;

Kurki-Suonio, 34]) and verifications of system properties can be based on separated consideration of subsystems.

In cTLA, the superposition-based verification of systems is called structured verification. Assume *Sys* to be a system which is composed from a series of processes $Proc_1, Proc_2, \ldots, Proc_n$ and *Spec* to be a system property which can be described by a conjunction of subformulas $Sf_1 \wedge Sf_2 \wedge \cdots \wedge Sf_m$. Furthermore, assume that *Sys* has a suitable structuring, i.e., for each subformula Sf_i of *Spec* there is a subsystem Sub_i of *Sys* which consists of a subset of the processes $Proc_{i_1}, Proc_{i_2}, \ldots, Proc_{i_n}$ in a way that the implication $Sub_i \Rightarrow Sf_i$ is true. Then the verification that the system *Sys* has the properties described by *Spec* is accomplished by the proof of the formula $Sys \Rightarrow Spec$ which can be deduced from the series of implications $Sub_i \Rightarrow Sf_i$. The different subimplications $Sub_i \Rightarrow Sf_i$ correspond directly with different aspects of the system. During framework design the interesting aspects can be identified and each theorem can concentrate on a subimplication.

The properties which can be expressed by cTLA process and system descriptions are safety and liveness properties as they are introduced for state transition systems in [Alpern and Schneider, 1]. In order to support convenient fine-grained aspect structures we propose to separate safety and liveness strictly and moreover to split safety as well as liveness considerations into a series of more basic aspects (e.g., the transfer protocol framework proposes separated protocol safety aspects like packet corruptions, packet loss, and packet reordering).

2. cTLA

In cTLA, specification modules describe process types where each process is modeled by a state transition system. As an example we refer to the definition of the process *C* in figure 1. This process specifies the service constraint that, except for phantoms, all delivered data units are transmitted between the users of a service without corruptions.

The syntax of cTLA is oriented at programming languages like Modula 2. The process header consists of the keyword `PROCESS`, the process name *C*, and optionally the list of generic parameters. In our example the generic parameter `usd` models the set of data units which can be transfered between two service users. The keyword `IMPORT` refers to the inclusion of other modules (i.e., *Symbols*) which contain definitions of data types, functions, and constants. The state space of a process is modeled by variables which are declared in the section `VARIABLES`. In our example process *C* `buf`, describing a set of pairs of a sequence number[1] and an user data unit, is the only variable. In `buf` all data units are stored which were ever sent by the transmitting service user. A predicate headed by the construct `INIT` models the set of initial states. Thus, in the initial state of *C* the variable `buf` corresponds to the empty set. The state transitions are modeled by actions which are defined in the section `ACTIONS`. An action models a set of transitions. It is a predicate about a pair of a current and a next

[1] The data type *key*, which specifies the set of available sequence numbers, is declared in process *Symbols*.

```
PROCESS C ( usd : Any ) ! usd : set of transferred user data
IMPORT Symbols;
BODY
  VARIABLES
    buf : SUBSET(key × usd);
    ! Buffer of all data units ever sent
  INIT ≜ buf = ∅;
  ACTIONS
    Rq (krq : key; d : usd) ≜
    ! Transmission of user data d with sequence no. krq
      buf′ = buf ∪ {(krq,d)} ;
    In (krq : key; d : usd) ≜
    ! Delivery of user data d with sequence number krq
      ( krq = "notsent" ∨ ∀ e ∈ usd :: ((krq,e) ∉ buf) ∨
        (krq,d) ∈ buf ) ∧
      buf′ = buf ;
END
```

Figure 1. Safety process *C*.

state. The current state is referenced by variables (fi. `buf`). The next state is referenced by so-called primed variables (fi. `buf'`). A pair of a current and a next state, the variables of which fulfill the predicate, is a state transition of the action. Action definitions can contain data parameters. In our example the action `Rq` corresponds to the submission of data units. For instance, `Rq(2,"data")` describes the submission of the data unit `"data"` and the assignment of the sequence number `2` to this data unit. The variable `buf` in the next state contains the pairs of `buf` in the actual state and additionally the pair `(2,"data")`. The action `In` models the delivery of a data unit `d` with the sequence number `krq`. A data unit may be delivered only if it is either a phantom message (`krq = "notsent" ∨ ∀ e ∈ usd :: ((krq,e) ∉ buf)`) or if the delivered data is not corrupted during the transmission (`(krq,d) ∈ buf`). In addition to the actions defined in the section `ACTIONS`, a cTLA process may also perform a so-called stuttering step, the execution of which does not change the process state.

With respect to the separation of safety and liveness properties (cf. [Alpern and Schneider, 1]), the process type *C* models only safety properties. Thus, *C* tolerates state sequences, where after a finite number of state changes only stuttering steps are performed (i.e., the process is suddenly terminated). To rule out terminating state sequences, process actions can be attributed with fairness assumptions. After the action definition section of a process, one adds description constructs of the form `WF: In;` or `SF: In;`. The WF construct expresses that an action (i.e., `In`) has to be executed weak fairly. A weak fair action must be performed eventually if it would otherwise be enabled continuously for an infinite period of time. By the SF construct an action is declared to

```
PROCESS XTPService (XTPCap : Nat) ! XTPCap : capacity of
                                  !          the service
  PROCESSES
    ...;
    C   : Corruptions (Byte,{ (k,k) | k ∈ Byte })
                            ! No Corruptions of data are allowed
    Cap : Capacity (XTPCap) ! Buffersize in number of SDUs
    Id  : SDUId             ! Assignment of unambiguous sequence
                            ! numbers
    G   : Gaps (0);         ! No Gaps in transfered data stream
    LIn : LiveIn (...);     ! Data units are delivered lively
    ...;
  ACTIONS
    Rq (krq : key; d : Byte) ≜
    ! Transmission of user data d with seq. no. krq
      Id.Rq (krq) ∧ C.Rq (krq, d) ∧ Cap.Rq (krq) ∧
      G.stutter ∧ LIn.stutter ∧ ...;
    fIn (krq : key; d : Byte) ≜ ...;
    nIn (krq : key; d : Byte) ≜ ...;
END
```

Figure 2. Service specification *XTPService*.

be strong fair. It has even to be performed, if it is disabled from time to time. Weak fair and strong fair actions were introduced in [Alpern and Schneider, 1]. In contrast to direct liveness properties, these fairness assumptions guarantee not to be contradictory to the safety properties of a process. Therefore, in TLA [Lamport, 36] and cTLA liveness properties are generally modeled by weak and strong fair actions. Moreover, the processes of the transfer protocol framework either model safety properties or liveness properties only.

In cTLA, processes are combined to systems similar to Lotos [41]. The processes interact via synchronous joint actions. The transfer of data between processes is modeled by action parameters. The variables of a process are private and therefore cannot be accessed by other processes. Like a process, the system is modeled by a state transition system as well. The vector of the process variables forms the system states. The system transitions are described by system actions. In a system action a subset of the processes executes simultaneously a joint action while the other processes perform a stuttering step. In [Herrmann, 21] we proved that a cTLA system specification corresponds to a single process model and in consequence to a canonical TLA formula.

The process *XTPService* in figure 2 is a typical example of a cTLA system composed from processes. The processes combined to the system are declared in the section `PROCESSES`. For instance, the process *C* (cf. figure 1) is an instance of the process type *Corruptions* with the parameter setting `(Byte,{ (k,k) | k ∈ Byte })`. In the section `ACTIONS` of the system description the system actions are declared by conjunctions of process actions and stuttering steps. In the example *XTPService*, the local process actions `Rq` of the processes *Id*, C, and *Cap* are coupled to the system action `Rq`, while the processes *R*, *G*, and *LIn* participate in `Rq` by stuttering steps.[2]

As emphasized above the process composition of cTLA has the character of superposition (cf. [Chandy and Misra, 4; Kurki-Suonio, 34]) which guarantees that a property fulfilled by a process or a subsystem is also a property of each system which contains this process or subsystem. In cTLA, superposition is based on the consistent logical conjunction of processes. The consistency of process compositions with respect to safety properties follows directly from the syntax of cTLA processes since the state space of processes is defined by process-private variables only. Therefore safety properties of different processes cannot interfere with each other.

With regard to liveness, superposition is more subtle due to the joint action coupling. Actions of different processes can participate in the same system action. If one process action of a system action is not enabled, the other process actions are blocked and liveness properties of those processes may be violated causing inconsistencies. cTLA prevents that by employing weaker fairness assumptions than those defined in [Alpern and Schneider, 1]. The WF and SF constructs of cTLA do only refer to so-called conditional fairness assumptions which require process progress only with respect to state sequences where a process action as well as the containing system action are enabled. Therefore blocking joint action peers cannot violate the fairness assumptions of cTLA processes.

That restriction of fairness to conditional fairness, however, makes it harder to express absolute liveness properties which are of particular interest for the system design. For that purpose conditional fairness statements have to be joined with assumptions that fair actions are not too often blocked by the environment of a process. Those assumptions are called environment conditions and have to be proven in the course of liveness theorem applications.

3. Transfer protocol framework

The transfer protocol framework consists of specification modules and of theorems. The specification modules are modeled by cTLA process type definitions. They describe service constraints, protocol mechanisms, and constraints of a basic transfer medium which is used by the protocol mechanisms. The specification modules are structured into three layers: the Service Contraints (SCs) model single properties of a communication service. As pointed out in the upper part of figure 3, one can develop service specifications by composing SC instances. Protocol specifications are described according to the well known scenario shown in the lower part of figure 3. Like SCs, the Abstract Medium Constraints (AMCs) model properties of the basic transfer medium. The protocol instances are composed from protocol mechanisms. To simplify the protocol verification, the framework provides two different groups of specification modules to describe protocol mechanisms. The Abstract Protocol Mechanisms (APMs) specify abstractions of real protocol mechanisms. They model only the essential functions of the protocol

[2] Stuttering steps are described by the pseudo-action name *stutter*.

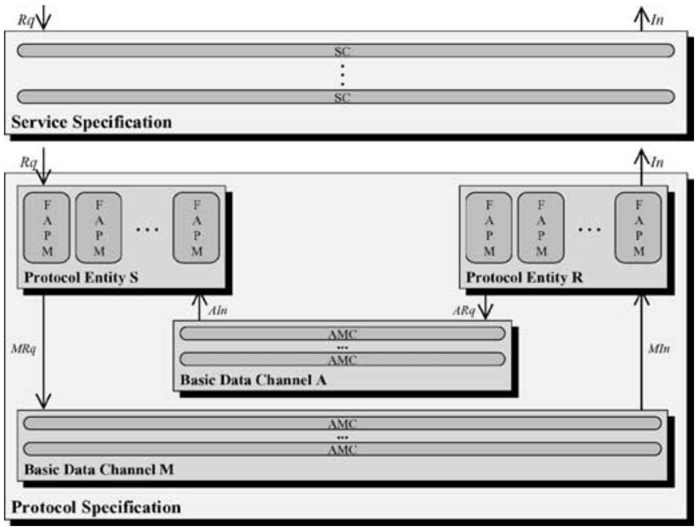

Figure 3. Structures of a service and a protocol specification.

mechanisms, but do not attach importance to details of an efficient implementation. In contrast, the Finite Abstract Protocol Mechanisms (FAPMs) model real protocol mechanisms in a quite direct manner. A real data transfer protocol is specified by composing appropriate FAPMs and AMCs. By the composition of APMs and AMCs, one creates a more abstract protocol which is used as an intermediate model in order to decompose the verification into two phases.

The framework theorems correspond to logical implications between cTLA systems. Due to the structuring of the specification modules into the three layers service, abstract protocol, and protocol, the theorems are divided into two groups. The SC theorems contribute to proofs that an abstract protocol fulfills a service. They guarantee that abstract protocol subsystems, modeled by APMs and AMCs, implement single SCs. The APM theorems are used to prove that a protocol fulfills an abstract protocol. They state implications between a protocol subsystem combined from FAPMs and AMCs and an APM.

Figure 4 refers to an SC theorem stating that an abstract protocol subsystem *Sys* implies the SC *LiveIn*. Among other APMs and AMCs, *Sys* contains the APMs *SLiveMRq* and *RLiveMRq*. This implication is only valid if the parameter condition *Pars* is true. It ensures that the actual parameters of the process instances of *Sys* and of *LiveIn* are consistent with each other. Furthermore, the whole abstract protocol has to fulfill the invariant condition *EnvCond*, which states that the fair actions in *Sys* are not blocked too often by processes of the abstract protocol specification. Thus, it is guaranteed that the conditional fairness assumptions of the actions in *Sys* fulfill the liveness property to be modeled by *LiveIn*. *EnvCond* has to be checked only if the SC to be fulfilled models a liveness property. To enable a mechanized check of *EnvCond* by COAST, all processes of the framework, which do not violate the environment condition of a theorem, are listed in the section `CORRESPONDS WITH`.

```
THEOREM LiveIn
  LET Pars ≜ mla = {(p,q)| skey[spci[p] ] = skey[spci[q] ] ∧
                           stack[spci[p] ] = stack[spci[q] ] ∧
                           stcre[spci[p] ] = stcre[spci[q] ] ∧
                           p ∈ encpdu  ∧  q ∈ encpdu} ∧ ...;
      Sys ≜ SLiveMRq (pdu, pci, usd, encpdu, spci, skey, sack,
                      snak, scre, stack, stcre, skk, skn, skm,
                      usdsize, usdsplit, mcr, rcc) ∧
            RLiveARq (pdu, pci, usd, encpdu, spci, skey, sack,
                      snak, scre, stack, stcre, skk, skn, skm,
                      usdsize, rcu, rcc, ma, mr, mo) ∧
            ... ∧ $CC_{LiveIn}$;
      EnvCond ≜
           ∀ krq,p,kd : Enabled(SLiveMRq.fMRq(krq,p,kd)) ⇒
             (krq,p,kd) ∈ $Sys.e_{fMRq}$ ∧
           ∀ p,kd : Enabled(RLiveARq.fARq(p,kd)) ⇒
             (p,kd) ∈ $Sys.e_{fARq}$ ∧ ...;
  IN Pars ∧ Sys ∧ [ ] EnvCond ⇒ LiveIn (usd, 0, tg, c, {});
  CORRESPONDS WITH
    Process ≜ SBufferKey (pdu, pci, usd, ...);
    Process ≜ SBufferUsd (pdu, pci, usd, skk, ...);
    ...;
END
```

Figure 4. SC theorem to prove the SC *LiveIn*.

At this moment, the framework consists of 133 specification modules (28 SCs, 44 APMs, 14 AMCs, and 47 FAPMs) and 165 theorems (31 SC theorems and 134 APM theorems).

4. COAST

The selection and arrangement of suitable framework theorems to perform protocol verifications is supported by the tool COAST (Consistency of a specification in cTLA^{+}) [Drögehorn, 10]. The input files of COAST are specifications of a more detailed system (e.g., a protocol specification) and of a more abstract system (e.g., a service specification) which both are modeled by cTLA process compositions (cf. figure 2). Furthermore, COAST has access to a database containing theorems in the syntax described in figure 4. By selecting theorems from the database and checking that the specifications are composed and parametrized according to the conditions of the theorem, COAST verifies that the detailed specification implies the abstract specification. This is equivalent to proving that the detailed system correctly implements the abstract system in the

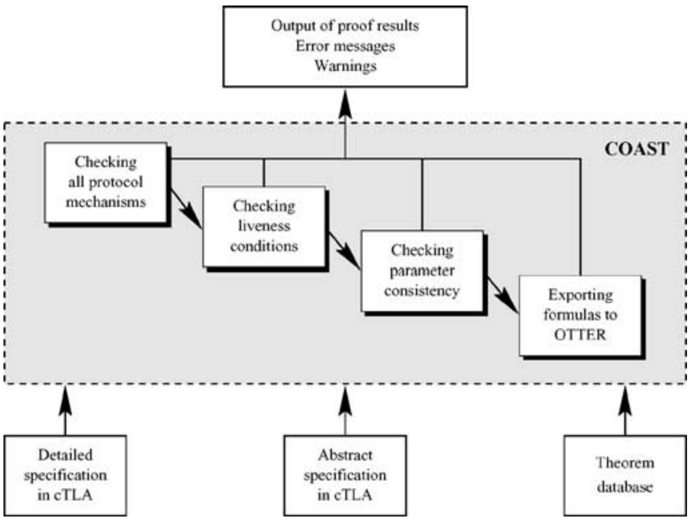
Output of proof results
Error messages
Warnings
COAST
Checking all protocol mechanisms
Checking liveness conditions
Checking parameter consistency
Exporting formulas to OTTER
Detailed specification in cTLA
Abstract specification in cTLA
Theorem database

Figure 5. Elements of the tool COAST.

sense, that the detailed system contains all of the mandatory properties of the abstract system.

Due to the subdivision of the framework theorems into three layers the tool has to be applied twice to perform a complete protocol proof. In one phase COAST checks that an abstract protocol fulfills a service. The protocol developer provides the abstract protocol specification as the detailed system, the service specification as the abstract system, and the database of all SC theorems to COAST. In the other phase COAST proves that a protocol fulfills the abstract protocol. In this case the protocol forms the detailed system and the abstract protocol acts as the abstract system. COAST uses the database of the APM theorems.

COAST performs a protocol proof in four sequent steps (cf. figure 5). If a check cannot be completed successfully, the tool terminates and reports an error message.

In the first check suitable theorems are selected from the database. As explained in section 3, a theorem guarantees that the subsystem *Sys* of the detailed system implies a process of the abstract system. COAST identifies a theorem for each process of the abstract system. The theorem has to contain the process on the right side of the implication. Thereafter, the tool checks if the processes of the subsystem *Sys* in the theorem are contained in the detailed system. If COAST cannot find a suitable theorem for each process of the coarse-grained specification, the proof will be aborted.

If the process on the right side of a theorem models a liveness property, COAST checks in a second step whether the detailed system contains processes which might spoil the liveness of the subsystem *Sys* and therefore violate the invariant condition *EnvCond* of the theorem. This task is simple since all processes of the framework, which are compatible to *Sys*, are listed in the section `CORRESPONDS WITH` of the theorem (cf. figure 4). Thus, COAST checks if the detailed system contains processes which are not contained in this list. If not all processes are compatible, the theorem is rejected and COAST jumps back to the first step selecting a new theorem from the database.

The last two steps deal with the consistency of the actual parameters of the processes. The framework assumes that these actual parameters of different process instances are represented by syntactically equal terms, which substitute equally named formal process type parameters. The tool checks this condition in the third step. If it detects syntactically different parameter settings, it adds corresponding proof obligations to the fourth step.

Finally, the fourth step is devoted to the proof of the condition *Pars* of the theorem. *Pars* ensures the logical consistency of the parameter settings and is represented by a first-order logic formula. COAST translates *Pars* (and possibly the additional obligations of the third step) into the input syntax of a frontend tool [Geist, 16] for the automated theorem prover OTTER [McCune, 43]. If OTTER verifies *Pars* and the additional obligations, the protocol verification is completed successfully. If OTTER fails to perform a proof, the user of COAST has to decide if the formula is false or if the proof was too difficult or too extensive to be proved without interactive support. Then one can structure the proof and supply additional lemmas to OTTER. Since, however, *Pars* is very simple in the most theorems, OTTER can usually prove these formulas without any further support. As an alternative to OTTER, one can also use the prover MACE [McCune, 42] which searches small finite models of the formula.

5. Example

For clarification we will outline the proof of the Xpress Transfer Protocol XTP [51,52]. To support the data transfer requirements of different distributed applications, XTP consists of a broad spectrum of protocol functions (cf. [Doeringer et al., 9; La Porta and Schwartz, 37]) which are mostly asynchronous to each other. Thus, various combinations of protocol functions are possible. The Transfer Protocol Framework supports this modularity directly. In [Herrmann and Krumm, 25] we introduced the formal specification and verification of XTP. There, the proofs were carried out manually by application of suitable framework theorems. Altogether we needed only three weeks for the specification and verification. However, by application of COAST we could reduce the verification time further.

As outlined in [Herrmann and Krumm, 25], we design a service specification, an abstract protocol specification, and a protocol specification by parametrizing and composing framework processes, first. The service specification *XTPService* is listed in figure 2. SCs of the framework (e.g., *SDUId*, *Corruptions*, *Gaps*, *LiveIn*) are instantiated and composed according to the desired properties of the service. Since, for example, the service does not tolerate gaps in the stream of delivered data, the specification contains the process instance *G* of the SC *Gaps*. The process parameter `tg` of *Gaps*, which describes the maximum number of gaps in the stream of delivered data, is set to 0. Thus, gaps are not tolerated at all. The protocol specification *XTPProtocol* is composed from FAPMs, which model the protocol mechanisms of XTP, and AMCs describing the

```
PROCESS XTPProtocolAbs
  PROCESSES
! APMs : Protocol mechanisms with infinite variables
    SBK : SBufferKey ! Protocol mechanism modeling the handling
                     ! of sequence numbers in the transmitter
                     ! entity
               (XTPpdu, ! format of the XTP pdu (modeled by a
                        ! cTLA record)
                XTPpci, ! format of the XTP protocol control
                        ! information
                Byte,   ! XTP provides bytewise data transfer
               ...);
    SBU : SBufferUsd ! Protocol mechanism modeling the store of
                     ! data in the transmitter entity
               (XTPpdu, ! format of the XTP pdu (modeled by a
                        ! cTLA record)
                Byte,   ! XTP provides bytewise data transfer
               ...);
    ...;
  ACTIONS
    Rq (krq : key; d : usd) ≜
    ! Transmission of user data d with sequence number krq to
    ! the service provider
      SBK.Rq (krq,d) ∧ RBK.stutter ∧ SBU.Rq (krq,d) ∧
      RBU.stutter ∧ RG.stutter ∧ RR.stutter ∧ RD.stutter ∧
      RP.stutter ∧ ...;
    ...;
END
```

Figure 6. Parametrized abstract protocol specification *XTPProtocolAbs*.

constraints of the basic service. In figure 6 the abstract protocol specification *XTPProtocolAbs* is sketched. It models the protocol mechanisms of XTP in a more abstract way than *XTPProtocol* and is composed from APMs (e.g., *SLiveMRq*) and AMCs.

Below, we sketch the proof that *XTPProtocolAbs* fulfills *XTPService*. COAST is provided with these specifications and the database of the SC theorems. For instance, the theorems listed in figures 4 and 7 are contained in this database.

COAST performs the four proof steps outlined in section 4. First, it selects the first process of the service specification. In the example, this is the SC *Corruptions* guaranteeing that corrupted data will not be delivered to the service user. In order to prove this SC, COAST selects the theorem listed in figure 7 from the database. In the first step COAST checks that all processes of the subsystem *Sys* are also contained in the specification *XTPProtocolAbs*. Since that is true, the tool completes this step successfully (output `TESTING Mechanisms` in figure 8).

```
THEOREM Corruptions

  LET Pars ≜  mtc ⊆
              {(p,q) | q ∉ encpdu  ∨
                       ( skey[spci[p] ] = skey[spci[q] ] ∧
                         ∀ n ∈ skey[spci[p] ] :
                           (susd[p,n],susd[q,n]) ∈ tc) ) } ∧
            ∀ l,m ∈ usd ∀ n ∈ {0, ..., usdsize[l] - 1}:
              (usdsplit[l,n],usdsplit[m,n]) ∈ tc ⇒ (l,m) ∈ tc;
      Sys  ≜SBufferKey (pdu, pci, usd, encpdu, spci, skey,
                        skk, skn, skm, usdsize, mb) ∧
            SBufferUsd (pdu, usd, susd, skk, skn, usdsize,
                        usdsplit) ∧
            RBufferKey (pdu, pci, usd, encpdu, spci, skey,
                        skk, skn, skm, usdsize, rcu, rcc) ∧
            RBufferUsd (pdu, usd, susd, skk, skn, usdsize,
                        usdsplit) ∧
            MSDUId ∧
            MCorruptions (pdu, mtc) ∧
            MPhantoms (pdu,encpdu) ∧
            CC_Corruptions;
  IN Pars  ∧  Sys ⇒ Corruptions(usd, tc)

  CORRESPONDS WITH
    ...;

END
```

Figure 7. SC theorem to prove the SC corruptions.

```
   * START THEOREM-CHECK !!!! *
   Checking Service_Element: Corruptions
   - Trying the 1. Theorem for: Corruptions
     TESTING Mechanisms:
     → OK
     TESTING Correspondings:
     → NOT NECESSARY
     TESTING Parameters:
     → OK
   → One or more theorems has been tested correctly for this
      Service_Element !!!!!
   ...
   * END of CHECK !!!! *
```

Figure 8. Output message of COAST.

Since *Corruptions* does not describe a liveness property, the liveness of *Sys* cannot be spoiled by its environment. Thus, COAST omits the second proof step.

In the third step COAST checks, that formal parameters in the theorem containing the same name are replaced by identical defined variables or identical values. For example, the parameters `pdu` of the processes *SBufferKey* and *SBufferUsd* in *Sys* are both replaced by the value `XTPpdu` (cf. figure 6). COAST finished this check successfully, too (output `TESTING Parameters` in figure 8).

In the forth step the condition *Pars* of the theorem *Corruptions* is checked. First, the formal parameters of *Pars* are replaced according to the instantiations of the processes in the specifications *XTPService* and *XTPProcotolAbs*. Thereafter COAST translates the formula into the syntax of the OTTER frontend. Figure 9 depicts the translation of the first conjunct of *Pars* in the theorem. In the part `FORMULAS` the name *form_Corruptions2_1* is assigned to the first conjunct of *Pars*. The proof script guiding OTTER is defined in the part `THEOREM`. OTTER shall prove the formula *form_Corruptions2_1* by contradiction assuming the already proven formulas *mtc_pred*, etc.

Likewise, COAST performs the checks for each process of *XTPService* and creates OTTER proof scripts. Finally, OTTER verifies the formulas *Pars* of all selected theorems by application of the proof scripts. Since for each process of *XTPService* at least one theorem was identified which passes the checks by COAST and since the OTTER proofs succeeded, the verification that *XTPProtocolAbs* fulfills *XTPService* is successful.

COAST selected 33 theorems of the database of SC theorems which passed the first proof step. Since the abstract protocol specification of XTP is compatible to these theorems with respect to liveness properties, they passed the second step as well. The theorems passed also the third step, since the actual parameters were always replaced by syntactically equal terms. OTTER could prove the *Pars* condition of 25 theorems directly. Eight theorems had to be enhanced with additional data-definitions in order to be proven by OTTER. By this support, OTTER could prove another four theorems. The

```
MODULE Corruptions2_equal
FORMULAS
     form_Corruptions2_1 ≜
       mtc ⊆ {(p,q) | q ∉ encpdu ∨
                      ( skey[spci[p] ] = skey[spci[q] ] :
                        (susd[p,n],susd[q,n]) ∈ tc) };

THEOREM Test_form_Corruptions2_1
<1>{1}ASSUME mtc_pred, q_pred, p_pred, encpdu_pred, skey_pred,
             spci_pred, susd_pred, n_pred, tc_pred
PROVE form_Corruptions2_1
QED BY CONTRADICTION;

END Corruptions2_equal
```

Figure 9. Part of the generated OTTER formula for the SC-theorem *Corruptions*.

remaining four OTTER theorem proofs had to be supplied by some simple intermediate lemmata. These lemmata, however, could be designed easily.

For the proof that the protocol system fulfills the abstract protocol system, COAST selected 52 theorems of the 134 theorems of the database of APM theorems. Due to the similarity of the protocol specification *XTPProtocol* and the abstract protocol specification *XTPProtocolAbs*, COAST and OTTER performed these proofs without further support by the user. Altogether, the complete proof that the protocol specification *XTP-Protocol* fulfills *XTPService* could be performed within three hours.

6. Conclusion

With the help of the XTP example we outlined the concept of the transfer protocol framework and, in particular, the verification tool COAST. Besides XTP and some simpler sliding window protocols, we applied the framework to specify and prove the complex high-speed protocol MSP [La Porta and Schwartz, 38], too, which could be examined also with a remarkable few expense of work (cf. [Hinz, 28]). The framework can be accessed via WWW (`http://ls4-www.informatik.uni-dortmund.de/RVS/P-TPM`).

Further work meanwhile expanded the specification technique cTLA in order to rise its acceptance in industrial protocol development. On the one hand, cTLA was extended to specify real-time properties and continuous behaviour [Herrmann et al., 22]. This facilitates the formal specification and verification of distributed real-time systems. In particular, cTLA and its extensions are already applied to control systems for hybrid chemical systems [Hermann and Krumm, 27]. The extensions, however, can be utilized for the modeling of quality aspects in communication protocols as well (fi. modeling the transmission of multimedia data [Graw et al., 18]).

The approach, moreover, was used to the translation of cTLA system specifications into the hardware description language VHDL (cf. [Lipsett et al., 40]). By means of design compilers (fi. Synopsis) VHDL hardware module descriptions can be transformed automatically into hardware circuits. In this field we are using cTLA as a system-level description language with the ability of proving the specification against the service specification and other constraints. The objective of the work is the mechanical transformation of already proven descriptions into hardware circuits in order to reduce the needs for expensive functional circuit tests and simulations [Drögehorn et al., 11; Hümmer and Geisselhardt, 31]. This work was partly funded by the postgraduate research programme CINEMA of the German research foundation DFG.

References

[1] B. Alpern and F.B. Schneider, Defining liveness, Information Processing Letters 21 (1985) 181–185.

[2] R. Alur, T.A. Henzinger and P.-H. Ho, Automatic symbolic verification of embedded systems, in: *Proc. of the 14th Annual IEEE Real-time Systems Symposium*, 1993, pp. 2–11.

[3] R.S. Boyer, M. Kaufmann and J.S. Moore, The Boyer–Moore theorem prover and its interactive enhancement, Computers and Mathematics with Applications 29(2) (1995) 27–62.

[4] K.M. Chandy and J. Misra, *Parallel Program Design – A Foundation* (Addison-Wesley, Reading, MA, 1988).

[5] C.-T. Chou, Mechanical verification of distributed algorithms in higher-order logic, The Computer Journal 38(2) (1995) 152–161.

[6] E.M. Clarke, O. Grumberg and D.E. Long, Model checking and abstraction, ACM Transactions of Programming Languages and Systems 16(5) (1994) 1512–1542.

[7] R. Cleaveland, J. Parrow and B. Steffen, The concurrency workbench: A semantics-based tool for the verification of concurrent systems, ACM Transactions of Programming Languages and Systems 15(1) (1993) 36–72.

[8] D.L. Dill, A.J. Drexler, A.J. Hu and C.H. Yang, Protocol verification as a hardware design aid, in: *1992 IEEE Internat. Conf. on Computer Design: VLSI in Computers and Processors* (IEEE Computer Soc. Press, Los Alamitos, CA, 1992) pp. 522–525.

[9] 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 38(11) (1990) 2025–2039.

[10] O. Drögehorn, Ein Werkzeug zum formal basierten Entwurf von Hochleistungstransportprotokollen (in German), Diploma thesis, University of Dortmund, Department of Computer Science, Dortmund, Germany (1996).

[11] O. Drögehorn, O. Terhorst, H.-D. Hümmer and W. Geisselhardt, Formal specification and verification of transfer-protocols for system-design in VHDL, in: *Proc. of the 1st Internat. Forum on Design Languages (FDL)*, Lausanne, Switzerland, 1998.

[12] U. Engberg, P. Grønning and L. Lamport, Mechanical verification of concurrent systems with TLA, in: *Verification 4th Internat. Workshop, CAV '92*, eds. G. von Bochmann and D.K. Probst, Montreal (Springer, Berlin, 1992) pp. 44–55.

[13] ESTELLE: A formal description technique based on an extended state transition model, ISO, International Standard ISO/IEC 9074 ed. (1997).

[14] J.-C. Fernandez and L. Mounier, "On the fly" verification of behavioural equivalences and preorders, in: *Lecture Notes in Computer Science* (Springer, Berlin/Heidelberg, 1991) pp. 181–191.

[15] H. Garavel, M. Jorgensen, R. Mateescu, C. Recheur, M. Sighireanu and B. Vivien, CADP'97 – Status, applications and perspectives, in: *COST247 Workshop on Applied Formal Methods in System Design*, 1997.

[16] A. Geist, Eine Theorembeweiser-gestützte Entwicklungsumgebung für die halbautomatische Verifikation verteilter Systeme (in German), Diploma thesis, University of Dortmund, Department of Computer Science, Dortmund, Germany (1994).

[17] W.W. Gibbs, Software's chronic crisis, Scientific American 271(3) (1994) 72–81.

[18] G. Graw, P. Herrmann and H. Krumm, Verification of UML-based real-time system designs by means of cTLA, in: *Proc. of the 3rd IEEE Internat. Symposium on Object-oriented Real-time distributed Computing (ISORC2K)*, Newport Beach (IEEE Computer Soc. Press, Los Alamitos, CA, 2000) pp. 86–95.

[19] J.V. Guttag and J.J. Horning, eds., *Larch: Languages and Tools for Formal Specification*, Springer Texts and Monographs in Computer Science (Springer, 1993).

[20] K. Havelund and N. Shankar, Experiments in theorem proving and model checking for protocol verification, in: *Formal Methods Europe (FME '96)*, Oxford (Springer, Berlin, 1996) pp. 662–681.

[21] P. Herrmann, *Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen* (in German) (Deutscher Universitätsverlag, 1998).

[22] P. Herrmann, G. Graw and H. Krumm, Compositional specification and structured verification of hybrid systems in cTLA, in: *Proc. of the 1st IEEE Internat. Symposium on Object-oriented Real-time Distributed Computing (ISORC'98)*, Kyoto (IEEE Computer Soc. Press, Los Alamitos, CA, 1998) pp. 335–340.

[23] 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, Vancouver, BC, Canada, 1994 (Chapman & Hall, London) pp. 339–346.

[24] 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, Warsaw, Poland, 1995 (Chapman & Hall, London) pp. 171–186.

[25] P. Herrmann and H. Krumm, Modular specification and verification of XTP, Telecommunication Systems 9(2) (1998) 207–221.

[26] P. Herrmann and H. Krumm, A framework for modeling transfer protocols, Computer Networks 34(2) (2000) 317–337.

[27] P. Herrmann and H. Krumm, A framework for the hazard analysis of chemical plants, in: *Proc. of the 11th IEEE Internat. Symposium on Computer-Aided Control System Design (CACSD 2000)*, Anchorage (Omnipress, 2000) pp. 35–41.

[28] V. Hinz, Formale Spezifikation und Verifikation eines Hochleistungstransferprotokolls mit dem Transferprotokollframework (in German), Diploma thesis, University of Dortmund, Department of Computer Science, Dortmund, Germany (1998).

[29] G.J. Holzmann, Algorithms for automated protocol verification, AT&T Technical Journal (1990) 32–44.

[30] G.J. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering 23(5) (1997) 279–295.

[31] H.D. Hümmer and W. Geisselhardt, New aspects in high-level specification, verification, and design of IT protocols, in: *14th Internat. Symposium on Integrated Circuits and System Design (SBCCI 2001)*, Pirenopolis, Brazil, 2001, pp. 58–63.

[32] C.N. Ip and D.L. Dill, Better verification through symmetry, Formal Methods in System Design 9(1/2) (1996) 41–75.

[33] H. Krumm, Projections of the reachability graph and environment models, in: *Automatic Verification Methods for Finite State Systems*, ed. J. Sifakis (Springer, Berlin, 1990).

[34] R. Kurki-Suonio, Hybrid models with fairness and distributed clocks, in: *Lecture Notes in Computer Science*, Vol. 736 (Springer, New York, 1993) pp. 103–120.

[35] R.P. Kurshan, *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach* (Princeton Univ. Press, Princeton, 1994).

[36] L. Lamport, The temporal logic of actions, ACM Transactions on Programming Languages and Systems 16(3) (1994) 872–923.

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

[38] 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) (1993).

[39] K. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, Springer International Journal of Software Tools for Technology Transfer 1(1/2) (1997) 134–152.

[40] R. Lipsett, E. Marschner and M. Shahdad, VHDL – The language, IEEE Design & Test (1986) 28–41.

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

[42] W.W. McCune, A Davis–Putnam program and its application to finite first-order model search: Quasigroup existence problems, Technical Report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA (1994).

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

[44] K.L. McMillan, A methodology for hardware verification using compositional model checking, Technical Report, Cadence Berkley Labs, Berkley, CA, USA (1999).

[45] A. Mester and H. Krumm, Composition and refinement mapping-based construction of distributed applications, in: *Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Aarhus, Denmark, 1995.

[46] S. Owre, S. Rajan, J.M. Rushby, N. Shankar and M.K. Srivas, PVS: Combining specification, proof checking, and model checking, in: *Computer-Aided Verification (CAV'96)*, eds. R. Alur and T.A. Henzinger, New Brunswick (Springer, Berlin, 1996) pp. 411–414.

[47] L.C. Paulson, Isabelle: The next 700 theorem provers, in: *Logic and Computer Science*, ed. P. Odifreddi (Academic Press, New York,1990) pp. 361–386.

[48] J. Rushby, F. von Henke and S. Owre, An introduction to formal specification and verification using EHDM, Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, 1991.

[49] SDL, SG X: Recommendation Z.100: CCITT specification and description language SDL, ITU (1993).

[50] 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*, Vol. VIII, eds. S. Agarwal and K. Sabnani (Elsevier, Amsterdam, 1988) pp. 189–204.

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

[52] XTP transport protocol specification, revision 4.0, XTP Forum, Santa Barbara, CA, USA (1995), available via FTP: `dancer.ca.sandia.gov in directory pub/xtp4.0`.

[53] Y. Yu, P. Manolios and L. Lamport, Model checking TLA+ specifications, in: *Correct Hardware Design and Verification Methods (CHARME '99)*, eds. L. Pierre and T. Kropf (Springer, Berlin, 1999) pp. 54–66.