FORMAL SPECIFICATION AND VERIFICATION OF A NETWORK INDEPENDENT ATOMIC MULTICAST PROTOCOL[*][†]

M. Baptista[‡] S. Graf[§] J-L. Richier[‡] L. Rodrigues[†]
C. Rodriguez[¶] P. Veríssimo[†] J. Voiron[‡]

Research on formal description techniques during the last years has revealed new trends on the description of distributed systems. Nevertheless, the application of these techniques to real and complex systems is not straightforward and there are not many case studies in this area. This paper presents an experience in protocol building area, by involving a close interaction between protocol design and formal verification, and shows off its application in the design of a real distributed system: a network independent atomic multicast protocol.

1 Introduction

During the last years, emphasis has been put in formal specification. The ISO committees are making efforts in the development of formal description techniques, such as Lotos and Estelle, while CCITT is the responsible for the definition of SDL. These techniques lead to new fields in the description of distributed systems. Their main purpose is the offering of unambiguous, concise and implementation independent descriptions of a system. They mean also to be a platform for a rigorous analysis and validation of the described system during its life cycle.

Several formal descriptions of well-known protocols have been carried out. A useful guide has been written by CCITT in order to clarify and encourage the use of formal description techniques [CCI88]. However, for the moment, there is a lack of tools for those techniques, which constrains their use, especially in the development of new complex protocols. And, as stated in [Rud88], this might be a reason why it is so difficult to convince implementors of the usefulness of these techniques.

The authors, through their experience in using either semi-formal [D4 89] or formal [RRSV87a] techniques, recognize the potentialities of formal description in protocol design. This paper presents an experience concerning the design of a new multicast

[*]This work has been supported in part by CEC, through ESPRIT project 2252, Delta4.

[†]A version of this paper was published in FORMAL DESCRIPTION TECHNIQUES, III, Ed. J. Quemada, and J. Mañas, and E Vazques, © 1991 North-Holland

[‡]IST - INESC, Rua Alves Redol, 9 - 6^o, 1000 Lisboa, Portugal

[§]LGI - IMAG, BP 53X, 38041 Grenoble Cedex, France

[¶]VERILOG S.A., Le Forum du Pré Millet, 38330 Montbonnot, France, and formerly LGI.

communication protocol. The aim of this work was to *develop and simultaneously formally validate* a protocol design by using a verification tool.

For this verification the complete design of the protocol has been taken into account. The errors to be detected concern any deviation from the expected service, and not only deadlocks and unspecified receptions, as is often the case. Futhermore, each error found during the verification can be fedback to the design. Thus, it is possible to eliminate as much design errors as possible before the actual implementation of the protocol. Our intention is not to validate the implementation itself.

This experiment has been carried out in the Delta4 ESPRIT project [D4 88]. This project seeks to define a distributed real-time computing architecture that is open, allowing the integration of heterogeneous computing elements, and dependable, providing a quality of service on which users can have justified reliance [PSB*88]. Delta4 systems have an OSI-like communication stack, where the lowest layers play an important role. One of these protocols is AMp (Atomic Multicast protocol) [VRB89] which has been developed in the project. Reasoning in terms of an OSI-like protocol stack, AMp is integrated in the Data Link Layer. It provides highly parallel reliable group communication primitives which are useful in a great number of applications, namely fault-tolerance techniques based on replicated computations.

The paper is organized as follows: the next section gives a presentation of the fundamental aspects of formal specification and verification. The following section provides an overview on AMp. Section 4 is devoted to the formal specification and verification of AMp, showing how the protocol has been modelled. And finally, we give some results and try to draw some conclusions about this experience.

2 Formal specification and verification of protocols

In order to formally verify the design of a protocol, a specification in some formal language with a rigorous and non-ambiguous semantics is needed. The non-ambiguous semantics should allow, besides the formal verification, the derivation of a reliable implementation, at least if the specification language is sufficiently close to the implementation language. Mainly two kinds of formal validation techniques have been proposed: deductive proof methods and model checking.

Deductive proof techniques, as proposed in [OG76,Hoa72,LS84], have the inconvenient of being of little help for the *analysis of errors*. This is due to the fact that the impossibility to construct a proof gives no information about the nature or the location of a possible error.

Model checking techniques are, like *simulation* techniques, based on the search of the graph of possible behaviours (called also *state graph*) of the system to be verified. But unlike simulation, model checking examines *all possible behaviours in a systematic way*. If the underlying state graph is finite - which can always be obtained for communication protocols - and not too large, the search is exhaustive and the result of the verification is reliable. As localization of errors was very important, we have chosen model checking techniques for the formal verification of AMp.

2.1 Xesar, a verification tool based on model checking

The verification tool Xesar [RRSV87b,GRRV89] implements model checking techniques: it evaluates properties given by formulas of the branching time temporal logic CTL [CES86] on a model obtained from a program written in Estelle / R [RRSV87b].

The system to be verified is described by a finite set of processes, forming a *closed system*: that is, for each possible communication, the complete description of both emission and reception must be provided. Therefore, the *protocol environment* must be explicitely described by a (set of) process(es). Each verification is done on a model obtained from such a program describing a particular system configuration with a particular initialisation, called in the sequel a *scenario*. Validation consists in the definition of a set of critical scenarios and their formal verification by using the Xesar tool.

The protocol description language Estelle / R is a variant of Estelle. The main difference is that in Estelle / R communication is modelled by *rendez-vous*. Communication by rendez-vous requires to represent finite buffers explicitly by processes. The problem of "communication through unbounded buffers" of Estelle is that deadlock situations can not be detected, as it is possible to just fill up infinitely some message buffer. The formal semantics of an Estelle / R model is based on the ATP algebra [NRSV90]: an explicit *clocktick* event is used for the translation of the "delay" construction of Estelle. This allows to evaluate time-bounds on the obtained model.

If the model can be stored in main memory (some hundred thousand states for a common workstation), the formulas are evaluated on this representation of the model.

Otherwise (the model has some millions of states), the service properties are expressed by automata and then evaluated on-the-fly, by traversing exhaustively the model without storing it completely in memory. An error is detected if during the traversal a state or a cycle not satisfying the property is encountered. In this case, the execution sequence leading to the error can be analysed. If the traversal can be completed without detecting any error, the model satisfies the property.

If a complete traversal is not possible in a reasonable time, the absence of detected errors does not allow to deduce that the protocol is correct; nevertheless, it increases the confidence we may have in it. In this case it is also possible to make several traversals with different criteria for the choice of the successor state in order to increase the coverage, as proposed in [Hol90].

2.2 Practical problems of formal verification

In practice, the verification of protocols raises different problems. A more complete discussion of this subject can be found in [GRV90]. Here we rather enumerate the problems:

- The first problem of model checking is *state explosion*: the state graph may increase exponentially with the number of processes in the system. A solution to this problem consists in the definition of a strategy by means of a set of "minimal scenarios". To deduce the correctness of the protocol in case of absence of errors, one must be able to affirm that each error of an arbitrary scenario is necessarily detected in one of the verified scenarios.

The set of minimal scenarios is defined by taking into account the structure of the algorithm, symmetry considerations, existence of distinct protocol phases, and so on. The definition of the scenarios needs a tight collaboration with the designer as a detailed knowledge of the design is needed. In fact, this task obliges the designer to explicitly structure the protocol. This has also the advantage that certain errors in the design are discovered during this phase of work.

- Another problem concerns the description of the environment. A complete description of the environment is not needed. But the environment process(es) must allow all behaviours satisfying the hypotheses on which the protocol under study is based. Therefore, the environment is in general extremely nondeterministic. If the protocol description combined with such an environment is verified to be correct, it can be deduced that the protocol works in any environment satisfying the hypotheses.

- A third problem is related to the verification of timing properties. The first solution is to model the expiration of a timeout by an arbitrary event, and then instead of verifying a timing property of the form “an action terminates in at most n time units” to verify a property of the form “the action *eventually* terminates” under some *fairness* assumption. Obviously, this approach does not allow to verify concrete timing aspects of the protocol.

  Particularily, for a protocol for real-time uses, it is important to have some results about time bounds. In our case it was possible to use the Estelle/R delay for the definition of timers and of the execution times of certain actions. This allows the verification of time bounds. However, the obtained results can only be guaranteed to be valid for the particular timer values and execution times of individual actions for which the verification has been carried out.

  In fact, it would be interesting to obtain execution time bounds as a function of parameters such as timer values and execution times of atomic actions, but to the knowledge of the authors no general methods yielding such results have been proposed.

3 AMp: a network independent atomic multicast protocol

AMp was engineered under the assumption that when a communication component fails, it fails *silent*: a failed component cannot disturb the communication system. AMp is appropriate for LANs: the current implementations are built on top of the exposed interface of VLSI LAN controllers; the resulting MAC[1] provides the user with multicasting primitives, while preserving the standard functionality.

However, from a formal viewpoint, AMp relies on a network service with a set of given properties, presented below. In that sense, it is network independent, since any network providing such a service is appropriate for supporting AMp.

[1]Medium Access Control sub-layer.

3.1 The Abstract Network concept

In the AMp design, an abstraction of the underlying medium was made. Several reasons justified this option. One of them was the intention of showing that the protocol worked correctly as a LAN independent protocol: there would be no need to establish verifications for every LAN port. The other reason relevant to this paper was the fact that, by abstracting from the physical network, and instead replacing it by a set of properties representing the functionality required by the AMp, we have indeed reduced the complexity of the system to verify. It may be seen that the properties are fulfilled almost directly by a large set of standard LANs; assuming that existing VLSI circuits correctly implement the relevant MAC protocols, this seemed to be a reasonable approach, with regard to reliance in the correctness of implementation of the abstract network properties.

The abstract network is defined as a set of nodes, each node having a source access point and a destination access point. The network takes a frame[2] transmission request at any source access point and delivers the frame, as an indication, at all the destination access points. A set of properties define the behaviour of the abstract network, namely: broadcast transmission, authentication, bounded omission degree, full duplex, network order and bounded transmission delay (a detailed description and justification of abstract network can be found in [VM90]).

A note about the *omission degree*: this property guarantees that, in a protocol phase composed of $k + 1$ series of N broadcasts, at least one fault-less series of N broadcasts is obtained (k is the allowed omission degree). The protocol is resilient to temporary omission failures, provided that, during each protocol phase, they are produced by one single component with an *omission degree* of value k and k is known and bounded during the life-time of the system.

3.2 The AMp service

The communication system nodes interconnected by the network will be called *stations*. Communication takes place inside groups of processes, each one in a different station, associated with a communication object called *gate*. All gates in a multicast gate group are identified by a unique, logical name. A user process may join and leave a group at any station through local gate opening or closing operations. Group membership is required to communicate.

The AMp service is defined by the following set of properties:

- *Unanimity*: any message[3] delivered to a correct participant[4] is delivered to all correct participants.
- *Non-triviality*: any message delivered was sent by a correct participant.
- *Accessibility*: any message delivered was delivered to a participant correct and accessible for that message.

[2]A frame is an AMp protocol data unit.

[3]A message is an AMp service data unit.

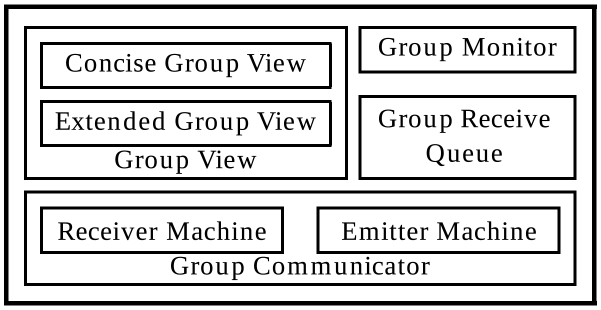
[4]A correct participant is a participant residing on a fault-free station.

- *Delivery*: any message is delivered, unless the sender fails or some participant(s) is(are) inaccessible[5].
- *Consistent causal order*: any two messages delivered to a participant are delivered to all correct participants, in the same order which obeys causality.
- *Termination*: messages are delivered within a known bounded time.
- *Consistent Group View*: given any receive ordering by the participants of the group, each change to group membership is indicated, in a consistent order, to all correct group participants.

These properties characterize what is meant by a *proper* service, and can thus be used as a basis for verifying the protocol design and its implementation. For the verification work, this set of properties is expressed by a set of formulas of temporal logic, which have been used to validate the protocol service.

3.3 The protocol

The complete description of AMp is not included here: for the interested readers, we suggest the reading of [VRB89]. The description below is a concise one, and we just describe the general ideas behind the protocol conception and the information relevant for the verification work.



Protocol entities
A station may belong to any number of gate groups. Each gate, the entity used by a participant to communicate, uses an instantiation of the AMp machine, comprising the Group Communicator, the Group View, the Group Monitor, and the Group Receive Queue (see figure above).

The Group View includes a Concise Group View, which is a reference for the current number of members, and an Extended Group View, a coded identification of all the group members. They are used for recovery from omissions and identification of the possible failed members. In a station, only one Emitter Machine may be active at a time; however, there are as many active Receive Machines as the currently pending atomic multicast transmissions in which the station takes part. The Group Communicator machines signal any detected failure to the Group Monitor. The Group Monitor assures

[5]The accessibility concept is presented and discussed in detail in [VRB89]; in short, this concept is related with recipient constraints such as lack of resources, which may cause it to sporadically refuse accepting a message. The recipient, however, signals its refusal to the sender, allowing the protocol to progress and terminate.

coherence of the Group View among all group members, executing the procedures of insertion and removal of group members and the recovery from error situations. The Group Receive Queue contains the pending messages, before they are delivered.

Protocol execution

Typically, an AMp user performs the following sequence of actions: first of all, it joins a multicast group; then, it issues send requests and receives confirmations (of its own transmissions) and indications (of group transmissions or of the new group composition); finally, it may leave the group. During these actions, any member of the group may fail, which implies the execution of a recovery procedure. Join, leave and recovery from failures are performed under the control of the monitor, which must previously be elected.

After this short introduction to the major steps of the protocol | message transmission, monitor election, joining, leaving and recovery | let us have a look at each one in more detail:

- Message transmission:

  In the *dissemination* phase, the sender sends a data frame. This phase ends after reception of all expected responses and may include the retransmission of the data frame, in case of missing responses. In worst case, it may take as many transmission rounds as the allowed omission degree plus one (See abstract network properties). Upon analysis of the received responses, the sender issues its decision (*decision* phase), which is normally *accept*. However, it can be *reject*, should some recipient be inaccessible. In any case, all recipients unanimously accept or reject the pending data frame.

  This is the normal way of performing message transmissions. To control recipients activity, a local timer is set in the sending station when a transmission starts. In order to control the sender activity another timer is set, locally to the receiving machine of each group member, after sending its response to the data frame while it waits for the decision that will terminate the protocol execution.

  Should it expire, the recipient sends a request for decision, which, like the data frame transmission, may also be retransmitted up to *omission degree* (K) times. A station is considered failed (silent) in two situations: either a recipient did not issue a response within $K + 1$ transmission rounds or the emitter did not issue a decision after $K + 1$ decision requests.

- Monitor election:

  In the protocol entities description, several critical activities were associated with a so called “Group Monitor”. The Group Monitor is normally inactive. So to speak, it only exists if locally activated by local invocations of the Group Monitor function. Since several group members may, concurrently, try to initiate similar actions, a monitor competition takes place. Only one competitor wins the monitor state; the others resign. The winner is called *active monitor*. If it fails on its turn, another monitor competition takes place in order to elect a new monitor; this procedure avoids deadlocks caused by monitor failure. The monitor acts in order to achieve group consensus on relevant events. The group traffic is suspended while the monitor is active. Upon completion of the action, it relinquishes the active state after traffic reestablishment.

- Joining and leaving a multicast group:

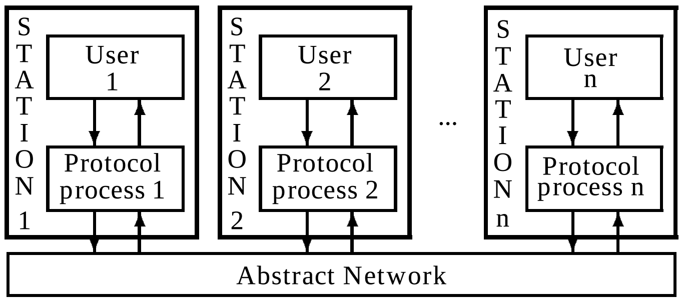
  In both operations, a new group view must be disseminated among all the group members, followed by a decision to effectively perform the group view update. These steps are similar to the message transmission actions. However, as they modify the group view, they are performed under monitor control. Notice that the joining step needs the extra action of acquiring the group view, by the participant desiring to enter a multicast group.

- Recovery:

  The failure of a station generates a short-lasting inconsistent situation, handled by the monitor in two phases: the first one (investigation) aims at identifying the failed stations and searching for the presence of pending messages; the second one (decision) includes the issuing of accept or reject for those messages, and the dissemination of a new group view.

4 Formal verification of AMp

The basis for the verification is the complete Estelle/R description of the AMp. Since a closed system is needed, a description of the environment must be provided, i.e. the network and the upper layers (Logical Link (LLC) and Network Management (NM)).



As already stated before, a set of scenarios is needed for the verification. Each of these scenarios describes a closed system containing the following processes (see the above figure for network configuration):

- A specific *protocol* process for each station depending on the scenario. It is obtained by assembling parts of the protocol description used in the particular scenario. In order to reduce the size of the generated state graph, parameters are tuned down (counters for retry, buffer sizes, ...), and only a single multicast group is considered as, according to the protocol and abstract network specifications, two different multicast groups cannot interfere. The messages are reduced to their identification, and the associated counters need not to be able to distinguish more than the maximum number of messages that can exist simultaneously.

- The *abstract network* is described by a single process delivering the service described in section 3.1. It models:
  - message transmission allowing the loss of messages up to a certain “omission degree”;
  - time: in the present modelling the progress of time is related to the transmission of messages through the abstract network. It is based on the following assumptions, which are similar to the ones made by the protocol designers:
    - transmission of frames through the network takes a bounded delay.
    - all other atomic actions take no time.

  As stated before, the abstract network should allow all behaviours compatible with the expected network service. In practice, to obtain a reasonable size for the model, some restrictions have to be made in the scenarios: for example, in some scenarios a timeout expires only if the corresponding message is really lost. In order to verify also situations of “early timeouts”, specific scenarios are considered where one station is “slower” or “faster” than the others.

- One *user* process for each station. It concerns the interactions with the LLC and NM, sending requests to the protocol process and receiving the confirmations and indications. The hypotheses concerning the user are extremely simple: it must always be able to accept confirmations and indications of any message; requests may be sent in any order consistent with the opened/closed status of the gate. The requests which are effectively sent depend on the scenario under consideration.

4.1 Definition of scenarios

The most difficult work during the verification of AMp concerns the definition of scenarios. This cannot be done automatically and every choice should be justified. Even if projective methods such as proposed in [GKM89] are applied, the problem is the same since all the morphisms used for the projections need to be found.

Scenarios were grouped in different sets, covering each of the relevant steps of AMp execution, described in section 3.3, including: message transmission, monitor election, joining a group, leaving a group, and failure recovery. A justification for the chosen set of scenarios can be found in [VM90]. The “sub-services”, i.e. the expected services for each of these steps were defined by sets of properties. For instance, for the monitor election scenarios the following properties were defined: a) There is one and only one winner of a monitor competition; b) If the current monitor fails and there are still alive members, another monitor competition takes place; c) The monitor competition always terminates successfully unless all the group members fail.

We also need some other informations to define the scenarios concerning each step:

- The set of the possible initial states in which the steps can start. For example, for the monitor election step the initial states correspond to the states reached after the occurrence of an error in any other step: failure of the sender or of one of the receivers during a message transmission step, failure of one of the stations during a joining step, and so on. For each class of initial states one or several scenarios are needed.

- The number of stations needed for the verification of each step. For example, for the monitor election step, it is possible to examine all possible error cases with 3 stations. The failed stations need not be represented. They just appear in the "Group Views" of the still correct stations.

5 Discussion

In the Delta4 project, the need to assure reliable communications among distributed sites has lead to the definition of the AMp protocol. As the general communication system is based on this low level protocol, the quality of its development is crucial. A practical solution to ensure this quality was to carry out simultaneously the design and its verification by using formal methods. Then, the implementation can be derived.

The original informal specification given as an implementation guide has not been adequate to achieve this aim. The first work has been to provide structured formal description of the protocol and of the delivered service, the second one to formally verify the consistency of the service definition with this protocol description. To achieve this work, the assumptions about the environment, i.e. the abstract network, had to be formalized.

These formalizations are useful on their own, as they help to detect early some inconsistencies in the formal specification. The verification has been reasonably carried out by using a set of scenarios whose definition has been derived easily from the protocol structure, and thus we have obtained good results. The generated models have a reasonable size (on average 350,000 states and 700,000 edges). The detected errors in the formal specification of the protocol were of two kinds:

- Local inconsistencies, easily corrected, such as bad initializations of local variables, wrong parameters, or too weak conditions for some transitions.

- More serious errors, requiring deeper changes, e.g. in some situations, the monitor election did not terminate.

The benefit of formal verification has been to increase the confidence in a more correct design of the Generic AMp protocol.

>From this significant case study, two major conclusions should be stressed:

- Concerning the formal validation:

– The formal definition of a suitable environment to carry out verification, derived from its formal requirements.

– For the modelling of time in case of many low-level protocols, a tractable solution is to relate the timing aspects with the messages passing through the network.

– The definition of scenarios depends on a good structuration of the protocol. As for abstract data types or structured programming, the verification task requires a good knowledge of the algorithm.

– The generalization of verification results to any number of connected stations is a difficult problem, for which we do not know any general solution.

- Concerning the implementation itself:

– Automatic translation can be hoped for, but is difficult, as a lot of constraints must be considered in any implementation, such as performance considerations.

– The automatic derivation, from the formal specification, of tests and assertions to be verified by the implementation can be envisaged.

Acknowledgments

A large part of this work has been developed in the scope of the Delta4 architecture. The authors wish to thank the project Consortium for recognizing the importance of formal specification and verification, and a number of people in the project team, who have contributed many useful suggestions.

References

[CCI88] CCITT. *Guidelines for the Application of ESTELLE, LOTOS and SDL*. 1988.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. *ACM TOPLAS*, 8(2):244–263, April 1986.

[D4 88] Delta-4 Consortium. *Overall System Specification*. BULL-SA, Grenoble, France, December 1988.

[D4 89] Delta-4 Consortium. *Delta-4 Implementation Guide, Release 1*. Multipoint Communication System, chap. 6.1-1: Generic Atomic Multicast Protocol specification, December 1989.

[GKM89] I. Gertner, R.P. Kurshan, and K. McMillan. Analysis of Digital Circuits through Symbolic Reduction. *IEEE Trans. CAD/ICS*, 1989.

[GRRV89] S. Graf, J-L. Richier, C. Rodriguez, and J. Voiron. What are the Limits of Model Checking Methods for the Verification of Real Life Protocols? In *Workshop on Automatic Verification Methods for Finite State Systems,* Grenoble, June 1989, LNCS Vol. 407, pp 275–285, Springer Verlag.

[GRV90] S. Graf, J-L. Richier, and J. Voiron. *Verification of systems with time-constraints*. Research Report Spectre C23, LGI-IMAG, Grenoble, 1990,

[Hoa72] C. A. R. Hoare. *Towards a theory of parallel programming*. Academic Press, 1972.

[Hol90] G. Holzmann. Algorithms for automated protocol validation. *ATT technical Journal*, 60(1):32–44, Jan / Feb 1990.

[LS84] L. Lamport and F. B. Schneider. The “Hoare logic” of CSP, and All That. *ACM Transactions on Programming Languages and Systems*, pp 281–296, April 1984.

[NRSV90] X. Nicollin, J-L. Richier, J. Sifakis, and J. Voiron. *ATP, an algebra for timed processes*. IFIP Conference on 'Programming Concepts and Methods', April 90, Israel

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, (6):319–340, 1976.

[PSB*88] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk. The Delta-4 approach to dependability in open distributed computing systems. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, pp 246–251, IEEE, Tokyo - Japan, June 1988.

[RRSV87a] J. L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. Verification in XESAR of the Sliding Window Protocol. In *Proc 7th International Symposium on Protocol Specification, Testing, and Verification*, pp 235–248, Zurich, Switzerland, May 1987.

[RRSV87b] J-L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. *XESAR: a Tool for Protocol Validation - User Manual.* Laboratoire de Génie Informatique, Grenoble, 1.2 edition, September 1987.

[Rud88] H. Rudin. Protocol Engineering: A Critical Assessment. In *Proc 8th International Symposium on Protocol Specification, Testing, and Verification*, pp 3–16, Atlantic City, USA, June 1988.

[VM90] P. Veríssimo and José A. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *9th Symposium on Reliable Distributed Systems*, IEEE, Huntsville, Alabama-USA, Oct. 1990.

[VRB89] P. Veríssimo, L. Rodrigues, and M. Baptista. AMp: a highly parallel atomic multicast protocol. In *SIGCOM'89 Symposium*, pp 83–93, ACM, Austin-USA, Sept. 1989.