COMPUTER COMMUNICATIONS SYSTEMS

edited by a. cerveira

COMPUTER COMMUNICATIONS SYSTEMS

Proceedings of the IFIP TC 6 First Iberian Conference on
Data Communications, IBERCOM '87
Lisbon, Portugal, 19–21 May, 1987

edited by

Alexandre CERVEIRA
University Nova de Lisboa
Monte de Caparica, Portugal

1988

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

ISBN: 0 444 70334 9

Published by:
ELSEVIER SCIENCE PUBLISHERS B.V.
P.O. Box 1991
1000 BZ Amsterdam
The Netherlands

Sole distributors for the U.S.A. and Canada:
ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
52 Vanderbilt Avenue
New York, N.Y. 10017
U.S.A.

LIBRARY OF CONGRESS
Library of Congress Cataloging-in-Publication Data

IFIP TC 6 Iberian Conference on Data Communications (1st : 1987 : Lisbon, Portugal)
Computer communications systems : proceedings of the IFIP TC 6 First Iberian Conference on Data Communications, IBERICOM '87, Lisbon, Portugal, 19-21 May 1987 / edited by Alexandre Cerveira.
p. cm.
Includes index.
ISBN 0-444-70334-9 (U.S.)
1. Data transmission systems--Congresses. I. Cerveira, Alexandre. II. International Federation for Information Processing. Technical Committee 6. III. Title.
TK5105.I325 1987
004.6--dc19 87-27277
CIP

PRINTED IN THE NETHERLANDS

COMPUTER COMMUNICATIONS SYSTEMS

IFIP TC 6 First Iberian Conference on
Data Communications, IBERCOM '87
Lisbon, Portugal, 19–21 May, 1987

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

PREFACE

This was the first IBERICOM conference. IBERICOM was intended as a periodical forum for technical and scientific knowledge exchange in Iberic Countries.

Portugal was proud to be the stage of this memorable conference on Data Communications which was made possible through the efforts of API (the Portuguese professional society "Assoçiacão Portuguesa de Informática"), together with IFIP TC6.

During the three days about thirty papers were presented in 10 technical sessions covering such subjects as Protocol Specification, Testing and Verification, Network Management, Performance and Inter-Netting. Some of the papers were of excellent quality and of great interest, as could be inferred from the reviewer's comments.

This conference gave an accurate sample of the problems which deserve world-wide attention of researchers and it provided the opportunity for these researchers to meet. In addition it lent itself as a perfect occasion for presenting Portuguese, as well as Spanish activities in Data Communication. Accordingly, the aim of this event, which was to promote the exchange of knowledge and experience between all the participants, has been fully achieved.

A great effort was made to organize such a conference and I am indebted and thankful to all the people involved. I also want to thank the Caloust Gulbenkian Foundation and the Instituto Nacional de Investigação Científica for their generous contributions which made IBERICOM '87 possible.

This book is a collection of most of the papers which were presented at the conference.

Alexandre Cerveira
Program Committee
Chairman

LIST OF COMMITTEES

PROGRAM COMMITTEE

Chairman: Alexandre Cerveira (P)
Vice Chairman: Luís Lavandera (E)

MEMBERS

Jorge Alves (P)
José Alves-Marques (P)
Bert Boutmy (NL)
Kiril Boyanov (BUL)
André Danthine (B)
Jan Ekberg (FIN)
Dipak Khakhar (S)
Koos Koen (RSA)
Fritz Kõnigshofer (USA)
H. Meier (DDR)
Louis Pouzin (F)
J. Puzman (CZ)
Peter Radford (GB)
Piercarlo Ravasio (I)
Juan Riera (E)
Harry Rudin (CH)
Wojciech Sobczak (PL)
Otto Spaniol (D)
Carl Sunshine (USA)
Tibor Szentivanyi (H)
Liane Tarouco (BR)
Ronald Uhlig (USA)

ORGANIZING COMMITTEE

Chairman: José Queiróz (P)

MEMBERS

Guilherme Arroz (P)
José Granado (P)
Cândido Manso (P)
José Tribolet (P)

TABLE OF CONTENTS

COMPUTER COMMUNICATIONS SYSTEMS
A.G. Cerveira (Editor)
Elsevier Science Publishers B.V. (North-Holland)

A REVIEW OF FORMALISMS FOR THE SPECIFICATION OF COMPUTER COMMUNICATION PROTOCOLS (invited paper) *

Vasco Freitas

Centro de Ciências e Engenharia de Sistemas
Universidade do Minho, 4719 Braga, Portugal

This paper reviews semiformal and formal specification methods currently being used for communication protocols. The review is informal and descriptive, draws from recent published work in the field and intends to address a significant number of tools. Verification techniques are also discussed.

1. INTRODUCTION

A specification of a product is a description of the way it is intended to behave, a predicate containing free variables, each of which standing for some observable aspect of the behaviour of the product [HOAR85]. It is a clear, complete and precise statement of *properties* of the product.

Specification methods may be broadly classified as *informal*, *semiformal* and *formal* [TURN84]. Although there is no clear definition of formality the common view is that a formal language consists of symbols plus rules for combining and manipulating symbolic expressions.

Fig.1 shows a classification given by Turner [TURN84] that categorises specification methods under a number of general headings.

Informal methods lack any form of mathematical basis. Natural language, though a convenient and natural way of expression, lack in precision and is often ambiguous. Diagrams are then often used to complement textual specifications, flowcharting being an example and exist in varied forms. Hierarchic decomposition techniques force specifications to be given at a high level first and then refined down into smaller parts. This technique merely make specifications more systematic.

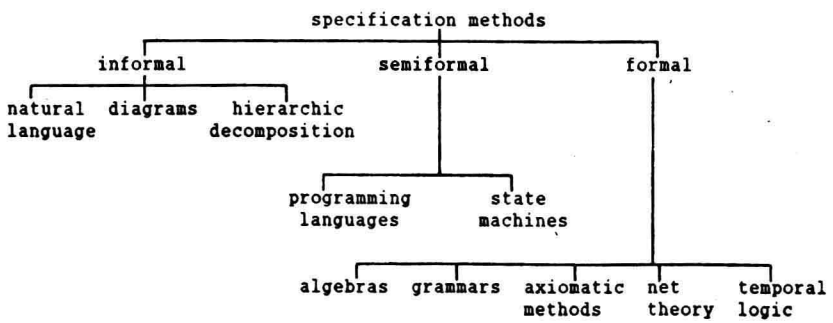

Fig. 1 Classification of specification methods [TURN84]

In semiformal methods, specifications are given some sort of mathematical rigour, the emphasis being less on proofs of properties and more on conveying

* Work sponsored by the Instituto Nacional de Investigação Cientifica, Portugal.

ideas in a readily understood form. Formal methods, which possess a richer mathematical structure have the great advantage of precision and that formal proofs may be conducted.

Computer software, is a critical component of computer systems and hence need to be accurately specified and validated before implementation can start. In particular, Computer Communication Protocols, a class of real-time software, pose specific specification and development problems due to their asynchronous and concurrent nature.

The OSI model of reference that has been standardized by ISO [ISO83] and CCITT [CCIT85], describes data communication systems as a series of layers, each providing services to the layer above. Each layer involves two types of specifications: *service specifications* and *protocol specifications*. The former is the abstract, implementation independent, definition of the functions provided by a particular layer (N) to the layer above (N+1), including the end-to-end significance of the services. The later define, at a lower level of abstraction, the peer-to-peer interaction of objects necessary to perform the layer N service, assuming services provided by the level N-1 below.

Validation is understood to mean all the actions necessary to show (or make shure) that a specification meets the communication requirements for which it is being designed.

Verification is an aspect of validation aimed at showing that certain protocol properties hold. Properties of interest are *safety* which ensure that only correct things will happen; *liveness*, something will eventually happen and *performance*, things will happen fast enough.

The purpose of this paper is to review and discuss the most commonly used formalisms for the specification of Data Communication Protocols. Examples are drawn from a transport layer service and the Alternating Bit Protocol (ABP) [BART69], where the terms *message* and *packet* refer, respectively, to the user defined abstract contents (data) and to the result of appending a sequence number (0/1) to a message. The ABP assumes that packets can be lost or duplicated in the transmission medium but not otherwise corrupted. A more formal and detailed comparison and evaluation of techniques can be found in [VENK86] and [COST86].

2. SEMIFORMAL SPECIFICATION METHODS

2.1. State Transition Models (Simple and Extended)

State Transition Models stem from the theory of finite state automata. This theory is closely related to the theory of grammars and networks. State Transition Models are based on the concept that a protocol can be represented as a Finite State Machine FSM = {S,I,O,T,P} where S, I and O are the sets of states, inputs and outputs and T and P are the state transition and output functions respectively ($T: I \times S \rightarrow S$, $P: S \times I \rightarrow O$). In Simple FSM models each state explicitly records all the information necessary to describe protocol behaviour, including all the acceptable events that may lead to a transition. Because large protocols have complex representations which become difficult to understand, Extended FSM models reduce the number of distinct states by grouping some of them together and associate context variables to states to distinguish between them.

A FSM may be represented graphically by drawing circles to represent distinct states and directed arcs connecting them, to represent transitions. Arcs are labeled with the inputs that lead to the transition [GOUD84; RUDI85]. Semantic actions expressed in some form of imperative language (like PASCAL or PL/I) are then often added to this graphic representation, to facilitate its translation to some other form of representation (algorithmic or implementation language).

Graphical models have the advantage of a great descriptive power, which is rapidly overcome if the number of states increase too much leading to diagrams difficult to follow. Furthermore, although verification of some properties can

be made over the graphical model, there is no method of doing graphically a thorough validation of the protocol description. State Transition Diagrams, have been extensively used to describe or specify protocols, eg, CSMA/CD [IEEE82], HDLC [BOCH80], CCITT recommendations [CCIT85], etc. Fig.2 shows a state machine diagram of a simple data transfer service. The service provides for transmission of one message at a time from a fixed *sender* to a fixed *receiver*. The sender must wait until the previous message is received before sending the next one. There is no posibility of message loss, duplication or corruption.

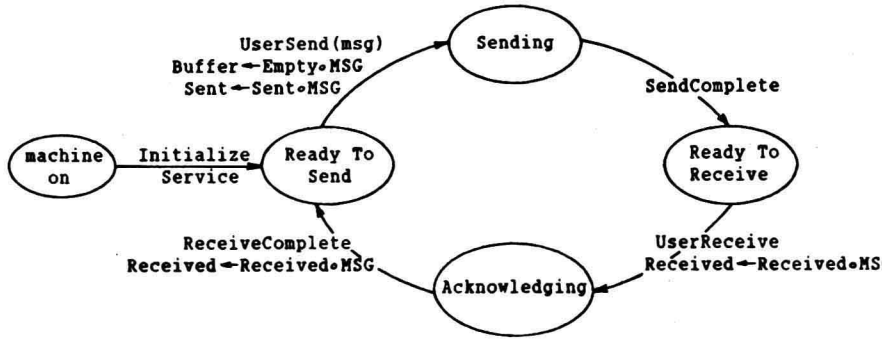

Fig. 2 State machine of a simple data transfer service

The verification technique most commonly used in FSM representations is Reachability Analysis (total or parcial) which involves the construction of a reachability tree describing all (or some of) the states that may be reached starting at a given initial state. The generation of the global state space is easily automated, but the major often quoted disadvantage of this technique is the *state space explosion* [BOCH80b; MERL79] that is, the rapid growth of the number of states to be analysed. Although techniques such as partial specification, sublayer decomposition and state classification by assertions have been developed to restrict this difficulty, in general, the analysis in state machine representation requires large computacional resources.

SDL

The Specification and Description Language (SDL) [CCIT85] is a standardized description technique for application during the development of initial protocol and service specifications, for communication among designers, to increase the degree of accuracy, readability and comprehension and for the formal presentation of specifications published in CCITT recommendations.

SDL is a graphical language based on the concept of communicating sequential processes modelled as extended finite state machines. Recommendation X.250 [CCIT85], defines the semantics of a PASCAL-oriented language in terms of the SDL common language which allows for the translation between them and the production of a program-like form of SDL.

2.2. Programming Languages

High level programming languages such as ALGOL, PASCAL or PL/I have been used [ECMA82] for the specification of algorithms. PDIL [VALE84] and IBM's FAPL [POZE82] are based upon these. Being real *implementation languages* they are not suitable for abstract specification as they rely on the informal understanding of the reader. In favour of their use is the argument that they are machine independent and so appropriate for specification. Very high-level languages such as those employed in *functional* and *logical programming* [HEND80; CLOC84] have given the elements of the language a precise mathematical meaning allowing properties to be stated declaratively. Specification programming languages are,

in general, based on the Extended FSM model.

ESTELLE

The Extended State Transition Language, ESTL, also known as ESTELLE, was developed by the ISO Formal Description Techniques group in collaboration with CCITT. ESTELLE is now a draft proposed standard within ISO to be used by standards committees for the specification of data communication protocols [ISO85b].

In ESTELLE, a protocol system is specified as a set of *modules* components interconnected via *channel* subsystems. Each module is a discrete state machine where state transitions and output functions are specified in PASCAL-like code.

A *channel* is a full-duplex instantaneous signal transmission system which can

```
Specification ABProtocol; {top level module body}
.. .. ..; {var, type, and channel definitions}
module Alternating_bit_type process  {module header definitions}
   (U:U_access_point(Provider) common queue;
    N:N_access_point(User) common queue; <paramlist> param Name:Cep_type);
    .. .. ..; {other module headers}
body Alternating_bit_body for Alternating_bit_type; {module body definitions}
   var Data_transfer: Data_transfer_type; Timer: Timer_type;
   const Retransmit_time = ...; Empty = ...; ...;
   type Msg_type = record ... end; .....;
   channel S_access_point(User,Provider);
      by User: TIMER_request; by Provider: TIMER_response;
   module Data_transfer_type activity
      (U:U_access_point(Provider) common queue;
       N:N_access_point(User) common queue;
       S:S_access_point(User) individual queue; param number:Cep_type);
   module Timer_type activity
      (S:S_access_point(Provider) individual queue; param Time:integer);
   body Timer_body for Timer_type; external;
   body Data_transfer_body for Data_transfer_type;
      var B: Ndata_type; Send_seq: Seq_type;
      state: (ACK_WAIT,AB_P); .. .. ..;
      initialize {initialization of data transfer activity}
         to AB_P
            begin Send_seq := 0; ... end;
      transition
         from AB_P
            to ACK_WAIT
               when U.SEND_request
                  begin P.Msgdata:=Udata; P.Msgseq:=Send_seq;
                        Store(Send_buffer,P); output N.DATA_request; ...
                  end;
         .. .. .. .. ..; {other state transition definitions}
   end; {of data transfer activity body and of module declaration part}
   .. .. ..; {other module bodies}
   initialize {initialization part of Alternating_bit_body}
      begin init Data_transfer with Data_transfer_body(Name);
                 connect Data_transfer.S to Timer.S;
                 attach U to Data_transfer.U; ...
      end;
end; {of the Alternating_bit_body}
initialize {initialization part of the specification}
   begin ... end
end. {of specification}
```

Fig. 3 Sections of a specification of the ABProtocol in ESTELLE [ISO85b]

be invoked by either of two modules designated as USER and PROVIDER. Modules and channels can execute concurrently.

The total state set of a module consists of a *major* finite state set whose values are represented as state variables and *minor* states which are auxiliary program variables. There is no criterion to categorize the state space of a protocol into major or minor states. The choice is subjective.

The primary components of a module specification are the *header* and the *body*. Module headers make explicit any interaction points associated with the module (formal parameter list, exported variables) and are given a header-type identifier which identifies the module as a member of one of two classes: *process* or *activity*. There are rules constraining parallel execution aimed at avoiding deadlocks in an environment of shared data and assure synchronization. Processes may run in parallel with others of the same class and level of hierarchy. Activities must be at the lowest level of the hierarchy subtree and those which are children of the same parent may not run in parallel.

Module bodies consist of a declaration-part, as in PASCAL, an initialization-part defining the initial state of the machine, a transition-declaration-part and a termination-part that defines a procedure which is automatically executed before the module and its resources are released. ESTELLE defines a set of operations on modules and interaction points. Modules may execute concurrently but the use of PASCAL constructs within a module renders some intra-module activity as strictly sequential. ESTELLE cannot monitor all protocol interface situations.

Fig.3 illustrates the structure of a specification in ESTELLE of the ABProtocol. The protocol machine has two states: AB_P and ACK_WAIT [ISO85b].

3. FORMAL SPECIFICATION METHODS

3.1. Petri Net Models

The theoretical applicability of Petri nets as a descriptive model for protocols is broader than, although close to, FSM models. In particular, Petri nets may be used to represent machines with an infinite number of states. Basicaly, it is a graphical model having the disadvantage of the rapid growth of the net with the complexity of the protocol.

In Petri nets, nodes are used to model *conditions*, transition bars are used to model *events* and directed arcs connect nodes to bars and bars to nodes. When a condition holds, a token is placed in the correspondent node and a transition is fired if and only if all nodes leading to a transition bar have tokens (i.e. a transition occurs if and only if all the necessary conditions hold [MERL79]).

To verify a Petri net specification, a Token Machine has to be built which identifies all possible system states and state transitions similarly to what is done in FSM reachability analysis.

To adapt Petri net representation to the specific characteristics of communication protocols new classes of nets were developed. X-Transition Nets allow the specification of alternate paths for a transition through the use of resolution and transition procedures [NUTT72]. In Timed Petri Nets it is possible to specify the minimum and maximum times in which a transition may occur. The two methods may also be combined together [DIAZ82; DANT80].

In spite of the added facilities, Petri nets are limited by the complexity of the protocols which lead to complex and confusing nets and to time consuming verification procedures. Petri nets are not suitable for the divulgation of protocol specifications.

3.2. Algebraic Languages

An algebra is a set of objects, operators for combining objects and axioms that describe the properties of the operators and equivalence relations between expressions. Objects of interest are *symbols* and *Abstract Data Types* (ADT).

There are algebraic techniques better suited for specifying static

characteristics such as properties of data structures or sequencial systems. Other tecniques exist for the specification of dynamic characteristics such as the behaviour of concurrent systems.

In order to specify concurrency by algebraic methods, a system is considered as a collection of *processes* which communicate via *channels*. A process is a sequence of actions (eg imperative statements executing in some machine). Processes progress asynchronously, that is, independently of others, except when they have to communicate.

Communication behaviour can then be represented in terms of algebraic expressions and the algebra axioms give the rules for combining processes.

Algebraic specification languages that support concurrency and have been used in protocol specification and verification are CSP (Communicating Sequential Processes) [HOAR85], CCS (Calculus of Communicating Systems) [MILN80], LOTOS (Language Of Temporal Ordering Specifications) [BRIN84,85] and CUPID (Columbia's Unified Protocol Implementation and Design) [YEMI83]. The latter two are based on Milner's CCS.

AFFIRM [SUNS82], is a running system supporting a set of tools for interactive mechanical theorem proving of sequencial systems.

AFFIRM

AFFIRM [SUNS82] is an experimental system for the algebraic specification and verification of user-defined ADTs. Specifications take the form of axioms of the algebra which are derived from a state transition representation of the protocol. The heart of the system is a natural deduction theorem prover for the interactive proof of data type properties and executes as a functional program written in a PASCAL-like form. It is said to be interactive because it is the user who develops the proof and the system simplifies propositions using the specification axioms.

The methodology of specification and verification involves the following steps:

1) Produce a service specification. Translate its state machine representation into AFFIRM representation;
2) Validate, at least partially, that the service specification meet the requirements of the upper protocol layer;
3) Produce the protocol specification. Translate its state machine representation into AFFIRM representation;
4) Verify that the protocol specification satisfies the service requirements by showing that the axioms of the service specification are theorems provable from the axioms of the protocol specification.

Fig.4 shows sections of a representation in AFFIRM of the service machine in fig.2 [SUNS82]. Each state variable becomes a *selector* function, each state transition function becomes a *constructor* and the axioms state how variables are modified by state transition functions.

In [SUNS82] the ABProtocol is used in a detailed example of a protocol that provides this service. Its specification in AFFIRM is derived from a 6-state machine defining the protocol. Fig.5 lists some sections of this specification.

After the axioms of a service specification have been rewritten as theorems these are then proved from the protocol axioms (fig.5). The proofs often require lemmas. As an example the following theorem is a lemma concerning the relationship between the sender's current sequence number SSN and the sequence numbers of the packets in the medium

```
theorem PktsOldPS,
    all s, m, med(Pending(s) ~= NewQueueOfPacket
                  and PktsOld(s,med)
                     imp PktsOld(ProtocolSend(s,m),med));
```

which says that if there is a packet waiting to be acknowledged (Pending(s)~=NewQueueOfPacket) and packets in the medium *med* are *old*, that is, then they are still old after a *ProtocolSend* event, ie, after the protocol tries

```
type DataTransferService; {state machine}
needs types Message, QueueOfMessage, ControlState;
declare s: DataTransferService, m: Message;
constructors InitializeService, UserSend(s,m), SendComplete(s),
             UserReceive(s), ReceiveComplete(s): DataTransferService;
selectors Buffer(s), Sent(s), Received(s): QueueOfMessage,
          State(s): ControlState;
interface Induction(s): Boolean;
axioms {State:}
   State(InitializeService) == ReadyToSend,
   State(UserSend(s,m)) == if State(s) = ReadyToSend then Sending
                                                     else State(s),
   State(SendComplete(s)) == if State(s) = Sending then ReadyToReceive
                                                    else State(s),
   .. .. .. .. ..;
axioms {Sent:}
   Sent(InitializeService) == NewQueueOfMessage,
   Sent(UserSend(s,m)) == if State(s) = ReadyToSend then Sent(s) Add m
                                                    else Sent(s);
axioms {Received:}
   Received(UserReceive(s)) == if State(s) ReadyToReceive
                                 then Received(s) Add Front(Buffer(s))
                                 else Received(s),
   .. .. .. .. ..;
axioms {Buffer:}
   .. .. .. .. ..;
schema Induction(s) == ...;
nochange State, Sent, Received, Buffer;
end {DataTransferService};
```

Fig. 4 Representation in AFFIRM of service machine of fig.2

```
type ABProtocol; {state machine}
needs types Message, QueueOfMessage, Packet, QueueOfPacket, Medium, Bit;
.. .. ..;
interface Induction(p): Boolean;
axioms {Pending:}
    Pending(ProtocolSend(p,m)) == if Pending(p)=NewQueueOfPacket
                                  then NewQueueOfPacket Add MakePacket(m,SSN(p))
                                  else Pending(p),
    Pending(ReceiveAck(p)) == if Seq(Front(ReceiverToSender(p))) = SSN(p)
                                and ReceiverToSender(p) ~= InitializeMedium
                                  then NewQueueOfPacket
                                  else Pending(p);
axioms {other axioms} ...;
schema Induction == ...;
end; {ABProtocol}

type Message;
declare m:Message;
axiom m = m == TRUE;
end; {Message}
   .. .. .. .. .. ..; {other types}
{Asserted procedures, ProtocolSend(m:Message;var success:Boolean), ...:}
   .. .. .. .. .. ..;
```

Fig. 5 A section of the AFFIRM specification of the ABProtocol [SUNS82]