Communication Protocols and Error Recovery Procedures*

by

Gregor V. Bochmann
Département d'Informatique
Université de Montréal

The design of multiple computer systems and computer networks poses some interesting new problems and shows some older problems in the light of a new context. We are interested in the problems which are related to the design of communication protocols and error recovery procedures. In the design of computer systems, error recovery procedures have often not received very much attention, although they are essential for the reliable operation of most systems. In the case of multiple computer systems, appropriate error recovery procedures can be used to obtain a very much increased availability of the system services. This is also an advantage of computer networks. Whereas single computer systems can sometimes function to a certain extent without elaborate error recovery procedures, data communication protocols, on the other hand, normally have to contain explicit error recovery procedures, since the underlying communication line is normally not very reliable. Therefore, approaches that have been used for the design of protocols in computer networks could also be useful in the design of local multiple computer systems for obtaining higher reliability and availability.

One important problem in the design of communication protocols is the choice of an appropriate method for specifying protocols. Such a method should have the properties that

(1) a protocol can be specified in a form which is comprehensive, in particular that the complete definition of a protocol can be partitionned into different levels of abstraction;

(2) the specification of a protocol allows to prove certain properties of the protocol and its operation, in particular that the error recovery is effective, and that all possible situations have actually been considered;

(3) given the specification of a protocol, its implementation is simple, and part of the implementation may be obtained automatically.

* Work supported in part by the National Research Council of Canada.

The different situations a communication protocol has to cope with, are well described in [1]. In any multiple computer system, a realistic scheme of interprocess communications must foresee appropriate actions for the following situations:

(a) Normal communication between processes.

(b) Occasional erroneous behavior of one or several of the processes. This includes the occasional malfunction of the communication lines. The recovery procedure will normally consist of trying the same action again.

(c) Long range erroneous behavior or failure of one or several processes or subsystems. The recovery procedure will normally consist of a reconfiguration of the system [2,3].

It seems that at present, there is no satisfactory method for specifying protocols for interprocess communication satisfying all these requirements. The author believes that research in this direction should be done from both points of view, practical as well as theoretical. The final objective of building multiple computer systems is a practical issue, but theoretical investigations on the tools to be used are important for understanding and solving the problems involved. Similarely as one needs a well-defined programming language for writing reliable programs, one needs a well-understood method for the specification of protocols in order to be able to design reliable multiple computer systems with high availability. In fact, the problem can be considered a special case of software engineering.

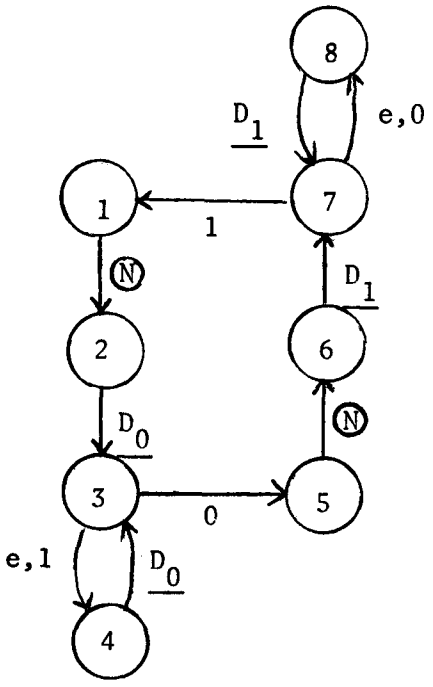
We describe in the following a method for proving properties of protocols [4]. In particular, this method can be used to prove the correct operation of a given protocol. We show as an example, how this method applies in the case of a simple protocol for data transmission with recovery of transmission errors. This method applies wherever the protocol is specified by a finite state description of the interacting processes [5]. We believe that the proving method can be conveniently adapted to more general specification methods.

Finite state description of protocols

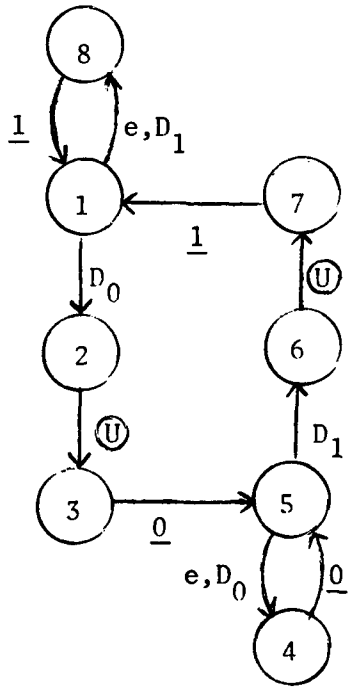
In the simplest case, we consider a system of two subsystems which communicate via a communication line. Each subsystem can be described in terms of a finite number of states and transitions between these states. The transitions are associates either with a particular local action of the subsystem or with the communication line between the two. We suppose that the two systems communicate by exchanging messages of a certain type, and that transmission errors will be detected by a lower level line control, not described here, which indicates an error reception whenever an ill-formed message is received. Then the possible transitions between two states of a subsystem are:

1. local action transitions, written "$\xrightarrow{\sigma}$" where σ represents the associated action, performed during the transition.
2. sending transitions, written "$\xrightarrow{\underline{m}}$" where m is the message send over the communication line.
3. receiving transitions, written "$\xrightarrow{m}$" where m is the message (correctly) received, or "$\xrightarrow{e}$" indicating the reception of an ill-formed message.

As an example we consider the data transmission protocol with alternation bit described in [6]. We consider data transmission from subsystem A to subsystem B only, and can describe the whole system by the following state diagrams:



Subsystem A



Subsystem B

We note that the actions N and U are filling the data buffer of subsystem A with new data, and using the data in the buffer of subsystem B respectively, and that the correct reception of one of the messages D_o or D_1 by system B implies that the content of the data buffer of system A has been copied into the data buffer of system B .

Synchronization and adjoint states

Synchronizing several subsystems or processes means introducing a correlation in the time sequence of the execution of their respective actions such that certain global requirements are satisfied. In order to verify whether these global requirements are satisfied for a particular system of interacting subsystems, one can consider the possible combinations of states in which the subsystems can be at any instant of the execution [5]. In [4] we show for the case of two communicating subsystems A and B, how one can determine the set A_s of states in which B could possibly be at the time when subsystem A is in the state s . We call A_s the set of adjoint states of s . The smaller the sets of adjoint states are, the closer is the synchronization between the two subsystems. If for some state s of A the set of adjoint states is the set of all states of subsystem B then system A does not know anything about system B, when it is in the state s (the synchronization is lost).

For the example above we obtain the following sets of adjoint states:

$A_1 = \{1\}$ $\quad$ $A_5 = \{5\}$

$A_2 = \{1\}$ $\quad$ $A_6 = \{5\}$

$A_3 = \{2,3,8,4\}$ $\quad$ $A_7 = \{6,7,1,8\}$

$A_4 = \{5,1\}$ $\quad$ $A_8 = \{1,5\}$

This means, for instance, that subsystem B must be in its state 1 when subsystem A is in its state 1, or that subsystem B could be in one of the states 2,3,8, or 4 when subsystem A is in its state 3 .

In this example, the above sets of adjoint state are a stable solution for the whole system, independent of the question of initial synchronization. This means that this protocol establishes synchronization even if initially, there was none.

Analysis of action sequences

In order to prove the correct operation of a protocol which describes the communication of several processes we consider the set of all possible action sequences (also called computation sequences) which can be obtained by the protocol, and ask whether this set corresponds to what we intend the protocol to do. The notion of action sequences has been used to compare different models of parallel computation [7]. An extension of regular expressions for action sequences has been used to describe synchronization requirements for operating systems [8]. Since we are here concerned with protocols which are described in terms of finite state machines, we can describe the action sequences in terms of regular expressions.

For a given protocol, the actions which we include in the analysis of the action sequences are those which are relevant for the correctness proof of the protocol. These may include local actions of subsystems, as well as the actions of sending or receiving messages. In the case of the above example, we want to prove the correct transmission of data. One sees that the action sequence NDU (where D stands for the correct reception of a data message by the subsystem B, i.e. transition D_0 or D_1) implies the correct transmission of one block of data. In order to prove the correctness of the protocol, it would therefore be sufficient to prove that the possible action sequences are described by the regular expression $(NDU)^\infty$, where $(...)^\infty$ stands for an indefinite repetition of the enclosed expression. But we note that $(NDD^*UD^*)^\infty$ would be sufficient, too.

Given the specification of a communication protocol in terms of a finite state description, as discussed above, one can determine the set of possible action sequences the protocol can give rise to [4]. In the case of the example above, with initialization of both subsystems in the states 1, we obtain the expression

$$(NDUD^*)^\infty$$

for the possible action sequences. This means that any action sequence that could occur consist of the indefinite repetition of sequences of the form N,D,U, followed possibly by one or more actions D . This expression does not contain symbols which represent the actions of retransmitting the messages 0 or 1 , or of receiving an ill-formed message; those actions are not explicitely included in the action sequences. The above expression then proves the correctness of the protocol as discussed above. It also

proves that there is no deadlock. (Note that we have ignored the problem of a message which is completely lost. This problem can be solved by a time-out mechanism in one of the subsystems, which results in an error transition.)

Another example is the protocol for data transmission given in reference [9]. This protocol is not correct because it falls into a loop as soon as an acknowledge message is received erroneously by subsystem A . This fact is reflected in the regular expression one obtains [4] for the action sequences:

$$NDU \quad (NDU)^* \; D^{\infty}$$

which means that a typical action sequence consists of a certain number of correct data block transmissions followed by an indefinite sequence of D . In deriving this expression, one finds immediately that this indefinite sequence of D is the result of looping between certain states.

We are presently working on applying a similar proof method to more complex communication protocols, of the kind used in the Arpa network. We also investigate if the concept of path expressions [8] can be useful as a tool for describing communication protocols. Certainly, more work has to be done in this area, in order to obtain convenient protocol description methods and other tools for designing multiple computer systems with high reliability and availability.

References

1. L. Pouzin, Network protocols, Nato International Advanced Study Institute, Brighton, Sept. 1973.
2. R.S. Fabry, Dynamic verification of operating system decisions, Comm. ACM 16, 659 (1973).
3. "Resource sharing computer networks", Special session at the SJCC, Atlantic City, 1970.
4. G.V. Bochmann, On proving protocols correct, Technical report, September 1974.
5. P. Gilbert and W.J. Chandler, Interference between Communicating parallel processes, Comm. ACM 15, 427 (1972).
6. K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, Comm. ACM 12, 260 (1969).
7. J.L. Peterson and T.H. Bredt, A comparison of models of parallel computation, Proceedings IFIP Congress 74, p. 466-470 (1974).
8. R.H. Campbell and A.N. Habermann, The specification of process synchronization by path expressions, Techn. Report # 55, Univ of Newcastle upon Tyne, Jan 1974.
9. J.P. Gray, Line control procedures, Proc. IEEE, Nov 1972, p. 1307.