[7] E. Cohen and D. Jefferson, "Protection in the Hydra operating system," in *Proc. 5th Symp. Oper. Syst. Principles*, vol. 9, no. 5, 1976.
[8] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, pp. 993-999, Dec. 1978.
[9] J. E. Donnelley and J. G. Fletcher, "Resources access control in a network operating system," presented at the ACM Pacific '80 Conf., Nov. 1980.
[10] *iAPX-432 General Data Processor Architecture Reference Manual*, Intel Corp., 1981.
[11] S. J. Mullender and A. S. Tanenbaum, "Protection and resource control in distributed operating systems," *Comput. Networks*, vol. 8, pp. 421-432, Nov. 1984.
[12] S. Rivoira and A. Valenzano, "A distributed operating system for object-based machines," in *Proc. Int. Conf. Parallel Processing*, Bellaire, Aug. 1984, pp. 46-50.
[13] K. Ramamritham, D. Stemple, D. A. Briggs, and S. Vinter, "Privilege transfer and revocation in a port-based system," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 635-648, May 1986.
[14] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The kernel of a multiprocessor operating system," *Commun. ACM*, vol. 17, pp. 337-345, June 1974.
[15] G. T. Almes, A. P. Black, E. D. Lazowska, and J. D. Noe, "The Eden system: A technical review," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 43-59, Jan. 1985.
[16] D. R. Cheriton, "The V kernel: A software base for distributed systems," *IEEE Software*, vol. 1, pp. 19-42, Apr. 1984.
[17] A. S. Tanenbaum and R. Van Renesse, "Distributed operating systems," *ACM Comput. Surveys*, vol. 17, pp. 419-470, Dec. 1985.

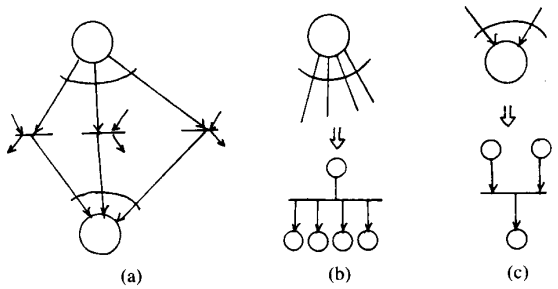

Fig. 1. (a) PAR construct. (b) Meaning of initial state. (c) Meaning of final state.

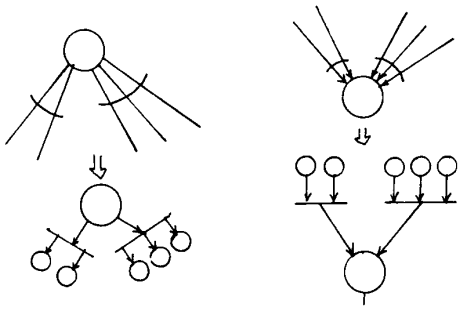

Fig. 2. (a) Meaning of initial state for ALT. (b) Meaning of final state for ALT.

A Simplification of a Conversation Design Scheme Using Petri Nets

JIE WU AND EDUARDO B. FERNANDEZ

Abstract—**The conversation scheme is a promising way of developing fault-tolerant software and several mechanisms for its implementation have been proposed. In an earlier paper Tyrrell and Holding used Petri nets to design conversations. Their procedure, although correct, appears to be more complex than necessary. In this correspondence, a simplified transition identification method is proposed. Using a robot arm control program we show that the corresponding Petri net graph is simpler than the one proposed by Tyrrell and Holding, but the communication state change table is the same. It is also shown that these two methods are equivalent.**

Index Terms—**Conversations, fault-tolerant software, Petri nets.**

I. Introduction

The conversation scheme has been proposed as a promising way of developing fault-tolerant software [1] and several mechanisms for its implementation have been considered [2]. Tyrrell and Holding [3] use Petri nets to design conversations. Their procedure, although correct, appears to be more complex than necessary.

In the design procedure, the definition of the state of the system is one of the most important aspects. The question is how to identify transitions in Occam programs in order to express them as Petri nets. In this correspondence, a simplified transition identification method is proposed. Using their robot arm control program we show that the corresponding Petri net graph is simpler than the one in [3], but the communication state change table is the same. It is also shown that these two methods are equivalent.

Manuscript received June 30, 1987; revised October 31, 1988.

The authors are with the Department of Electrical and Computer Engineering, Florida Atlantic University, Boca Raton, FL 33431.

IEEE Log Number 8926737.

II. Transition Identification

For convenience, we use a slightly modified representation for Occam constructs. Three rules are needed to model Occam programs by Petri nets:

1) For communication and synchronization we use the scheme of [3].

2) For PAR we define the constructs shown in Fig. 1.

3) For ALT we define constructs as shown in Fig. 2. We show in this case the initial and final states separately since we never need them together as in [3].

Note that this PAR model is equivalent to the one in [3], only the representation has changed. This means here that only one token is needed in the starting place. Similarly for the ALT construct. This notation allows to conveniently hide intraprocess communication transitions as shown later.

The essential part of a conversation is the boundary which is used to prevent error propagation. The intraprocess communications are the main concern in building the scheme, while the intraprocess communications should be removed as early as possible. It is advantageous to remove intraprocess communications before forming the state change table rather than after forming this table as it is done in [3].

The simplified identification method can be defined as follows:

1) Define as transition only those statements that indicate interprocess transfers, e.g., $X!a$, $X?a$.

2) Define one state between every two transitions.

3) Define separately the initial and final states for each process.

In general, there will be fewer states in these Petri nets, but the recovery points and test points that we can choose remain the same.

```
Robot Example. OCC
...
PROC operator (...) =
    VAR ...:
    SEQ
      ...
      WHILE
      SEQ
        ...
          send ! x                          --(t2)
          send ! y
          send ! z

          receive ? ANY                     --(t3)

          IF
            ...
              SEQ
                PAR ...
                  stop [i] ! ANY            --(t5,t16,t22,t28)
              ...
              TRUE
                PAR ...
                  go [i] ! ANY :            --(t8,t18,t24,t30)
PROC motor (...) =
    VAR ...:
    SEQ
      ...
      WHILE ...
        SEQ
          motion ? step                     --(t10,t19,t25)
          motion ? direction
          ...
          finished ! ANY                    --(t13,t21,t27)
          ALT
            stopi ? ANY                     --(t16,t22,t28)
              ...
            goi ? ANY
              SKIP :
PROC control (...) =
    VAR ...:
    SEQ
      ...
      WHILE ...
        SEQ
          receive ? xnew                    --(t2)
          receive ? ynew
          receive ? znew
        ...
        PAR
          SEQ
            motion [i] ! step [i]           --(t10,t19,t25)
            motion [i] ! direction [i]
            ...
        WHILE ...
          ALT ...
            finished [i] ? ANY              --(t13,t21,t27)
          ...
        send ! ANY                          --(t3)

        ALT
          stop i ? ANY                      --(t5)
            ...
          goi ? ANY                         --(t8)
            SKIP :

...
PAR
    PAR i = [0 for 3]
        motor (...)
    Control (...)
    Operator (...)
```

Fig. 3. Occam program for 3-axis robot arm controller.

III. Robot Control Example

We use the proposed method to identify transitions in the robot control example. In order to make a comparison we use the same labels in the statements as in the example of [3]. Fig. 3 shows this Occam program. Some irrelevant details of the control program have been omitted and we emphasize only the statements that contain transitions.

The Petri net model of this program is shown in Fig. 4. To compare it to the method in [3], we number the states as the states just before transitions in Fig. 1 of [3]. Note that this results in slight changes in some state numbers, e.g., P_1', now becomes P_2'.

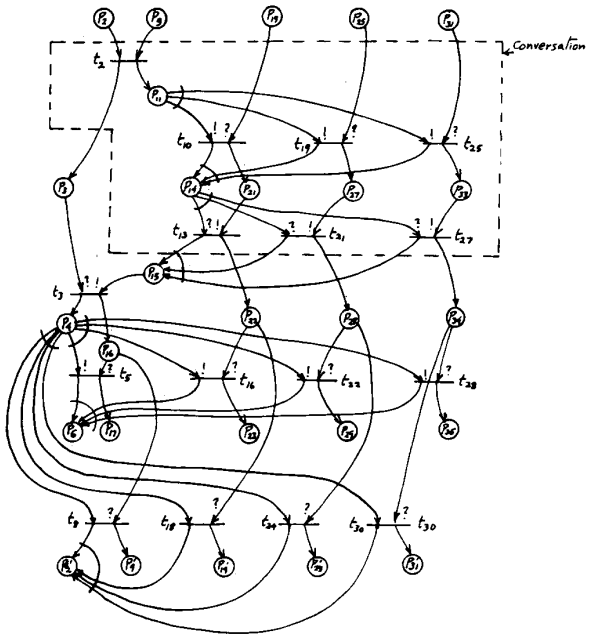

Fig. 4. Petri net graph of robot arm controller.

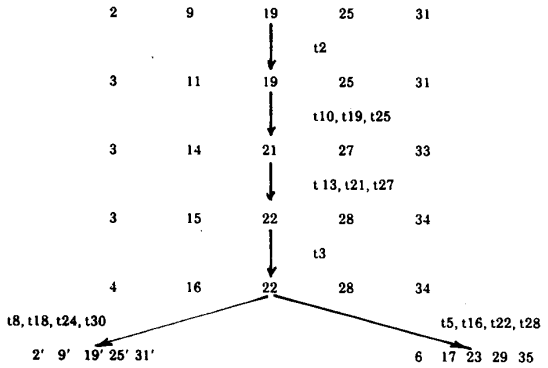

Fig. 5. Reachability tree of the graph of Fig. 4.

Transitions	I	E
t2	2, 9	3, 11
t10,t19,t25	11,19,25,31	14,21,27,33
t13,t21,t27	14,21,27,33	15,22,28,34
t3	3,15	4,16

t8,t18, t24,t30	4,16,22, 28,34	2',9',19' 25'31'	t5,t16, t22,t28	4,16,22, 28,34	6,17,23, 29,35

Fig. 6. State change table of Fig. 5.

The reachability tree of the Petri net can be formed from the set of all next state functions and is shown in Fig. 5.

The state-change table lists the state changes for each transition in the reachability tree. The elements of the state-change table can be identified as in [3]. The state-change table is shown in Fig. 6.

The state change table here is equal to the communication state change table in [3] (remember that $2' = 1'$ in the original paper). In fact, in Fig. 6 for tj there exist $p1$ and $p2$ where:

$$p1, p2 \in Ij \quad \text{or} \quad p1, p2 \in Ej \quad \text{and}$$
$$p1 \in \text{PROC}q;\ p2 \in \text{PROC}r\ (q \neq r),$$

so there is no intraprocess communication.

IV. Summary

We have suggested a simplification to the method proposed by Tyrrell and Holding in [3] for the design of conversations for reliable software. Our method considers only interprocess communication and results in a change state table that also contains all the information of the communication state table. The resulting conversation is equivalent to the previous case with respect to recovery points and test points. Since Petri nets for real programs are complex, we believe this simplification can be of practical value for fault-tolerant software design.

As pointed out by one of the reviewers, this simplification does lose some of the state information. If a conversation is required to start or finish at a state which is not a communication, this method will only provide an approximate boundary. This is also true for approaches such as the refinement method of Reisig [4], where a single place is replaced by a net of arbitrary size. In fact, the reachability tree for the Petri net representation of [3] gives only one possible execution order of the net, and as such it also loses information. A study of this aspect is currently being performed [5].

References

[1] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 3, pp. 221–232, June 1975.

[2] K. H. Kim, "Approaches to mechanization of the conversation scheme based on monitors," *IEEE Trans. Software Eng.*, vol. SE-8, no. 3, pp. 189–197, May 1982.

[3] A. M. Tyrrell and D. J. Holding, "Design of reliable software in distributed systems using the conversation scheme," *IEEE Trans. Software Eng.*, vol. SE-12, no. 9, pp. 921–928, Sept. 1986.

[4] W. Reisig, "Petri nets in software engineering," in *Petri Nets: Applications and Relationships to Other Models of Concurrency* (Lecture Notes in Computer Science vol. 255). New York: Springer-Verlag, 1987.

[5] J. Wu and E. B. Fernandez, paper in preparation.