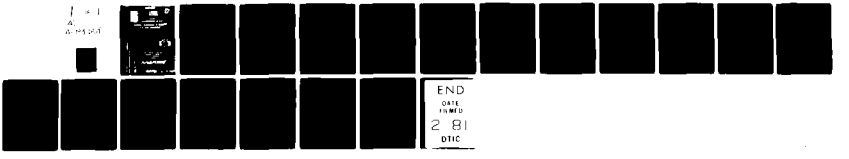
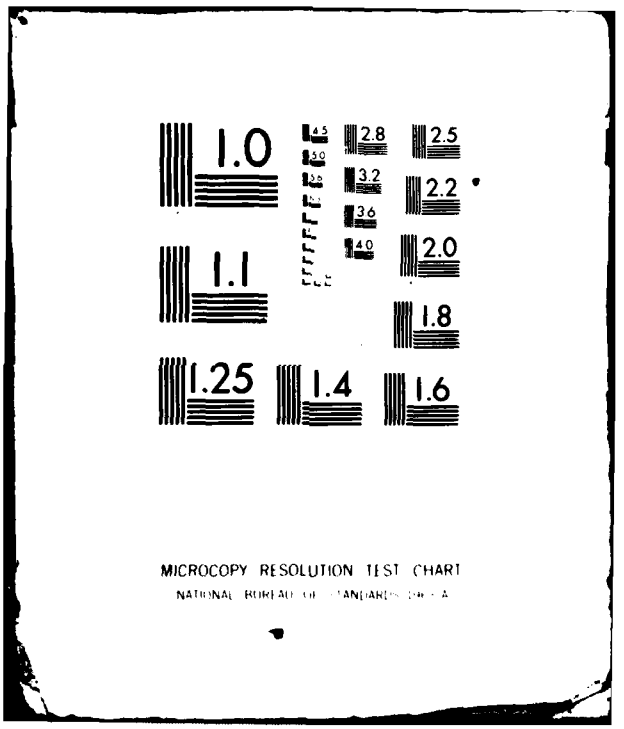
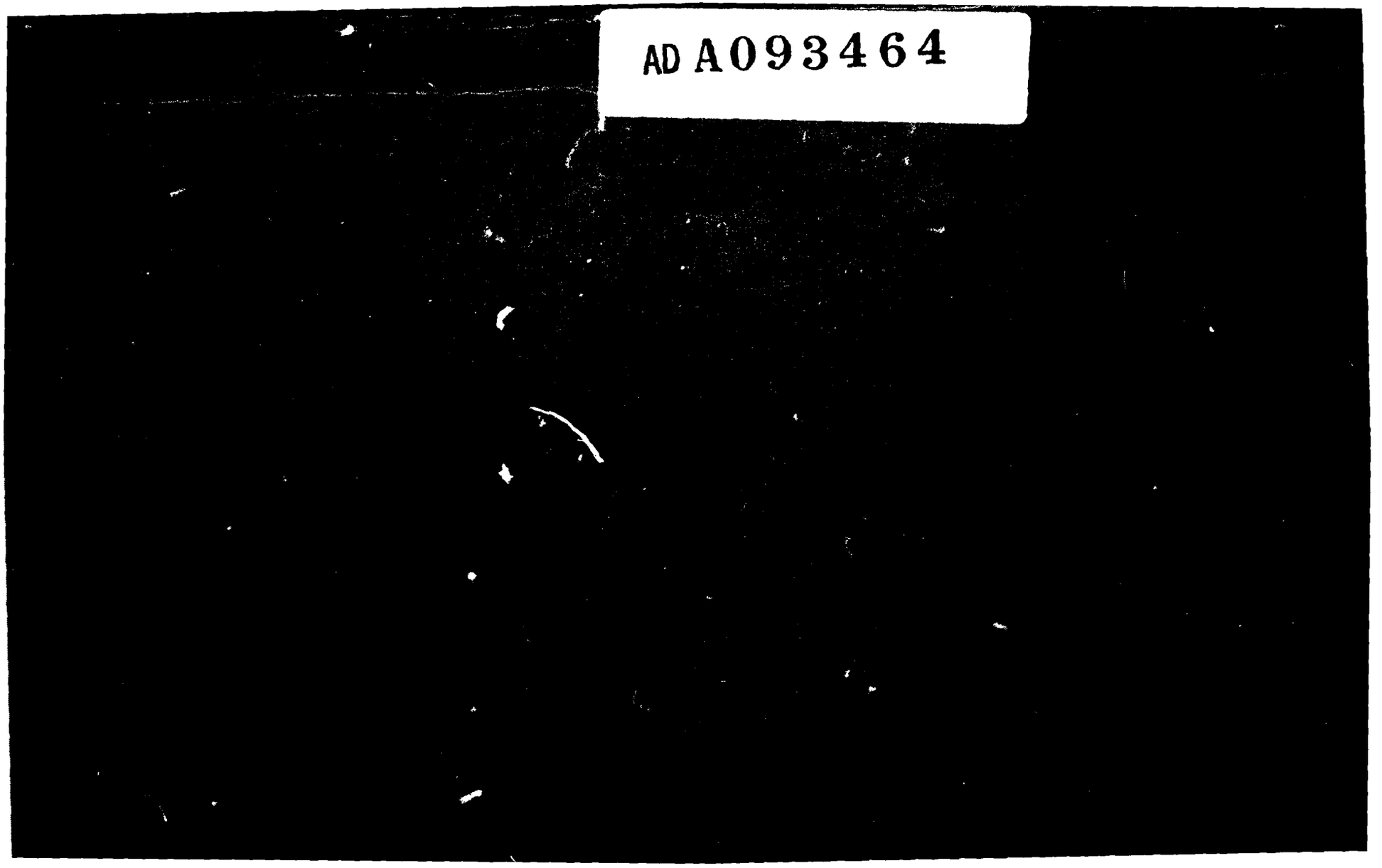
AD-A093 464 ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND) F/6 9/2
A PRELIMINARY IMPLEMENTATION OF POSER,(U)
SEP 80 S J BEVAN
UNCLASSIFIED RSRE-MEMO-3272 DRIC-BR-76603 NL



END
DATE FILMED
2 81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A



(18) DRIC (19) BR-966013

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 3272

Title: (6) A PRELIMINARY IMPLEMENTATION OF POSER

Author: (10) S J Bevan

Date: (11) September 1980 (12) 78

(14) RSRE-MEMO-3272

SUMMARY

POSER is a process organisation to simplify error recovery intended for use in fault - tolerant, distributed computer systems running real - time programs.

This memorandum describes the process organisation used in POSER and how the organisation has been experimentally implemented in a multi-computer simulation. Application program design has been studied by producing a large radar tracking program which runs on the POSER simulation. A version of the radar program exists in MASCOT and some comparisons of the two complete programs have been made. Finally, some broad comparisons of the MASCOT and POSER methods are made.

409929

SUMMARY

POSER is a process organisation to simplify error recovery intended for use in fault - tolerant, distributed computer systems running real - time programs.

This memorandum describes the process organisation used in POSER and how the organisation has been experimentally implemented in a multi-computer simulation. Application program design has been studied by producing a large radar tracking program which runs on the POSER simulation. A version of the radar program exists in MASCOT and some comparisons of the two complete programs have been made. Finally, some broad comparisons of the MASCOT and POSER methods are made.

A PRELIMINARY IMPLEMENTATION OF POSER

0) INTRODUCTION.

This document describes the facilities provided for the application programmer by POSER, and tries to evaluate the efficacy of these facilities. POSER was originally described in [1] but in order to make the document self - contained, some of the basic ideas are repeated here.

POSER is a process organisation intended for use in real-time, distributed computer systems, and is in many ways similar to MASCOT [2]. The main differences are that POSER does not allow shared data areas; POSER synchronisation is implicit, not explicit; and the motivation of POSER was to simplify the implementation of the error recovery mechanisms needed to make computer systems fault tolerant. Inter process communication in POSER is constrained to be entirely by message passing - i.e. we have a data flow machine at the highest level of abstraction. A general introduction to the philosophy of fault - tolerance and error recovery can be found in [3], and a background to the POSER proposals in [4].

In a practical implementation, a "POSER Kernel" would be provided giving facilities similar to those found in a MASCOT "frozen system". The kernel will contain the scheduling facilities, and mechanisms for performing error recovery.

1) POSER - a brief description.

1.1) Processes and Channels.

A POSER "process" is the counterpart of a MASCOT activity, and the POSER "channel" corresponds to a MASCOT "channel". The important distinction between the two types of channel is that a MASCOT channel is simply a data area with some associated access procedures to this data, whereas the POSER channel is another process constrained to perform only a message passing function between POSER processes. Thus both processes and channels can be scheduled in the POSER system.

Processes and channels consist of four objects:

Process code.
Own data.
Space for input parameters.
Space for output parameters.

The own data is private to the process or channel of which it forms a part and the process code uses this, together with input parameters, to form results which are put in the output space. The kernel is responsible for the movement of data from the output space of one process or channel to the input space

of the next.

The process is allowed to modify or merge input parameters to form single or multiple outputs. The only function of a channel is to provide buffering between processes and to ensure correct process connectivity. Channels may have multiple inputs, but only a single output. This simplifies the connection between the processes and in most cases allows processes to be designed with a single input and a single output.

In a similar manner to a MASCOT Activity Channel Pool (ACP) diagram, the interconnections between the POSER processes and channels can be expressed in a diagram. The POSER process is drawn as a box containing the process code and owndata and the channel as a bar. The interconnections are shown by drawing directed arcs between processes and channels. An example of a simple POSER system is shown in figure (1.1).

1.2) Communications.

The processes and channels communicate entirely by message passing, thus the data flow defines the process synchronisation. Since POSER prevents the use of shared data areas, processes which wish to share information can only do so by passing copies of the information to each other. It is this data flow foundation that makes the process organisation conducive to automatic error recovery.

1.3) Error Recovery.

The details of the mechanisms used by the POSER kernel for error recovery are not relevant here, but the applications system is, to some extent, influenced by these mechanisms. In particular it is necessary for the applications system to tell the kernel what information it requires to be preserved in the event of an error occurring. This can best be done by giving the kernel a procedure which can copy the salient data.

2) PRACTICAL IMPLEMENTATION.

2.1) Multi Computer Simulation.

In order to try out some of the ideas put forward by POSER, a computer program has been written which simulates a distributed computer system. Each computer has its own POSER kernel and a network is provided so that any computer may communicate with any other.

This experimental simulation has been produced in ALGOL68RT [5,6] which runs on RSREs main computer, an ICL 1900. This implementation provides facilities for inter - connecting processes and channels and a scheduler based on the data flow between processes and channels. The checkpointing and error recovery mechanisms have not been included. There is

also an implementation of MASCOT on the main computer so some comparisons can be made between the two programming methods.

The simulation system suffers from several problems which could only have been solved by using special machine code instructions. As a result, the user has access to some of the kernel data areas which would be hidden from him in a proper system.

These problems are being tackled in a new simulation which is currently being written to run on the FLEX computer [7]. The new simulation will provide a cleaner user interface and the kernel data areas will be inaccessible to the user. The checkpointing and error recovery mechanisms are being included in this simulation.

2.2) Process and channel shells.

Each process and channel is kept within a "shell" which contains the owndata, process code, parameter spaces and a scheduling condition. Since the only difference between a process and a channel is in the process code and owndata, it suffices to describe a process shell.

The process shell in the present simulation is an ALGOL68 procedure which takes as parameters the space for the inputs and outputs to the process, the number of the process and a data area which contains information about the process for the kernels use. The format is:

```
PROC process shell = ( [] REF PARAM input, output,
                       INT process number,
                       REF MYPROCESS this process
                     ):
    BEGIN
         .........
         .........
         .........
    END;
```

Inside the shell, the user must declare the owndata, process code and the scheduling condition for the process. The scheduling condition is put into the kernel data area and is used by the kernel scheduler which is described in (2.4).

A detailed example of a process shell is given in the appendix.

2.3) Setting up processes and channels.

The processes and channels used in a POSER system have to be specified with interconnection information so that the kernels can provide the links necessary for the data flow. This is equivalent to the MASCOT "form" command for setting up subsystems.

A computer for each process or channel must be given and the shell is loaded onto that computer with space allocated for the input and output parameters. The transfer of data between the parameter spaces is taken care of entirely by the kernel.

2.4) Scheduling.

Each computers kernel has its own scheduler which operates on the processes and channels loaded on the computer. The scheduler calls the users running condition for each shell in turn until a runnable process or channel is found. The process code of the process or channel is then executed. After this "activation", control is returned to the scheduler.

For efficiency, the scheduler was later modified to only evaluate the condition if there had been a change in the state of the shell parameters since it was last called. This improved scheduler gave a considerable improvement in scheduling overhead. Some timing results obtained using a large test program will be presented later.

3) WRITING APPLICATION PROGRAMS.

3.1) Overall design.

The breakdown of a real-time problem into the various processes and their interconnecting channels must be done before any detailed programming can occur. This task should be within the capabilities of an experienced programmer who will also decide the running conditions for each process. The production of the process code resembles normal sequential programming and could be written by a less experienced programmer.

3.2) Process code.

The programmer has to provide a procedure which transforms the input parameters provided by the shell and sends the result to the shell output parameters. It is necessary for the programmer to check the validity of the input data received since he and not the kernel knows what is expected.

With valid input data, the process code can operate using any data local to the process shell. Data required from another process must have been previously requested and made available to the process as input parameters. To simplify the process code, the user can, of course, define local procedures but these must be declared inside the process shell.

3.3) Scheduling.

At present the kernel requires an element of the MYPROCESS belonging to each shell to be set by the user. This is a procedure called "runnable" which delivers true if the shell

is available for activation.

In all cases, since data flow controls all synchronisation, a process or channel will be runnable if a subset of the input parameters are full and the appropriate output parameters empty. A typical case might be:

```
runnable OF this process := BOOL:
   BEGIN
     ((full OF input[1]) AND (NOT full OF output[1]))
      OR
     ((full OF input[2]) AND (NOT full OF output[2]))
   END;
```

In this case, the process can either take data from input 1 and deliver to output 1 or take from input 2 and deliver to output2.

3.4) Synchronisation.

Since all processes and channels are scheduled by the arrival and departure of data, normal synchronisation will cause no problems. If a process needs three sets of data from three different channels, making "runnable" deliver FALSE unless all three inputs are full will achieve the necessary synchronisation.

There are, however, some limitations on synchronisation. To suspend a process until all others have run as far as they can is such an example. In MASCOT there is provision for attaching a priority to an activity so that high priority activities will be scheduled in preference to low ones. If the simulation is to be used for a wide class of programs, this is one area that will need attention.

4) TEST PROGRAM: A Radar Tracking Simulation.

4.1) Program Description.

In order to investigate application program design and to compare MASCOT and POSER, a large radar tracking program, written in ALGOL68, was converted from MASCOT to POSER.

The program splits into four subsystems:

1) A radar simulation which can be controlled by input from a keyboard. This produces a simulated radar display with the usual sector sweeps.

2) An aircraft tracking system which can be manually updated from the keyboard. In the absence of manual updates, the tracker estimates position using the previous manual updates.

3) A control system for co-ordinating the two main subsystems.

4) An output system to produce the radar picture on the terminal.

The original MASCOT system used some special peripherals which are no longer available and some of the activities were used to transfer data to and from these peripherals. The system which was modelled consisted of seven activities, six channels and five pools and had simplified input - output. An ACP diagram of this system is shown in figure (4.1).

4.2) The new program.

The POSER program tried to keep the operation of the radar and tracking subsystems as close to the original as possible. The control system was simplified and the output system changed to cope with the different peripherals.

Since POSER does not allow pools of data with many processes sharing access, some means of achieving data sharing had to be found. The approach used was a Database Manager process whose own data forms the pool. Other processes must request data from the manager and wait for it to be sent out along the channels.

Of the five pools used by the original program, only three were accessed by more than one activity; the others could quite easily be incorporated in the owndata of a process. The three large datapools were merged into two; one for the tracking system and one for the radar simulation. Each pool was implemented as the own data of a database manager process with each manager accepting commands only from its own subsystem.

In all, ten processes were set up; two new processes came from the two database manager processes, and a third from a new keyboard process. To provide the necessary communications, twelve channels were required. Each channel had a fifteen item buffer and operated on a first in - first out principle with equal priority on all inputs.

A diagram of the POSER process organisation is shown in figure (4.2).

4.3) Process Scheduling and Synchronisation.

The original program worked by having the two main systems, tracking and simulation, controlled by a common timekeeper process. By using the priority mechanism, the timing activity could be suspended until all activities were waiting for a new timing stimulus. At this point the timer was reactivated and the program continued. The scheduling of the individual processes was controlled by direct stimulation with true parallel processing being used when more than one process was active.

The POSER program was run completely by message passing between processes and consequently the order in which processes were called was fairly well known. The manual interactions with the tracking system, however, made full use of the parallel processing facilties since keyboard commands could be processed at the same time as normal system updates.

The corresponding timekeeper process worked by waiting for messages to be sent out to it from the two main subsystems indicating that all possible updating had been completed and then increasing time for both subsystems. The message passing system was also used for the keyboard interfaces: keyboard commands would not be acted upon unless the database was free at that time.

All the runnable conditions used were kept as simple as possible and simply looked for a certain subset of the shell parameters to be in a suitable state. Since some processes had several possible alternative runnable conditions, the modified scheduler significantly reduced the scheduling overhead. Some timing results will be presented in section (5).

4.4) Improvements and Shortcomings.

The greatest improvement in the new program is that of data management. The bulk of the data kept in the MASCOT pools has been moved into the owndata of database processes and the rest into owndata of individual processes. Hence greater data security is obtained together with a well defined data flow in and out of the databases.

The increased number of processes and channels has made the size of the new program larger than the old. For a large number of processes, a large number of interconnecting channels are required which in most cases at least doubles the total number of shells in the system. Since MASCOT pools have to be replaced by database processes and associated communications channels, the number of parallel processes is increased further. Thus there will, in general, be more code in a POSER program than in an equivalent MASCOT program.

5) EFFICIENCY.

5.1) Comparison of POSER Schedulers.

Some timing runs of application programs were carried out which were intended to give some idea of the relative efficiency of the two POSER schedulers. In all cases the runnable conditions in the application program were left unchanged.

Running the Radar Simulation program gave the following

results:

Old scheduler.
average time to schedule 1600 micro-seconds.

New scheduler.
average time to schedule 1100 micro-seconds.

5.2) MASCOT and POSER schedulers.

In order to investigate the efficiency of similar implementations of MASCOT and POSER, two new programs were written, one in ALGOL-68 MASCOT and the other using the POSER simulation. In both programs, two parallel processes were set up, each process sending a stimulus to the other and then waiting for the stimulus to be returned.

In MASCOT this was done by having two activities and two control queues, one for each activity. The activity, when scheduled, "stim"s the other ones queue and then waits for its own queue to be stimmed.

In the POSER program, a process and a channel were set up on a single computer. When scheduled, the process emptied its input parameter and filled its output parameter. The scheduling conditions and connectivity used meant that the channel shell was now runnable and the process not. The channel then emptied its input and filled its output making the process runnable and the channel not.

The results obtained were:

MASCOT
Average time between schedules 820 micro-seconds.

POSER
Average time between schedules 560 micro-seconds.

5.3) Conclusions.

The old POSER scheduler was very slow if the application programmer wrote long, complicated runnable procedures since it actually called each procedure as it tried to schedule a process. By only calling the procedure when a shell parameter had changed, the scheduling overhead was substantially reduced. If the program is written so that there are only a few process shells runnable at each schedule time, the scheduling overhead will increase. Neither scheduler will give good results in this case, but the modified one will waste less time searching.

The comparison between the ALGOL 68 implementations of MASCOT and POSER is not a very useful one because the mechanisms used are so different. The MASCOT implementation keeps lists of activities and adjusts these lists at each schedule with the result that the schedule time is largely independent of the number of activities. Since the POSER

scheduler searches through all processes and channels for a runnable shell, more work will have to be done before a shell can be activated. The fact that MASCOT moves items which are not currently availiable for running off the main scheduler list means that for a large number of activities, the time to select one to run will be much smaller than the corresponding time using POSER. However, for programs with only a few processes, the search time will probably be less than the time to manipulate the lists. This probably explains why the POSER program gave the faster schedule time.

6) GENERAL CONCLUSIONS.

In the present, experimental, state it is difficult to make valid comparisons between POSER and MASCOT. The areas of difference allow some general observations to be made.

1) The well organised data flow in a POSER program should result in better overall clarity. Since each process is a self-contained unit, it should be possible to see how each process works by simply studying the process code, the running condition and the interconnections to other processes. Thus a clarity at the process level is achieved.

2) At present there is no mechanism provided by POSER for attaching a priority to a process so certain types of process synchronisation are more difficult to achieve.

3) There is no provision for interrupt handling or interactive input - output on the simulation so the real time aspect of some programs may be lost.

To write a large program using POSER presents little difficulty provided the operation of the individual processes is well thought out. Since this problem is common to all real time programs the fact that POSER is being used makes little difference.

If the guidelines of single output channels and, where possible, single input processes are used then process synchronisation will be easy to achieve. This will make running conditions straightforward and give each process only a small number of possible ways of being activated. Thus the process code becomes relatively simple and the number of data checks at each activation small.

POSER is an elegant means for writing real-time computer programs with the provision for checkpointing and error recovery. In its present state, the implementation is largely experimental and a little untidy but this does not hinder the programmer too much. The FLEX program, when completed, will give a simpler and more secure basis from which POSER can be studied.

7) REFERENCES.

[1] "POSER - A Process Organisation to Simplify Error Recovery."
RSRE memorandum 3249, J.A. McDermid, 1979.

[2] "MASCOT - A Modular Approach to Software Construction, Operation and Test."
RRE technical note 778, K. Jackson, H.R. Simpson, 1975.

[3] "Reliable computer systems."
B. Randell, P.C. Lee, P.A. Treleaven.
Lecture notes in computer science No 60: Operating Systems - An advanced course.
G. Goos, J Hartmanis Eds., Springer Verlag, 1978.

[4] "Fault Tolerant Computing."
RSRE memorandum 3197, J.A. McDermid, 1979.

[5] "ALGOL68R Users Guide."
P.M. Woodward, S.G. Bond, HMSO, 1974.

[6] "Parallel Processing and Simulation."
P.M. Woodward, MOD unpublished work, 1974.

[7] "An Introduction to the FLEX Computer System."
RSRE report 79016, J.M. Foster, C.I. Moir, I.F. Currie, J.A. McDermid, P.W. Edwards, J.D. Morison, C.H. Pygott, 1979.

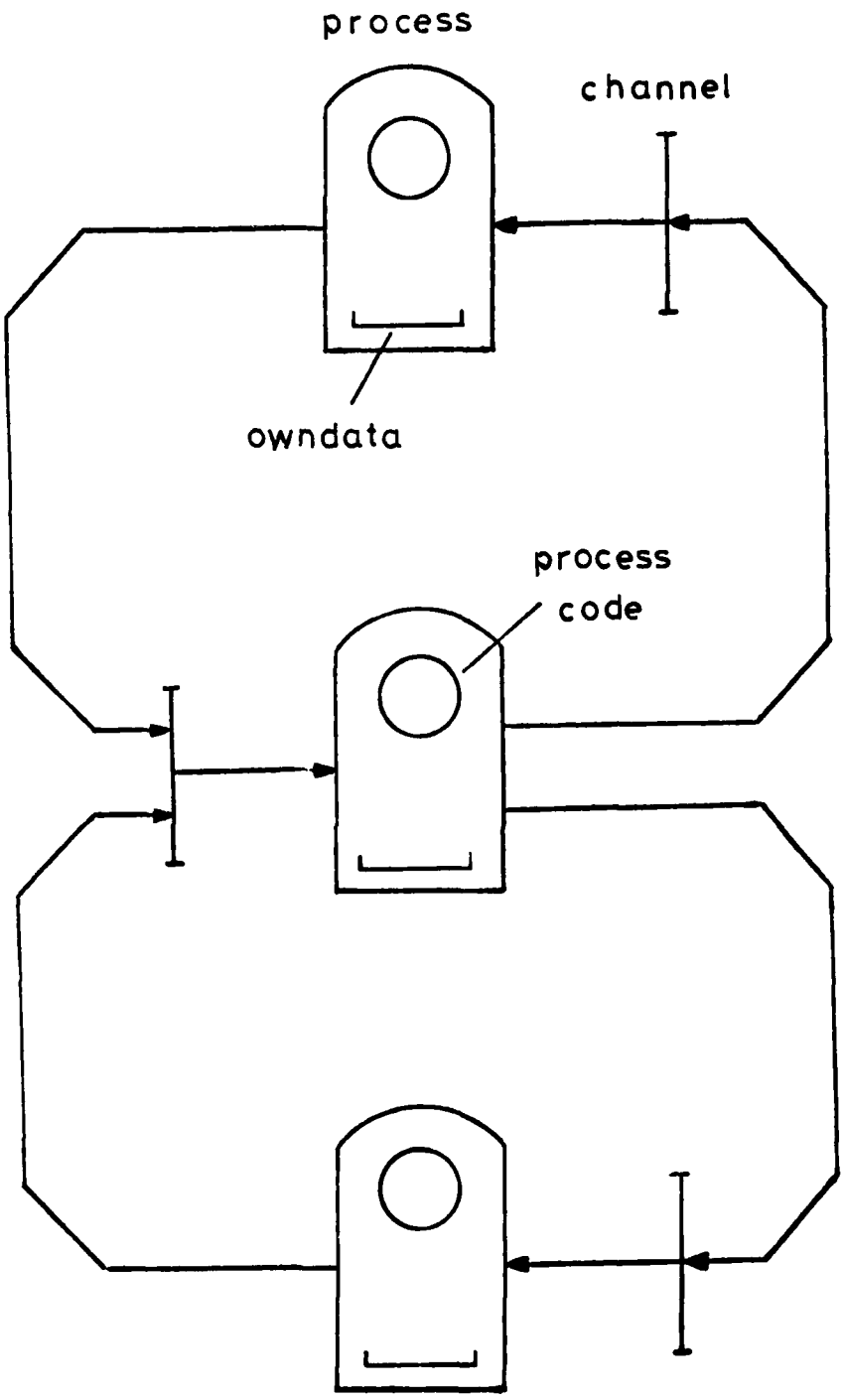


Figure 1.1

A Simple POSER Diagram.

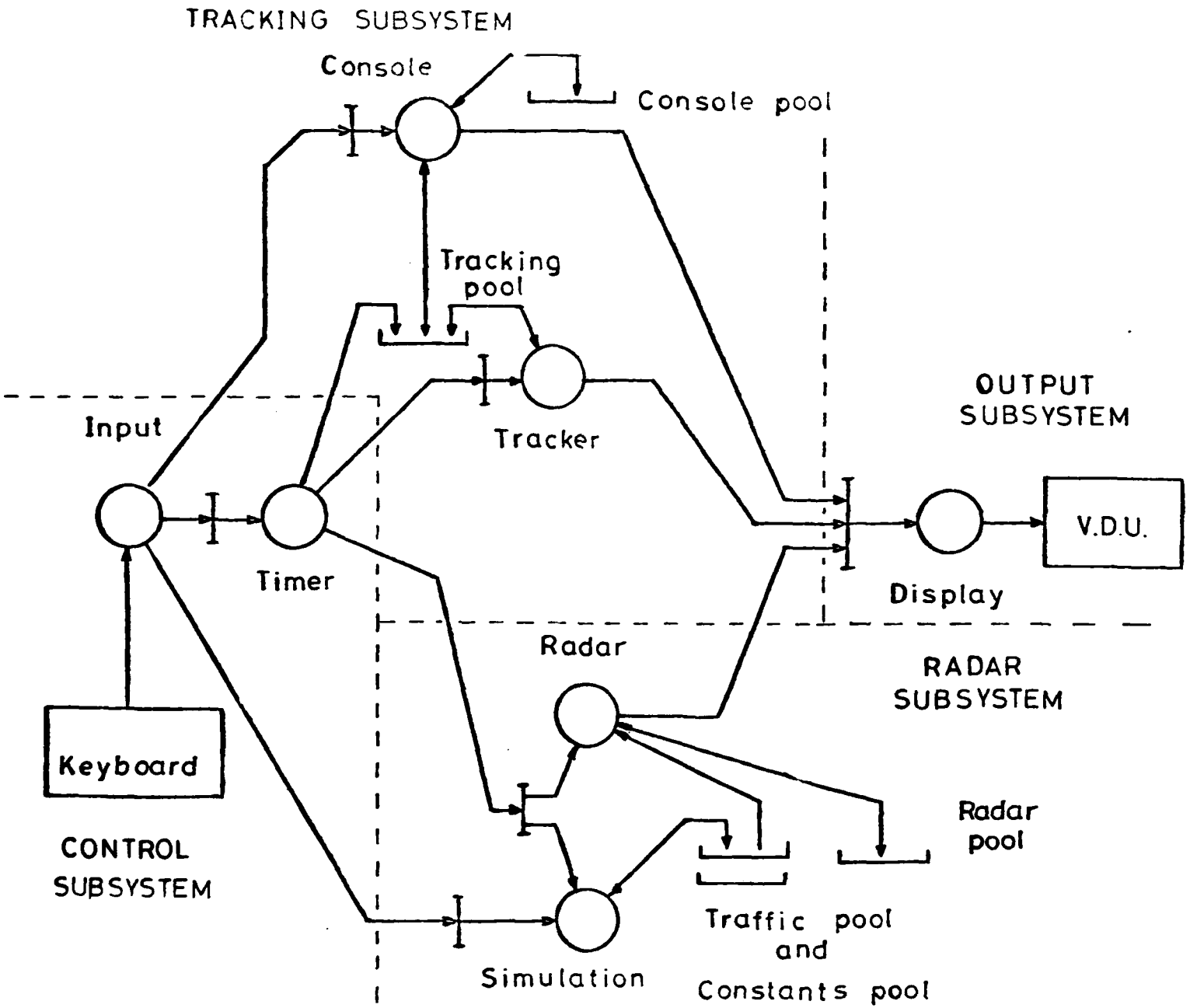


Figure 4.1 Test Program: MASCOT.

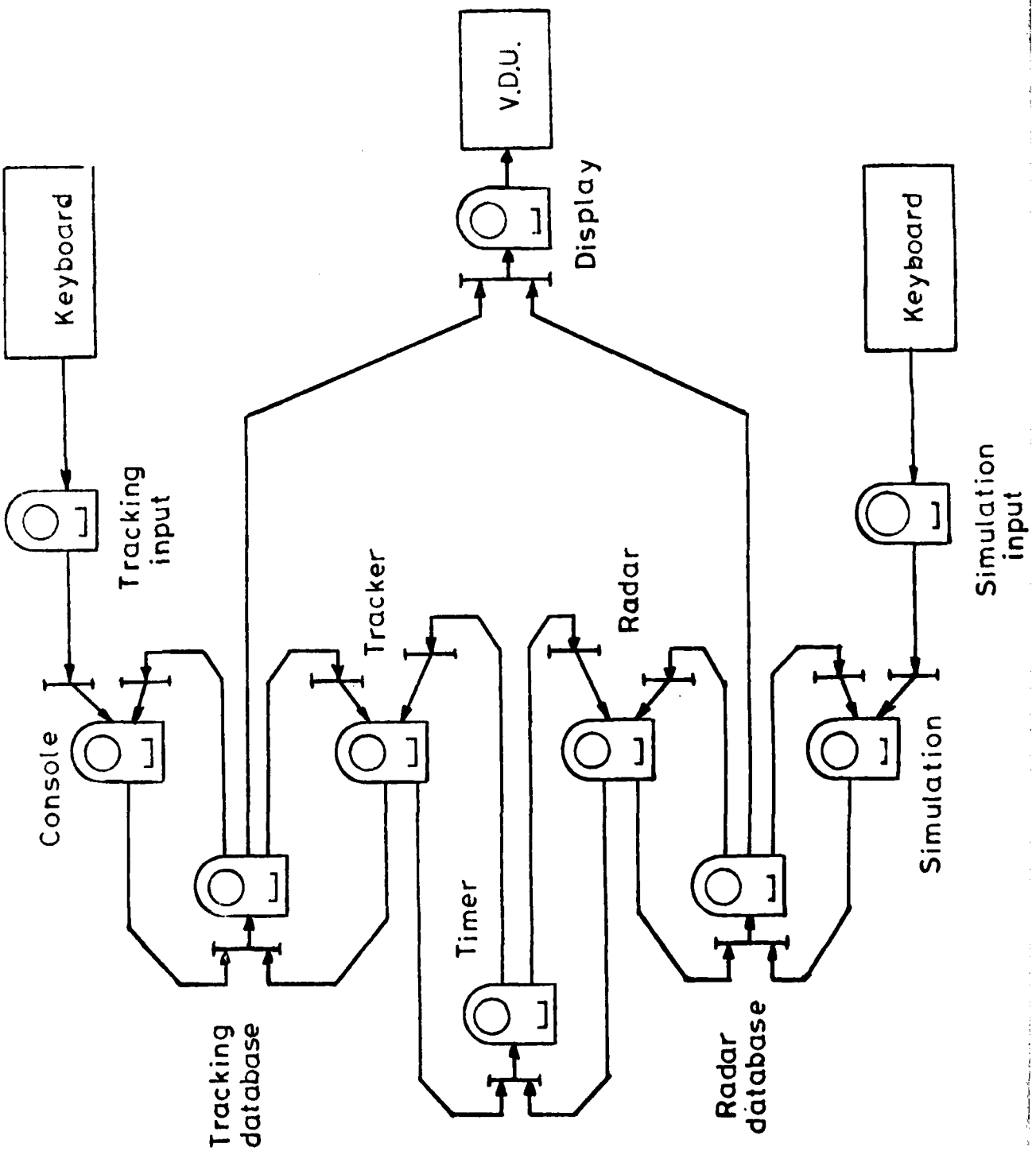


Figure 4.2

Test Program: POSER.

APPENDIX.

The following program is a complete process and channel system. To keep the example simple, only one process and one channel have been used with the process sending messages to itself via the channel. The program is assumed to be in a free running state having been started by some mechanism such as an external or software stimulus.

It must be pointed out that the example is taken from the current state of the simulation and the general format is therefore subject to change.

```
BEGIN
   PROC process shell = ([]REF PARAM input, output,
                         INT process number,
                         REF MYPROCESS this process):
      BEGIN
         COMMENT  owndata          COMMENT

         INT count := 0;

         COMMENT  process code     COMMENT

         PROC process = (REF PARAM in)PARAM:
            (PARAM output := empty param;
             print ((data OF in, newline));
             count PLUS 1;
             data OF output := (process number, count);
             full OF in := FALSE;
             full OF output := TRUE;
             output
            );

         COMMENT   scheduling condition  COMMENT

         runnable OF this process := BOOL:
            (full OF input[1]   AND
             NOT full OF output[1]
            );

         initialisation complete;

         DO
         BEGIN   output[1] := process(input[1]);
                 wait
         END

      END;
```

```
PROC channel = ([]REF PARAM input, output,
                INT channel number,
                REF MYPROCESS this  process):

   BEGIN
      PARAM buffer := empty param;
      BOOL buffer full := FALSE;

      PROC in access = (REF PARAM in):
         (buffer := in;
          full OF in := FALSE;
          buffer full := TRUE
         );

      PROC out access = PARAM:
         (buffer full := FALSE;
          buffer
         );

      runnable OF this process := BOOL:
         (IF buffer full
             THEN NOT full OF output[1]
             ELSE full OF input[1]
          FI
         );

      initialisation complete;

      DO
          BEGIN
             IF buffer full
                THEN output[1] := outaccess
                ELSE in access(input[1])
             FI;
             wait
          END

   END;

   COMMENT simulation set up  COMMENT

   INT computer 1 =1, computer 2 = 2;
   set up computers(2);
   set up process(computer 1,
                  INTPAIR VAL (1,1),
                  INTPAIR VAL (1,1),
                  process shell);
   set up channel(computer 1,
                  INTPAIR VAL (1,1),
                  INTPAIR VAL (1,1),
                  channel);
   end set up

END
```

Some of the items used in the program need explanation.

1) The mode PARAM is the input /output parameter to the shells. It contains BOOL full and INTPAIR data. In a general case, the data mode will be able to handle a wider class of modes. Empty param is a PARAM with full = FALSE.

2) MYPROCESS is the kernel process descriptor. The user has access to PROC BOOL runnable, the use of which is described in (2.4).

3) Initialisation complete is a procedure called after the process code, owndata and PROC runnable have been declared. Control is returned to the simulation and another process shell is initialised.

4) Wait is a procedure which returns control to the scheduler. It is called after each process activation.

5) The process shell is set up in the simulation by a call of "set up process" and the channel by "set up channel". These procedures take as parameters the following:

   a) The number of the computer on which the shell is to be loaded.

   b) A row of INTPAIR each describing one input to the shell. The first number is the number of the shell providing the input, and the second the output parameter space number. The processes and channels are numbered separately. Thus the process input above comes from channel 1, output 1.

   c) Another row of INTPAIR describing shell outputs as in (b).

   d) The name of the shell procedure to be loaded.

6) End set up is a procedure which sets up the kernels on the computers and starts the simulation.