Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

USING DISCRETE-EVENT COMPUTER SIMULATION TO TEST CONTROL SYSTEMS

Trevor I. Miles
Hawker Siddeley
Factory Automation Systems
Avon House PO Box 46
Chippenham, Wilts SN15-1JH
England

ABSTRACT

The material handling control system of an automated manufacturing facility is tested during the commissioning of the plant. This leads to protracted commissioning periods as well as the possibility of costly equipment damage. The testing is performed in a modular fashion so that complex interaction are often overlooked.

It is suggested that simulation can play a valuable part in speeding up the commissioning stage of plant startup by interfacing the control system to a computer simulation of the factory. The control system receives information about the state of the manufacturing system from the simulation and the simulation receives information from the controller on what to do next. By developing a complete simulation model and running the simulation for long periods of simulated time, the control system can be tested rigorously prior to the plant startup.

The emphasis of this paper is on some of the important practical concerns of linking a discrete-event simulation system to a controller. The impact this has on the simulation model will be discussed in conjunction with two small example problems. The examples will include some of the code used to connect the controller and the simulation.

1. INTRODUCTION

Automation has brought with it the curse of controlling the automated systems in an "optimal" and cost effective manner. Much has been written about the optimal configuration of computer integrated manufacturing (CIM) environments as well as the control strategies for CIM systems. However, each CIM system is different and many of the complex interactions exhibited by the system are impossible to predict. Consequently the control system and control logic, at the time of initial startup of the plant, is incomplete and, very likely, faulty. This can lead to protracted startup periods and equipment loss while the control system is being commissioned.

Simulation is being used increasingly in the configuration and planning stage of CIM systems. The simulation is used to determine required capacities for such items as pallets, automatically guides vehicles (AGVs), and machining centers. The simulation is also used to test different control strategies in an effort to improve the CIM systems performance. Consequently a detailed simulation of the CIM system often exists before the plant is commissioned.

By combining the development of the control system and the simulation models, much of the effort of the commissioning can be shifted to the development stage of the project. Furthermore, by the time the plant is built, the control system will be complete and fully tested. The commissioning period will consist therefore of testing only the mechanical function of the control system rather than a combined mechanical and logical function test.

2. LEVELS OF CONTROL

Many authors have suggested that there are several layers of control in a CIM environment. At the highest level, the level with least engineering detail, is the financial control. At the lowest level, the level with most engineering detail, is the tool path control, such as robotic arm movement or cutting tool movements. The number of levels of control and categorization of the levels of control between these two extremes is a matter of some debate.

Clearly, however, somewhere between these two extremes is the control at the level of material movement, which includes all the material handling equipment. This level of control includes such details as the setup of machining centers and grip changes on a robotic arm. The type of material handling equipment and the configuration of the material handling system determines whether the material handling equipment is controlled at the cell level or whether a system wide approach needs to be used. The degree of localization of the control system has serious implementation consequences for both the real system and the simulation model. It is sufficient to state that any level of detail of control

above and including the material movement and material handling level is easily interfaced to discrete-event simulation languages.

Note that the material movement and material handling control level deals with the interaction of various areas and components of the system. Practical reasons motivate the choice of the material movement and material handling control level as the lowest control level of interest. The most important practical reason is that discrete-event simulation models are seldom developed to the detail of cutting tool path control. The development of a model of such detail in the discrete-event simulation environment is both impractical and imprudent.

This paper deals specifically with cell level control, simply to reduce the size of the examples.

3. INTERFACING ISSUES - THE SIMULATION LANGUAGE

The SIMAN simulation language was used to model the examples in this paper. SIMAN allows the user to call user written FORTRAN or C subroutines from within the block structure of the language, and also to schedule events, create entities, etc. from within the user written subroutines. Subroutines can be written which will communicate with peripheral devices to obtain data from a bank of switches, say, which reflect the state of the system of interest. The concepts explained in this section can, however, be implemented in most of the other leading simulation languages such as SLAM II and GPSS.

Recall that the purpose of this study is to interface a discrete-event simulation language with a controller to test the control logic for material movement and material handling. The consequences of this statement are that all decisions concerning the dispatching of the material handling equipment is performed by the controller, while the simulation language issues requests for the material handling equipment and acts on the response from the controller. In fact, any control function for the material handling system is performed by the control system, while the simulation merely reports on the status of the system being simulated to the controller and acts upon commands issued by the controller.

An example of the change in the SIMAN code is given in Figure 1 for an AGV system with a merge point which sees heavy traffic. Only one AGV is allowed in the merge point at a time. The decision as modelled in the SIMAN code is to allow an AGV to proceed from the incoming track having most AGV's waiting. In the case of a tie, the first track in the list is given priority.

```
            station,1-3;
            branch,1:
                    if,m.eq.1,first:
                    if,m.eq.2,second:
                    if,m.eq.3,third;
;
first       queue,m+10:detach;
second      queue,m+10:detach;
third       queue,m+10:detach;
;
            Qpick,lnq:first,second,third;
            seize:merge;
            transport:agv(a(1)),4;
;
            station,4;
            release:merge;
```

Figure 1: SIMAN code including the merge point control code.

The equivalent SIMAN code when connecting to a controller is given in Figure 2. The first EVENT block is used to trigger a switch to indicate the arrival of an AGV from a particular track to the merge point. The second EVENT block is used to trigger a switch to indicate the departure of an AGV from the merge point. The AGVs wait in the QUEUE blocks until the appropriate signal is sent from the user written subroutine. Notice that in this case the decision logic has not been specified. The decision is made entirely by the controller and the SIMAN model merely acts on the decision reached. Different control strategies can be tested, therefore, without modification of the SIMAN code.

```
  Station,1-3;
  event,m+100;  Send a signal 101-103 to controller
  queue,m+10;
  wait:m+10;  Wait for signal 11-13 from controller
  transport:agv(a(1)),4;
;
  station,4;
  event:m+100;  send signal 104 to controller
```

Figure 2: SIMAN code when interfaced to a controller.

In conventional discrete-event simulation, the simulation clock is updated immediately to the event time of the next event on the calendar. However, when interfacing to a controller, the simulation clock must progress smoothly so that timing considerations between the controller and the real system can be evaluated accurately. The smooth simulation clock was achieved by comparing the simulation clock plus a reference astronomical time with the current astronomical time whenever a signal is sent to or

received from the controller. If the simulation clock plus the reference astronomical time is greater than the current astronomical time, delay until they are equal. If the simulation clock plus the reference astronomical time is greater than the current astronomical time proceed. An example of how to achieve the smooth simulation clock is given in Figure 3. The routine ITIME, which is used to return the astronomical time, is system dependent. An example is given in Assembler for an IBM PC compatible in Figure 4.

```
        subroutine prime
        real reftim
        common /time/ reftim
c
C *** store the astronomical time at the start of
the simulation.
C
        call itime( ihr, imin, isec, ihun )
        reftim = 60.*float( ihr ) + float( imin )
     1        + float( isec )/60. +
     1          float( ihun )/6000.
C
        return
        end
c
c
c
        subroutine event(ptr,n)
        integer ptr
        real reftim, curtim
        common /time/ reftim
        common/sim/s(50),sl(50),d(50),dl(50),
     1          x(50),tnow,tlast,tfin,j,nrun
c
        data offset/ 0.0 /
c
C *** Loop until the simulation clock
C     and astronomical clock are synchronized.
C     The simulation clock unit is minutes.
c
   10 continue
        call itime( ihr, imin, isec, ihun )
        curtim = 60.*float( ihr ) + float( imin )
     1        + float( isec )/60. +
     1        float( ihun )/6000. + offset
        if( curtim .lt. reftim ) then
        offset = offset + 24.*60.
        curtim = curtim + 24.*60.
        endif
        if( reftim + tnow .gt. curtim ) go to 10
c
```

Figure 3: The FORTRAN code to synchronize the clocks.

```
;
;   SUBROUTINE ITIME(HOUR,MINUTE,SECOND,HUNDREDTH)
;
;
FRAME      STRUC ;
SAVEBP     DW ? ;
SAVERET    DD ? ;
HUND       DD ? ; Address of the Second/100  (OUT)
SEC        DD ? ; Address of the Seconds     (OUT)
MIN        DD ? ; Address of the Minutes     (OUT)
HOUR       DD ? ; Address of the Hour        (OUT)
FRAME      ENDS
;
_DATA         SEGMENT   WORD PUBLIC 'DATA'
_DATA         ENDS
DGROUP        GROUP     _DATA
;
CSEG       SEGMENT 'CODE'
           ASSUME CS:CSEG,DS:DGROUP,SS:DGROUP
;
ITIME      PROC  FAR
           PUBLIC ITIME
           PUSH BP
           MOV BP,SP
;
;  Determine the time
;
           MOV AH,2CH
           INT 21H
;
           MOV AH,0H
           MOV AL,CH
           LES BX,[BP]+HOUR
           MOV ES:[BX],AX
;
           MOV AL,CL
           LES BX,[BP]+MIN
           MOV ES:[BX],AX
;
           MOV AL,DH
           LES BX,[BP]+SEC
           MOV ES:[BX],AX
;
           MOV AL,DL
           LES BX,[BP]+HUND
           MOV ES:[BX],AX
;
DONE:      POP BP
           RET 16
ITIME      ENDP
CSEG       ENDS
END
```

Figure 4: The Assembler routine to return astronomical time on an IBM PC compatible.

Obviously the code has been written for a simulation clock which progresses in minutes. Little effort is required to change the synchronization code for

other simulation clock time references. The test for CURTIM .LT. REFTIM is used to correct for the case when the simulation is performed beyond midnight. The routine ITIME returns 23 for the hours for any time between 11:00pm and midnight and 0 for any time between midnight and 1:00am.

4. INTERFACING ISSUES - THE HARDWARE AND SOFTWARE

The simulation needs to be able to detect when the controller sends messages to the simulation, and to report a change in system status to the controller. The manner in which this information is captured depends very much on the type of information being sent and the communications hardware and software available. Two systems will be discussed, namely direct connection to switches and RS232 communication.

Regardless of the type of communication, a crucial aspect of the communication process is the cycle time between when the simulation reports a change in the system status to the controller and when the controller can send a response back to the simulation. The cycle time is comprised of two components: The first is the actual time required for message passing, while the second is the time required for the controller to act on the new information and to formulate the correct response. Either of these times could be negligible, but it is unlikely that both will be negligible. When using communication software, the message passing time can be replaced be a method of message acknowledgement. When the simulation sends a message to the controller, the controller has to respond with a message to acknowledge the receipt of the message. The examples given below for specific communication systems include the necessary code for the cycle time considerations.

For banks of switches, the only available method is to poll the state of the switches at regular intervals and compare the current state with a predefined action state. The user written code will have imbedded within it the necessary actions to be performed depending on the state of the switches.

An example of the code required for state comparison is given in Figure 5. The example refers to the AGV track merge point problem where the track number along which a waiting AGV may proceed is binary encoded on switches 100 and 101. The subroutine READSW, which is used to return the value of a relay from the controller, is hardware dependent. The example given in Figure 6 is for a MetraByte MDB-64 board in an IBM PC compatible. An equivalent routine, SETSW, which is given in Figure 7, is used to set the value of a relay which will be read by the controller. The MDB-64 board is used to connect to the MetraBUS, which can support up to 64 MetraByte MDI-16 digital I/O boards. The assembler routines INP and OUTP, given in Figures 8 and 9 respectively, are used to read from and write to the Metrabyte MDB-64 board. The controller used was a GE Series III PLC which was connected to the Metrabyte MDI-16 boards via 110V optically isolated relays.

```
      subroutine event( n, ptr )
      implicit integer(a-z)
      logical one, two
C
.
.
c
C *** event 10 is used to poll the controller to
c     determine which agv may proceed.  The event
c     is rescheduled to occur in the next 0.1 time
c     units
C
      if( n .eq. 10 ) then
            call readsw( 10, one )
            call readsw( 11, two )
            call sched( ptr, n, 0.1 )
c
            if( one .and .not.two ) then
                  call signal( 11 )
c
            elseif( .not.one .and. two ) then
                  call signal( 12 )
c
            elseif( one .and. two ) then
                  call signal( 13 )
            endif
c
      elseif( 101 .le. n .and. n .le. 104 ) then
            call setsw( n, .true. )
c
      endif
.
.
```

Figure 5: The FORTRAN code to compare switch states.

Communication standards such as RS232 or ethernet can be used at two levels. The first is to communicate between the controller and devices, such as a robot or a CNC/DNC machine. The second is to communicate between two computers, one of which could be connected to a data acquisition system or could be a controller at a lower/higher level in the control hierarchy. The type of communication of interest in this paper is that between the controller and devices.

```
      subroutine readsw( switch, state )
      implicit integer(a-z)
      integer*4 it1, it2
      logical  state
c
      integer*4       lstwrt, wrttim, cyctim
      common /bustim/ lstwrt, wrttim, cyctim
c
      common /switch/ sw(8,20)
      common /baddr/  base
c
C ** extract the port and bit from
c    the switch number
c         switch = ppb
C
      port  = switch / 10
      bit   = switch - 10*port
c
c ** check that the required time has elapsed to
c    complete a cycle of the PLC since the last
c    write to the PLC
c
   10   call itime( ih, im, is, in )
        it1 = 360000*ih + 6000*im + 100*is + in
        if( lstwrt + cyctim .gt. it1 ) go to 10
C
C ** activate the required port
C
      call outp( base+1, port )
      call itime( ih, im, is, in )
      it1 = 360000*ih + 6000*im + 100*is + in
C
C ** read the new value from the port
C
   20   call itime( ih, im, is, in )
        it2 = 360000*ih + 6000*im + 100*is + in
        if( it1+wrttim .lt. it2) go to 20
        value = inp( base+0 )
c
C ** Extract the required bit from the 8 bit value
c    and store the new value
C
      do 30 i = 1, bit+1
      if( i .eq. bit+1 ) then
            if( mod( value, 2 ) .eq. 0 ) then
                  sw(i,port+1) = .false.
            else
                  SW(I,PORT+1) = .TRUE.
            ENDIF
      endif
      value = value / 2
   30 continue
      state = sw(bit+1,port+1)
c
      return
      end
```

Figure 6: The FORTRAN code to read a switch state.

```
      subroutine setsw( switch, state )
      implicit integer(a-z)
      logical  state
      integer*4 it1, it2
c
      integer*4       lstwrt, wrttim
      common /bustim/ lstwrt, wrttim
c
      common /switch/ sw(8,20)
      common /baddr/  base
c
C ** extract the port and bit from the switch
c    number -  switch = ppb
C
      port  = switch / 10
      bit   = switch - 10*port
      value = 0
c
C ** change the setting of the current bit only and
c    calculate the 8 bit value from previous
c    settings
C
      do 10 i = 1, 8
      if( i .eq. bit+1 ) then
            if( .not. state ) then
                  sw(i,port+1) = .false.
            else
                  SW(I,PORT+1) = .TRUE.
            ENDIF
      endif
      if(sw(i,port+1)) value = value + 2**(i-1)
   10 continue
c
C ** activate the required port
C
      call outp( base+1, port )
      call itime( ih, im, is, in )
      it1 = 360000*ih + 6000*im + 100*is + in
C
C ** write the new value to the port
C
   20 call itime( ih, im, is, in )
      it2 = 360000*ih + 6000*im + 100*is + in
      if( it1+wrttim .lt. it2) go to 20
      call outp( base+0, value )
C
C ** record the time of the last write to
c    the metrabus
C
      call itime( ih, im, is, in )
      lstwrt = 360000*ih + 6000*im + 100*is + in
c
      return
      end
```

Figure 7: The FORTRAN code to set a switch state.

```
;
;       Value = inp( port )
;
;       inputs a byte from the specified port
;
;
frame           struc;
save_bp dw ? ;
save_ds dw ? ;
save_retdd ? ;
port_   dd ? ;          Address of the port
frame   ends
;
tools_  segment 'code'
        assume          cs:tools_
;
inp     proc            far
        public          inP
        PUSH            DS
        PUSH            BP
        MOV             BP, SP
;
        LDS             BX, [BP]+PORT_
        MOV             DX, DS:[BX]
;
        IN              AL, DX
        sub             ah, ah
;
return: mov             sp, bp
        POP             BP
        POP             DS
        RET             8
;
INP     ENDP
TOOLS_  ENDS
;
END
```

Figure 8: The Assembler routine to read from a port.

The RS232 system which will be discussed below is one in which the controller sends encoded messages to the devices to perform specific actions. The message content is specific to the device. When the device has completed the required action, an encoded message is sent back to the controller.

An efficient method of reducing the CPU requirements for the communication is to use separate communication processes which run in "background". That is to say that the communication processes remain idle until they receive a specific interrupt from a communications device, such as the RS232 board. Buffer areas which are common to both the communication processes and the simulation are used to store the messages until such time as they can be retrieved or transmitted. When the communication process receives an interrupt from the communications device, the messages is placed in the input buffer. When a message is placed in the output buffer, an interrupt is generated and the communications process transmits the contents of the output buffer.

```
;
;       OUTP( port, BYTE )
;
;       OUTPuts a byte TO the specified port
;
;
frame           struc;
save_bp dw ? ;
save_ds dw ? ;
save_retdd ? ;
BYTE_   DD ? ;  ADDRESS OF THE BYTE TO OUTPUT
port_   dd ? ;  Address of the port
frame   ends
;
tools_  segment 'code'
        assume          cs:tools_
;
OUTP    proc            far
        public          OUTP
        PUSH            DS
        PUSH            BP
        MOV             BP, SP
;
        LDS             BX, [BP]+PORT_
        MOV             DX, DS:[BX]
;
        LDS             BX, [BP]+byte_
        MOV             al, DS:[BX]
;
        OUT             DX, AL
;
return: mov             sp, bp
        POP             BP
        POP             DS
        RET             8
;
OUTP    ENDP
TOOLS_  ENDS
;
END
```

Figure 9: The Assembler routine to write to a port.

The efficient utilization of the CPU is especially important for the computer performing the simulation because the simulation is very CPU intensive. The simulation needs to query the input buffer at regular intervals and act upon any messages which may be waiting. The information sent by the controller is an encoded message stating explicitly the action to be performed. Consequently no comparison need be made with some previous state

to determine if any change has taken place. To send a message to the controller, the simulation merely writes the message to the output buffer. This triggers an interrupt for the communications process which then takes care of protocol and/or hardware concerns of send the message to the controller.

An example of the code required for RS232 communication is given in Figure 10. The code is for the AGV problem again, but now the track number is returned as a specific value. The subroutine GETMSG is not supplied because it is dependent on the communication hardware and software available. The code was fully tested using the C Asynch Manager from Blaise Computing Inc. The controller in this case was another PC, also running the C Asynch Manager. The controller PC would pick up the state messages from the simulation PC and send the required action messages to the simulation PC.

```
        subroutine event( n, ptr )
        implicit integer(a-z)
        logical inbuf
C
.
.
.
c
C *** event 10 is used to poll the controller to
c     determine which agv may proceed.  The event
c     is rescheduled to occur in the next 0.1 time
c     units
C
        if( n .eq. 10 ) then
              call sched( ptr, n, 0.1 )
C
c *** check whether any signals are waiting
c     in the input buffer
C
              if( .not. inbuf() ) return
C
C *** retreive the track number from the
c     message queue
C
              track = getmsg()
              call signal( track )
c
C *** send signal to controller that an
c     agv has arrived
C
        elseif( 101 .le. n .and. n .le. 104 ) then
              call sndmsg( n )
c
        endif
```

Figure 10: The FORTRAN code to receive the path number via RS232 communications.

5. Test Problem - AGV Track Merge Point

Three different flexible manufacturing systems use the same automatic storage and retrieval system (AS/RS) for raw material, finished product, work-in-progress (WIP), and tool and pallet storage. Automatically guided Vehicles (AGVs) are used to transport the material to and from the AS/RS. The AS/RS has a single request area and a single pickup/dropoff area. A single track is available for the AGVs to issue requests and to pick up or drop off their material. The details of the AS/RS are not simulated.

Obviously this is a very traffic intensive area which needs to be controlled with precision. If the traffic intensity of this area is very high, the control of this area will have a large impact on the overall performance of the system.

```
begin,,,,AGV,y;
        create : ex(1,1) : mark(1);
        assign : ns = dp(2,2);   Assign the FMS
        assign : m  = ns;
        assign : is = ed(m);   Assign request type
;
        branch,1:
             if, ns.eq.1, first:
             if, ns.eq.2, second:
             if, ns.eq.3, third;
;
first   queue,1 : detach;   Wait until the request
second  queue,2 : detach;   area is available
third   queue,3 : detach;
;
        qpick, LNQ : first : second : third;
        seize : request;     Assign track with
;                            most requests
;
        assign : m = ns+3;
        tally : m, int(1);   Collect Q statistics
;
        delay  : tr(6,6);    Delay by request time
        queue,4;             Wait until
;                            pickup/dropoff area
        seize : IO;          is available before
        release : request;   releasing request
;                            area
        assign : m = 3 + 10*(ns-1) + is;
;                            Assign distribution
        delay : ed(m);       number and delay by
;                            pickup/dropoff time
        release : IO;;
        assign : m = ns;
        tally : m, int(1) :
                dispose;  Collect TIS statistics
;
end;
```

Figure 11: The AGV Merge Point example model.

```
begin,,,,n;
project,AGV Example,TIM,5/21/1989;
discrete,500,1,4;
;
resources : 1,request:
            2,IO;
;
tallies : 1,TiS FMS 1:
          2,TiS FMS 2:
          3,TiS FMS 3:
          4,TiQ FMS 1:
          5,TiQ FMS 2:
          6,TiQ FMS 3;
;
dstats : 1, nq(1), Requests Q1:
         2, nq(2), Requests Q2:
         3, nq(3), Requests Q3;
;
parameters :
   1, 1 :                  !Time between arrivals
   2, .2,1, .7,2, 1.,3 :   !FMS #
   3, .50,1,    !FMS 1     !Raw material pickup
      .70,2,               !Finished product dropoff
      .76,3,               !WIP pickup
      .82,4,               !WIP dropoff
      .89,5,               !Tool pickup
      .94,6,               !Tool dropoff
      .97,7,               !Pallet pickup
     1.0,8:                !Pallet dropoff
   4, .50,1,    !FMS 2     !Raw material pickup
      .70,2,               !Finished product dropoff
      .76,3,               !WIP pickup
      .82,4,               !WIP dropoff
      .89,5,               !Tool pickup
      .94,6,               !Tool dropoff
      .97,7,               !Pallet pickup
     1.0,8:                !Pallet dropoff
   5, .50,1,    !FMS 3     !Raw material pickup
      .70,2,               !Finished product dropoff
      .76,3,               !WIP pickup
      .82,4,               !WIP dropoff
      .89,5,               !Tool pickup
      .94,6,               !Tool dropoff
      .97,7,               !Pallet pickup
     1.0,8:                !Pallet dropoff
   6, .04,.05,.06:         !Request time
   7, .45,.65,2.5: !FMS 1  !Raw material pickup
   8, .2,.4,.6:            !Finished product dropoff
   9, .45,.65,2.5:         !WIP pickup
  10, .2,.4,.6:            !WIP dropoff
  11, .45,.65,2.5:         !Tool pickup
  12, .2,.4,.6:            !Tool dropoff
  13, .45,.65,2.5:         !Pallet pickup
  14, .2,.4,.6:            !Pallet dropoff
  15, 0.:
  16, 0.:
  17, .45,.65,2.5: !FMS 2  !Raw material pickup
  18, .2,.4,.6:            !Finished product dropoff
  19, .45,.65,2.5:         !WIP pickup
  20, .2,.4,.6:            !WIP dropoff
  21, .45,.65,2.5:         !Tool pickup
  22, .2,.4,.6:            !Tool dropoff
  23, .45,.65,2.5:         !Pallet pickup
  24, .2,.4,.6:            !Pallet dropoff
  25, 0.:
  26, 0.:
  27, .45,.65,2.5: !FMS 3  !Raw material pickup
  28, .2,.4,.6:            !Finished product dropoff
  29, .45,.65,2.5:         !WIP pickup
  30, .2,.4,.6:            !WIP dropoff
  31, .45,.65,2.5:         !Tool pickup
  32, .2,.4,.6:            !Tool dropoff
  33, .45,.65,2.5:         !Pallet pickup
  34, .2,.4,.6:            !Pallet dropoff
  35, 0.:
  36, 0.;
;
distributions:
    1, dp(3,3) :    !FMS 1 request type
    2, dp(4,3) :    !FMS 2 request type
    3, dp(5,3) :    !FMS 3 request type
    4, tr( 7,4) :   !FMS 1 !Raw material pickup
    5, tr( 8,4) :     !Finished product dropoff
    6, tr( 9,4) :         !WIP pickup
    7, tr(10,4) :         !WIP dropoff
    8, tr(11,4) :         !Tool pickup
    9, tr(12,4) :         !Tool dropoff
   10, tr(13,4) :         !Pallet pickup
   11, tr(14,4) :         !Pallet dropoff
   12, tr(15,4) :
   13, tr(16,4) :
   14, tr(17,4) : !FMS 2 !Raw material pickup
   15, tr(18,4) :    !Finished product dropoff
   16, tr(19,4) :         !WIP pickup
   17, tr(20,4) :         !WIP dropoff
   18, tr(21,4) :         !Tool pickup
   19, tr(22,4) :         !Tool dropoff
   20, tr(23,4) :         !Pallet pickup
   21, tr(24,4) :         !Pallet dropoff
   22, tr(25,4) :
   23, tr(26,4) :
   24, tr(27,4) : !FMS 3 !Raw material pickup
   25, tr(28,4) :    !Finished product dropoff
   26, tr(29,4) :         !WIP pickup
   27, tr(30,4) :         !WIP dropoff
   28, tr(31,4) :         !Tool pickup
   29, tr(32,4) :         !Tool dropoff
   30, tr(33,4) :         !Pallet pickup
   31, tr(34,4) :         !Pallet dropoff
   32, tr(35,4) :
   33, tr(36,4) ;
;
replicate,1,0,50000;
;
end;
```

Figure 12: The AGV Merge Point example experiment.

The AGVs are assumed to arrive by a Poisson process with the rate of arrival dependent on the FMS from which they emanate and the type of request, viz. tool and pallet dropoff and pickup, raw material pickup, finished product dropoff, or WIP pickup and dropoff. The AGVs are all assumed to be of the same type and to have the same travel speed. Both the time to accept a request the time to drop off items are assumed to follow triangular distributions which approximate normal distributions. (There is a dropoff buffer which is used to store the material until it can be stored in the AS/RS.) The time to pick up items is assumed to follow a triangular distribution which approximates a low order Erlang distribution.

The SIMAN MODEL and EXPMT files are given in Figures 11 and 12 respectively. Note that this example is for the case where the control logic is imbedded in the model file.

6. Test Problem - Robotic Cell Scheduling

The material handling requirements of three different machining centers are satisfied by a single robot. The raw material arrives on a conveyor and the finished product leaves on a separate conveyor. The input conveyor is stopped when a pallet reaches the end. The conveyor remains stopped until the robot removes the conveyor. Each Machining center has one output buffer and one input buffers. Four part types arrive to the system and each part type has a different sequence of operations. The machining centers' tool magazines contain sufficient tools to machine all required parts, but part programs are down-loaded from a host computer.

Obviously the critical component in the system is the order in which parts are moved by the robot. The control logic of the robot movement is coded into the model, but could be performed by a PLC. To simplify the example problem, all parts first go to machining center 1, then 2, and then 3. As coded in the model, the logic for control of the robot's arm is to move the part that is closest to completion if the input buffer to the next machining center is available. The control logic for the general case of part routings is considerably more complex than the example.

It is assumed that there is an inexhaustible supply of parts to enter the robotic cell, and that the parts are always available. The part types are assigned according to discrete probability distribution. The machining times follow a triangular distribution which approximates a normal distribution, and the part program loading time follows a uniform distribution.

The example as implemented is a trivial case of the more complex problem in that all the parts go to each machining center in sequence. In fact, all parts are identical because all the part specific PARAMETERS elements are identical. The example problem can be made more complex by simply changing the SEQUENCES element and the control logic at the end of the model file. The model and experiment files are given in Figures 13 and 14 respectively.

7. Conclusions

The use of simulation to present a controller with a wide variety of situations has been demonstrated to be both cheap and easy. While the examples were for fairly small systems, large systems should present no additional problems beyond the possibility of the simulation being too slow.
The choice of the type of interface between the controller and the simulation depends on the controller being used. The emulation of digital I/O is cheap and easy to implement, but also rigid and restricted to a low level of control. The use of RS232 and ethernet is more difficult to implement, but the simulation can be used to emulate a much wider range of control levels.

Acknowledgements

I wish to thank C. Dennis Pegden for allowing me to pursue my interests as described in this paper while employed at Systems Modelling Corp. I wish to thank Tony Vandenberge for many fruitful discussions on the topic over a cup of coffee at Systems Modeling.

Author's Biography

TREVOR MILES is a Senior Consultant with the Factory Automation Division of Hawker Siddeley in England. Miles previously worked for Systems Modeling Corporation for 5 years where he was involved in the development and maintainance of many of the Systems Modeling products.

Miles received his MSc in Engineering from the University of the Witwatersrand, Johannesburg, South Africa and his BSc in Chemical Engineering for the University of Cape Town, Cape Town, South Africa. He is still trying to finish a PhD in Industrial Engineering from the Pennsylvania State University in State College, PA.

```
begin,,,,robot,y;
         create;
regen  assign : a(1) = tnow;  Loop for
;                              next creation
       assign : ns = dp(2,2); assign the
;                              part type
       assign : is = 0;  Reset sequence pointer
;
       route : 0,seq;    send to arrival station
;
       station,7;
       queue,m;          wait for the in conveyor
       access : in;
       convey : in,seq;    convey to load region
;
       station,5;
       branch,2:
         always, enter:    !Enter primary entity
         always, regen;    !Create a new arrival
enter  signal : 1; notify controller of arrival
       queue,m+10;     wait for available robot
       wait : m+10;
       queue,m+20;
       request : robot;     get access to robot
       queue, m+30;
       seize : inbuf(m-4); access input buffer
       exit : in;           get off the conveyor
       transport : robot, seq;     send to first
;                                  machining center
       station,4;
       queue,m;       wait for the out conveyor
       access : out;
       release : inbuf(m);   resource available
       free : robot;
       signal : 1;     signal robot availability
       convey : out,seq;    send to exit station
;
       station,6;
       exit : out;          get off out conveyor
       assign : x(1) = ns; collect statistics
       tally : x(1), int(1) : dispose;
;
       station, 1-3;
       free : robot;     release robot
       signal : 1;     signal robot availability
       queue,m;
       seize : machine(m); get machining center
       release : inbuf(m); free input buffer
;
       delay : ed(a(2));  load the part program
       delay : ed(a(3));  do the machining
;
       queue,m+40; wait until the output buffer
       seize : outbuf(m); is available
       release : machine(m); free up the
;                          machining center
       signal : 1;    send signal to controller
;
       queue,m+10;
       wait : m+10;  wait until the robot is
       queue, m+20;    available before accessing
        request : robot;
;
       assign : a(2) = m; store current station
          assign : m    = a(4); set current
                          station to next station
       queue, m+30;      get access to the next
       seize : inbuf(m);    input buffer
       assign : m = a(2);  reset station number
       release : outbuf(m);  free output buffer
       transport : robot,seq;  send to the next
;                              machining center
       create;                 create controller
;
; cascade down the list until a part waiting is
; found for which the next input buffer is
; available.  Always move the part closest to
; completion first.
;
control branch,1:
        if, nq(13).gt.0.and.nr(4).eq.0, sig13:
        if, nq(12).gt.0.and.nr(3).eq.0, sig12:
        if, nq(11).gt.0.and.nr(2).eq.0, sig11:
        if, nq(15).gt.0.and.nr(1).eq.0, sig15:
        else, wait;
;
; send the appropriate signal and then wait
; until the next signal gets sent to the
; controller
;
sig11   signal : 11 : next( wait );
sig12   signal : 12 : next( wait );
sig13   signal : 13 : next( wait );
sig15   signal : 15 : next( wait );
;
wait    queue,10;
        wait : 1 : next( control );
;
end;
```

Figure 13: The Robotic Cell example model.

```
begin,,,,n;
project,ROBOT Example,TIM,5/21/1989;
discrete,500,4,50,7,4;
;
resources : 1- 4, INBUF:
            5- 7, MACHINE:
            8-10, OUTBUF;
;
transporters : 1, ROBOT, 1, 1, 1.0, 5-a;
;
distances : 1, 1-5, 1, 2, 3, 1/
                      1, 2, 2/
                         1, 3/
                            4;
;
conveyors : 1, IN,  1, 1.0, 1, a:
            2, OUT, 2, 1.0, 1, a;
;
segments : 1, 7, 5-1:
           2, 4, 6-1;
;
tallies : 1,TiS PART 1:
          2,TiS PART 2:
          3,TiS PART 3;
;
dstats : 1, nt(1), Robot Util;
;
;a(2) = dist. number for part program loading
;a(3) = distribution number for machining time
;a(4) = next station that part visits
sequences :
1, 7/5/1,, 1, 2, 2/2,, 3, 4, 3/3,, 5, 6, 4/4/6:
2, 7/5/1,, 7, 8, 2/2,, 9,10, 3/3,,11,12, 4/4/6:
3, 7/5/1,,13,14, 2/2,,15,16, 3/3,,17,18, 4/4/6:
4, 7/5/1,,19,20, 2/2,,21,22, 3/3,,23,24, 4/4/6;
;
distributions:
  1, un( 3,3) :  !Part 1 Mach 1 Program Loading
  2, tr( 4,3) :  !  Machining
  3, un( 5,3) :  !  Machine 2  Program Loading
  4, tr( 6,3) :  !                  Machining
  5, un( 7,3) :  !  Machine 3  Program Loading
  6, un( 8,3) :  !                  Machining
  7, un( 9,4) :  !Part 2 Mach 1 Program Loading
  8, tr(10,4) :  !                  Machining
  9, un(11,4) :  !  Machine 2  Program Loading
 10, tr(12,4) :  !                  Machining
 11, un(13,4) :  !  Machine 3  Program Loading
 12, tr(14,4) :  !                  Machining
 13, un(15,5) :  !Part 3 Mach 1 Program Loading
 14, tr(16,5) :  !                  Machining
 15, un(17,5) :  !  Machine 2  Program Loading
 16, tr(18,5) :  !                  Machining
 17, un(19,5) :  !  Machine 3  Program Loading
 18, tr(20,5) :  !                  Machining
 19, un(21,6) :  !Part 4 Mach 1 Program Loading
 20, tr(22,6) :  !                  Machining
 21, un(23,6) :  !  Machine 2  Program Loading
 22, tr(24,6) :  !                  Machining
 23, un(25,6) :  !  Machine 3  Program Loading
 24, tr(26,6) ;  !                  Machining
;
parameters : 1, 0:
  2, .2,1, .5,2, .7,3,1.,4 :!Part #
  3, .04,.06: !Part 1 Machine 1 Program Loading
  4, .45,.65,2.5:   !                Machining
  5, .04,.06:     !   Machine 2 Program Loading
  6, .45,.65,2.5:   !                Machining
  7, .04,.06:       ! Machine 3 Program Loading
  8, .45,.65,2.5:    !               Machining
  9, .04,.06: !Part 2 Machine 1 Program Loading
 10, .45,.65,2.5:   !                Machining
 11, .04,.06:       ! Machine 2 Program Loading
 12, .45,.65,2.5:   !                Machining
 13, .04,.06:       ! Machine 3 Program Loading
 14, .45,.65,2.5:   !                Machining
 15, .04,.06: !Part 3 Machine 1 Program Loading
 16, .45,.65,2.5:   !                Machining
 17, .04,.06:       ! Machine 2 Program Loading
 18, .45,.65,2.5:   !                Machining
 19, .04,.06:       ! Machine 3 Program Loading
 20, .45,.65,2.5;   !                Machining
 21, .04,.06: !Part 4 Machine 1 Program Loading
 22, .45,.65,2.5:   !                Machining
 23, .04,.06:       ! Machine 2 Program Loading
 24, .45,.65,2.5:   !                Machining
 25, .04,.06:       ! Machine 3 Program Loading
 26, .45,.65,2.5;   !                Machining
;
replicate,1,0,50000;
;
end;
```

Figure 14: The Robotic Cell example experiment.