CODESIM: A COMPACT DISCRETE EVENT SIMULATOR

Lawrence L. Rose
Battelle
Columbus Laboratories
505 King Avenue
Columbus, Ohio 43201

CODESIM is a compact software facility\(^1\) that supports discrete event simulation modeling. Written in Fortran, the CODESIM routines enable facile model construction with a maximum of modeler independence. CODESIM requires no inputs, handles queuing without buffer movement, provides built-in trace control, and permits arbitrary user naming and typing of entity attributes. Statistical routines for data collection and reporting are provided by the CODESIM software—all outputs are in the user domain, as are additional model input parameters, etc. Thus, CODESIM represents a no-frills tool for discrete event simulation. Its advantages include a free programmer structure, speed, size, interactive use, and potential for both machine and language portability.

1. INTRODUCTION

The development of CODESIM was motivated by two major factors: interactive modeling and programmer independence. Modeling flexibility (e.g., user inputs, dynamic trace and report selection, variable modification, and stopping criteria) cannot be provided with batch systems. Yet the traditional discrete event languages such as GASP (Pritsker 1975), SLAM (Pritsker and Pegden 1979), and SIMSCRIPT (Kivist et al. 1973) were developed for the batch mode; they represent large, sophisticated software packages. Feasible interactive modeling mandates compact, efficient software; CODESIM was designed and implemented in a compact (60K) and efficient manner.

The second concern of programmer independence stems from the understanding that large scale discrete event simulations are large software packages that can best be implemented by non-neophyte programmers. As each programmer has his own style, etc. and language capabilities, the CODESIM philosophy is to build upon these rather than to impose a heavy environment with requisite data structures, naming conventions, formats, etc. The CODESIM interface is minimal, in an effort to make the package as easy to use as possible.

The result of this research is the CODESIM software facility: a machine-independent set of Fortran routines to control and support interactive discrete event simulation. Its straightforward pointer-based implementation enables facile conversion to other languages such as PL/1 or PASCAL. This paper outlines the overall structure of CODESIM and illustrates its use by example of a telephone model.

2. CODESIM

CODESIM was designed as a minimal, no-frills, package to support the essentials for discrete event modeling. As such, it comprises both entity control and statistical support routines with an interface to the user model. Figure 1 illustrates the 10 major components of CODESIM which serve as a noninvasive simulation environment for the modeler.

The CONTROL software is the CODESIM main program—it handles the Future Events Queue (FEQ), ensuring that time advances in a forward-only manner, and passing control to the user MODEL routine for the handling of each simulation event. A null pointer (i.e., empty FEQ) results in a normal halt; values outside the available space domain result in an erroneous halt with appropriate message.

Implicit to the CODESIM design is the tenet that the user need not learn all about the intricacies of the system internals in order to effectively use the package. CODESIM handles attribute space

---

\(^1\) CODESIM is a proprietary software package of Battelle's Columbus Laboratories. Further information regarding its availability and use can be obtained by contacting the author.
for the modeler, storing the event time, event
code, entity id, entity predecessor, entity suc-
cessor, and entity queuing time of each active
entity. The modeler is at liberty to define as
many additional attributes as desired, based upon
the active entity pointer.

The user-written MODEL routine acts as the con-
trol buffer between CODESIM and the User-written
Event Routines. Within MODEL the modeler uses
the event code of the active entity to determine
appropriate event routine invocation, including
simulation initialization and interactive report-
handling.

When an Event Routine is invoked the CODESIM
entity control routines come into play. The
modeler may KILL the entity when its role in the
simulation is complete. This will update the FEQ
and add the entity to the free storage pool. One
may otherwise SCHEDULE this entity for a success-
itive event, or ADDQ this entity to a particular
queue. The CREATE module enables initial space
allocation for a new entity, while DELQ enables
the head of a queue to be deleted.

Further simulation support is provided in the
statistical area: STATSI and STATSD derive time-
independent and time-dependent statistics respec-
tively. STATSR returns the min, max, current,
mean, standard deviation, and observations for
any given statistical variables to the modeler
routine for report generation. CODESIM provides
an automatic statistical collection of queue
usage, if so desired by the modeler, regarding
both queue lengths and entity times enqueued.

It is important to note that I/O is almost
entirely removed from the CODESIM software. The
user may alter the CODESIM default values if so
desired, but all input is modeler-handled. On
the output side, the CODESIM trace of events is
output (on the crt and a backup file for post-
mortem printout) if the modeler turns the CODESIM
trace variables on. Statistical reports are gen-
erated by the modeler—CODESIM will derive and
return statistics but the modeler must output
them and provide label identification.

3. THE USER MODEL

Subroutine MODEL is written by the modeler to con-
trol the simulation and to interface with CODESIM.
The very first time MODEL is called the FEQ is
empty, and all CODESIM variables are at their
default values. In this case MODEL must perform
whatever initialization is desired, to include:

1. Any alterations of CODESIM defaults,
   and

2. Posting new events to the FEQ.

The user MODEL routine will never again be called
with an empty FEQ, for that is permissible only
at simulation start-up. Note that since we are
creating an interactive model, crt i/o can be
carried out to determine MODEL variable initial
values and to determine event times.

The modeler may define as many attributes as de-
sired, using the active entity pointer provided
by CODESIM. The CREATE and SCHEDULE routines are
sufficient to post the new entities on the FEQ
for simulation start-up.

One of the events in routine MODEL should be a
crt Reporting Event, specified to occur at times
defined by the interactive user. In this way the
modeler can examine the simulation at any time
desired during the simulation execution.

Statistical collection and reports are also in
the modeler domain, and can be flexibly con-
structed with guidance from the interactive user.
Lastly the interactive user should be able to control the length of the simulation dynamically --running the model until the desired results are achieved.

4. EXAMPLE: THE TELEPHONE MODEL

Consider, for illustrative purposes, the following scenario for simulation: a telephone operator handling calls, with a maximum capability of 1 active caller and 4 callers on hold. We wish to model this in a process-oriented manner using CODESIM. So we write a MODEL routine with three events: START, RING, and REPORT using a computed goto on the event code of the active event to call the proper event routine.

START can interact with the user to define whatever parameters are desired, to include the next reporting time, call interarrival time, etc. Given this data START can SCHEDULE entities on the FEQ (using CREATEEs as necessary) for each different model event.

REPORT, when invoked, can interact with the crt user to determine what outputs are desired at this time, to alter the trace of other model variables, and to post statistics using STATSR for value determination. The next reporting time can be elicted from the crt user and SCHEDULEd on the FEQ, or the simulation can be halted.

The crux of the telephone model is routine RING, which handles all aspects of the phone call process. Table 1 illustrates this subroutine in seven paragraphs of code.

The first paragraph of code utilizes the modeler-defined ENTRY attribute to determine whether we are starting or ending a conversation. At label 2 the phone rings and we create the successor to this call, in bootstrap fashion. At label 3 we check for call abort; if so we collect abort stats, kill the entity and return.

Otherwise the call can go through at label 4, but it must be enqueued if someone else currently has the line. At label 5 the line is open so we determine call length, schedule call completion and return. Upon return the ENTRY attribute will be incremented from 1 to 2 so control will pass to step 6 as desired. Here we collect stats on call time and kill the entity as we hang up.

At step 7 we check to see if someone else is on hold, and if so we take the first caller and initiate that conversation. In this way we can fully characterize the calling process within one event routine, rather than the event-oriented approach of two events: one for the call and one for its completion.

The next three questions emanate from the user model START routine, defining a nontraced run for 20 time units. At that time CODESIM calls the user MODEL routine which then invokes the user REPORT routine. The statistical report is output using routine STATSR provided by CODESIM, and describes the queueing time and size of queue Q1, the number of call aborts, and the length of those calls completed.

Further interaction at the end of the user REPORT routine enables the user to continue with selected changes to the model parameters. The crt terminal TRACE is turned on for the next 2 time units and control returned to CODESIM. We see the next report interrupt posted, the call completion processing for call 30, and the call start processing for call 31. The lines without the word CODESIM emanate from the user model.

6. CONCLUSIONS

This paper has outlined the essential features of CODESIM, and demonstrated its use through a simple informative example. The CODESIM software package has decided advantages in size and speed over other discrete modeling packages. Almost complete programmer independence is provided with a single minimal common block. The pointer concept for queues frees the programmer from buffering the current active entity and enhances speed as well.

The primitive routines of CODESIM are sufficient for typical discrete modeling . . . the design is predicated on providing a flexible foundation for modeling. Certain models no doubt will result in unique extensions to CODESIM. Our aim was to provide a basis which would not impede immediate use or quick extensions.

CODESIM was successfully utilized at Battelle to support a discrete event simulation of data traffic on a computer-to-computer high speed channel link (Freuler and Rose 1983). It proved to be sound, easy to use, fast, and efficient for interactive modeling. It is further providing the basis for a hierarchica extension for generalized modeling (Rose 1983a).

The telephone model presented herein illustrated the capabilities for CODESIM to support interactive modeling in a format-free model. Future research at Battelle is planned for the incorporation of two nontrivial GASP-based models into CODESIM, enabling cost-effective, feasible interactive use (Rose 1983b, Rose and Freuler 1982). Additional research is planned in the portability area, both in different machines and different languages.

5. CODESIM OUTPUTS

Table 2 illustrates a portion of the output generated by the example Telephone Model. The header information is output by CODESIM upon the first modeler invocation of the CREATE routine. This provides the modeler an opportunity at simulation start-up to override in MODEL or its subroutines any default values in CODESIM.
```c
SUBROUTINE RING(PTR)

...DETERMINE PROPER ENTRY POINT THIS ENTITY...
1 ENTRY(PTR) = ENTRY(PTR) + 1
    LOC = ENTRY(PTR)
    GO TO (2,6),LOC

...INCOMING CALL: BOOTSTRAP...
2 IF(TRACE .EQ. 1.) PRINT*, "BR-RING"
    CALL CREATE(KPTR)
    ENTRY(KPTR) = 0
    CALL SCHED(KPTR, RNG, 1.2*RAND(1))

...ABORT IF ALL LINES IN USE, ELSE HOLD OR CONVERSE...
3 BEGIN(PTR) = T
    IF(INUSE .LT. 5) GO TO 4
    ABORTS = ABORTS + 1.
    CALL STATSD(MAXQ+1, ABORTS)
    CALL KILL(PTR)
    IF(TRACE .EQ. 1.) PRINT*, "HANG UP"
    RETURN

...ON HOLD IF ALREADY CONVERSING...
4 INUSE = INUSE + 1
    IF(INUSE .EQ. 1) GO TO 5
    CALL ADDQ(PTR,Q1)
    IF(TRACE .EQ. 1.) PRINT*, "PUT ON HOLD"
    RETURN

...START CONVERSATION AND SCHEDULE COMPLETION...
5 IF(TRACE .EQ. 1.) PRINT*, "HELLO"
    CALL SCHED(PTR, RNG, 1.5*RAND(2))
    RETURN

...END OF TALKING: STATS...
6 TSPAN = T - BEGIN(PTR)
    IF(TRACE .EQ. 1.) "GOODBYE ... CALL TIME =", TSPAN
    CALL STATSI(MAXQ+1, TSPAN)
    CALL KILL(KPTR)
    INUSE = INUSE - 1

...SERVICE WAITING CALL...
7 CALL DELQ(PTR,Q1)
    IF(PTR .NE. 0) GO TO 5
    RETURN

END
```
### TABLE 2. CODESIM SAMPLE OUTPUT

```
<table>
<thead>
<tr>
<th>STAT</th>
<th>MEAN</th>
<th>SIGMA</th>
<th>MIN.</th>
<th>MAX.</th>
<th>OBS.</th>
<th>CURRENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 TIME</td>
<td>.2014E+01</td>
<td>.8320E+00</td>
<td>.3975E+00</td>
<td>.3611E+01</td>
<td>.2300E+02</td>
<td>.2415E+01</td>
</tr>
<tr>
<td>Q1 SIZE</td>
<td>.2492E+01</td>
<td>.1306E+01</td>
<td>0.</td>
<td>.4000E+01</td>
<td>.1975E+02</td>
<td>.3000E+01</td>
</tr>
<tr>
<td>#ABORTS</td>
<td>.1344E+01</td>
<td>.2857E+01</td>
<td>.1000E+01</td>
<td>.5000E+01</td>
<td>.1836E+02</td>
<td>.5000E+01</td>
</tr>
<tr>
<td>CALLTIM</td>
<td>.2768E+01</td>
<td>.9210E+00</td>
<td>.4789E+00</td>
<td>.3924E+01</td>
<td>.2300E+02</td>
<td>.3544E+01</td>
</tr>
</tbody>
</table>

SHALL I CALL THE CODESIM BOMB ROUTINE FOR CHECKOUT? no
DO YOU WISH TO STOP THE MODEL NOW? no
DO YOU WISH TO CHANGE THE MODEL PARAMETERS? yes
ENTER: MAXNTT, TTRACE, QSTATS

==>

ENTER NEXT INTERRUPT TIME 22

* CODESIM: T = 20.0000  SCHEDL EVENT 2 AT 22.0000 FOR ENTITY 1
* CODESIM: T = 20.0546  MODEL: EVENT 3 WITH ENTITY 30
GOOD BYE... CALL TIME = .2715E+01
* CODESIM: T = 20.0546  KILL.. ENTITY 30
* CODESIM: T = 20.0546  REMOVE ENTITY 31 AT FRONT OF QUEUE 1
HELLO
RANDOM NUMBER .1925  DRAWN FROM STREAM 2
* CODESIM: T = 20.0546  SCHEDL EVENT 3 AT 20.3434 FOR ENTITY 31
* CODESIM: T = 20.3434  MODEL: EVENT 3 WITH ENTITY 31
GOOD BYE... CALL TIME = .2468E+01
* CODESIM: T = 20.3434  KILL.. ENTITY 31
* CODESIM: T = 20.3434  REMOVE ENTITY 33 AT FRONT OF QUEUE 1
HELLO
RANDOM NUMBER .9239  DRAWN FROM STREAM 2
```
REFERENCES


Rose LL (March 1983), Production Modeling with PRISIM. In: Proceedings 16th Annual Simulation Symposium, pp. 41-54.