Edited by Tuncer I. Oren • *University of Ottawa*

DISCO – a SIMULA-based language for continuous combined and discrete simulation

by

Keld Helsgaun
Department of Computer Science
Roskilde University Center
4000 Roskilde, Denmark

KELD HELSGAUN received his MSc in computer science and mathematics in 1973 from the University of Copenhagen in Denmark. For the past five years he has served as lecturer at Roskilde University Center in Denmark. His research interests include simulation languages, system theory, and artificial intelligence.

SUMMARY

DISCO is a SIMULA-based language for the simulation of systems which contain both continuous and discrete-event processes. It is a true extension of SIMULA's concepts for discrete-event simulation and is capable of representing and analyzing models including both types of processes. With the addition of few new concepts, a general yet simple simulation language has been achieved.

This paper describes the basic concepts of DISCO and demonstrates through examples the descriptive power of the language. DISCO is described mainly from the user's point of view; its implementation is only sketched.

Keywords: combined discrete and continuous simulation, digital simulation (combined systems), SIMULA, simulation languages

INTRODUCTION

A central consideration in the development of a simulation model is the representation of the system in question.

Properties of the system (such as continuity, simultaneity, and interdependence) may make it difficult to construct such a representation. Moreover, the description tool, that is, the simulation language, may prescribe a "world view" which makes the task awkward, or even impossible, to accomplish.

Two extreme views are found in the so-called *discrete* and *continuous* simulation languages.

The discrete simulation languages (such as GPSS, SIMSCRIPT and SIMULA) stress the viewpoint that state variables interact discretely, that is, instantaneously and only at particular points in time (event times).

In contrast, the continuous simulation languages (such as CSMP, DARE, and ACSL) stress the viewpoint that state variables interact continuously – a viewpoint that leads to models expressed by differential equations.

Most current simulation languages belong to one of the two types. However, there are systems containing important interactions between discrete and continuous subsystems so that neither type of language is adequate by itself for simulation. Both types introduce limitations to the formulation of a model, limitations which may be so severe that certain aspects of a system can not be described.

Some continuous simulation languages can accommodate simple types of discontinuities, such as step functions and hysteresis functions. However, use of these features often causes serious numerical difficulties. If a fixed-step-size integration algorithm is used, the accuracy suffers; on the other hand, a variable-step-size algorithm gives rise to ineffi-

ciency, since the step-size is reduced to very small values in the neighbourhood of a discontinuity.

Systems requiring a combined continuous and discrete description are typically systems in which discrete actions are superimposed on continuous subsystems. Many such systems are found in industry. For example, consider a steel-making process. Steel ingots with different arrival times are heated to a desired temperature in a furnace. The heating of each ingot is a continuous process, while arrival and departure of the ingots are discrete events.

Simulation of industrial systems, however, is not the only area in which a combined simulation approach may be appropriate. In fact, there exists a diversity of systems involving both continuous and discrete phenomena.

To name a few, first consider treatment of diabetes. The biochemical reactions are continuous processes. Injections of insulin and ingestion of food may be considered as discrete events. Another example is automobile traffic. The vehicle dynamics constitute the system's continuous part, and the queueing and driver decisions, the discrete part. Lastly, consider the human brain. The biochemical reactions are continuous processes, whereas the triggering of a neural impulse is a discrete event.

To describe such mixed systems, it may be necessary to use a language combining the two types of language, a so-called *combined-system simulation language*.

One of the first advocates of combined-system simulation languages was Fahrland[7] in 1970. As late as 1969 most simulation specialists did not see any need for combined languages. Today this situation has changed. More complex systems are simulated, and the value of combined languages is commonly recognized. The number of software packages developed for combined-system simulation is rather impressive.[16]

FORTRAN is the language which is used in most of the implementations. GASP IV,[18] probably the most used package for combined simulation, is a collection of FORTRAN subroutines. In some packages the host language is PL/I, for example, PROSIM[19] and GASP-PL/I.[17] In general these packages are more powerful than packages based on FORTRAN because of the inherent properties of PL/I. However, neither FORTRAN nor PL/I is well suited for system description in that these languages lead to an inconvenient notation and often to a lack of generality in the formulation of the model.

Some few combined-system simulation languages have their own translator, for example GSL[9] and COSY[4]; but, for the most part, the target language is FORTRAN. The COSY translator generates GASP IV programs, an approach which has the drawback that GASP IV restrictions are passed on to COSY (for example, the data structuring capabilities of COSY which are as limited as those offered by FORTRAN).

The objective of this paper is to present DISCO, a SIMULA-based language for DIScrete and COntinuous system simulation. This language is an elaboration of an earlier-described SIMULA *class* called COMBINEDSIMULATION.[10] DISCO is also provided in the form of a so-called SIMULA *class* and as such it needs no special translator or preprocessor.

SIMULA is a general-purpose programming language.[1,5] In addition to the facilities of ALGOL 60, it offers the class and coroutine concepts, reference variables, list-handling facilities, discrete-event simulation facilities, and extensive text and input/output capabilities.

Using SIMULA as host language for a combined-system simulation language offers several advantages. First, SIMULA is very well suited for system description. Its class concept is a powerful tool for modelling. Class concatenation opens the prospect of representing extremely complex systems by organizing the model into a hierarchy of submodels. Second, extra features can be added to SIMULA so neatly that they appear to be extensions to the language. Third, SIMULA has excellent facilities for discrete simulation. Its discrete-process concept is rather strong, making possible process-oriented as well as activity- and event-oriented approaches to simulation.[11] SIMULA encompasses both GPSS and SIMSCRIPT, offering the user expressive and general means for decomposing and describing systems.[12]

DISCO generalizes the process concept of SIMULA to include continuous processes and interaction between continuous and discrete processes. A SIMULA user will find DISCO relatively easy to learn and use.

The following sections introduce DISCO and give examples of its use. Some knowledge of SIMULA is an advantage but not a necessity. The basic ideas of DISCO are accessible without such knowledge.

MODELLING PHILOSOPHY

DISCO builds on SIMULA's process view of simulation.[2,8]

A system is conceived of as a collection of processes which undergo active and inactive phases and whose actions and interactions comprise the behaviour of the system.

In DISCO a distinction is made between two types of processes, *continuous processes* and *discrete processes*. Continuous processes undergo active phases during time intervals and cause continuous changes of state. In contrast, discrete processes have instantaneous active phases, called *events*, and cause discrete changes in the state of the system. The events of a discrete process are separated by periods of inactivity, during which continuous processes may be active.

Any process may be created, activated, deactivated, or removed from the system at any time. However, DISCO permits more than coexistence of separate processes. It allows processes to communicate with one another and to influence each other in a completely general way. Any process may reference and modify any variable in any other process and may affect delimiting and sequencing of active phases.

BASIC CONCEPTS

DISCO is a true extension of SIMULA's system-defined class for discrete-event simulation, class SIMULATION, so that all the latter's concepts are available to the user. Thus, class PROCESS and the event scheduling constructs (HOLD, activate, etc.) can readily be used in describing the discrete processes of a system.

In order to make possible description of systems involving continuous processes as well, DISCO provides the following additional concepts:

class VARIABLE

class CONTINUOUS

procedure WAITUNTIL

The most essential attributes of DISCO are shown in the class outline below.

```
SIMULATION class DISCO;
begin
  class VARIABLE(STATE); real STATE;
  begin
    real RATE;
    procedure START; ... ;
    procedure STOP; ... ;
  end;

  class CONTINUOUS;
  begin
    procedure START; ... ;
    procedure STOP; ... ;
  end;

  procedure WAITUNTIL(B); name B; Boolean B; ... ;

  real DTMIN, DTMAX, MAXABSERROR, MAXRELERROR;
end;
```

Class VARIABLE

Objects of class VARIABLE can be used to represent state variables that vary according to ordinary first-order differential equations.

The value of such a variable is denoted by STATE, while RATE denotes its derivative with respect to time. The initial value of STATE is passed as a parameter on object generation.

After its START-procedure is called, a VARIABLE-object becomes *active*, that is to say, its STATE undergoes "continuous" change between discrete events. The value of STATE is changed according to the value of RATE, as computed by the active continuous processes. The active phase will cease when the object's STOP-procedure is called.

Example:

An object of class VARIABLE, say X, may be generated with an initial STATE-value of 3.14 by the statement

```
X:-new VARIABLE(3.14);
```

X's attributes STATE and RATE are designated X.STATE and X.RATE. The object is started and stopped by calling X.START and X.STOP.

Class CONTINUOUS

Class CONTINUOUS can be used to describe continuous processes defined by ordinary differential equations. The description is given in one or more subclasses which compute derivatives of state variables.

After its START-procedure is called, a CONTINUOUS-object becomes *active*, that is to say, its user-defined actions are executed "continuously." This active phase will cease when the object's STOP-procedure is called.

Example:

A continuous process defined by the two first-order differential equations (Lotka-Volterra equations)

$$dX/dt = (A+BY)X$$
$$dY/dt = (C+DX)Y$$

can be described by the following declaration

```
CONTINUOUS class DYNAMICS;
begin
  X.RATE:=(A+B★Y.STATE)★X.STATE;
  Y.RATE:=(C+D★X.STATE)★Y.STATE;
end;
```

where X and Y are VARIABLE-objects.

An object of this class, say EVOLUTION, is generated by the statement

```
EVOLUTION:-new DYNAMICS
```

and is started and stopped by calling EVOLUTION.START and EVOLUTION.STOP.

Moreover, by exploiting SIMULA's class concept it is possible to define *macros* with general dummy variables, as, for example,

```
CONTINUOUS class FILTER(P,Q,QDOT,R,G);
ref(VARIABLE) P,Q,QDOT; real R,G;
begin
  Q.RATE:=QDOT.STATE;
  QDOT.RATE:=P.STATE-R★QDOT.STATE-G★Q.STATE;
end;
```

which defines a filter with input P, output Q, and parameters R and G such that

$$\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + GQ = P$$

Procedure WAITUNTIL

Procedure WAITUNTIL can be used to schedule a discrete event to occur as soon as a prescribed system state is reached. Such an event is called a *state-event*, in contrast to a *time-event* which is an event scheduled to occur at a specified projected point in time.

WAITUNTIL(B) causes the active discrete process CURRENT to become passive until the Boolean expression B evaluates as true. However, this phase of inactivity may be ended sooner by an explicit activation of the waiting process.

The procedure is a generalization of the WAITUNTIL-construct of the discrete simulation language SOL.[14]

Example:

Typically procedure WAITUNTIL is used to schedule an event to occur when a state variable crosses a prescribed threshold.

By calling

```
WAITUNTIL(X.STATE>100)
```

the active discrete process postpones its actions until X's STATE becomes greater than 100.

The state-condition could have been more complex, as for example in the call

```
WAITUNTIL(X.STATE>100 and Y.RATE<0)
```

In fact, a state-condition may be of arbitrary complexity.

DTMIN, DTMAX, MAXABSERROR, MAXRELERROR

Between the event times the state of the model is automatically updated in steps of varying size. DTMIN and DTMAX are used to specify the minimum and the maximum allowable step-sizes. MAXABSERROR and MAXRELERROR can be used to specify the maximum absolute and maximum relative error allowed in updating the STATE-values of the active VARIABLE-objects.

Example:

Below is given a complete DISCO program which simulates a predator-prey system. The use of class DISCO is indicated by prefixing the program with the name DISCO (line 1). The simulation period is 100 time units (line 16). Note that the main program (lines 11-16) is a discrete process (in this example, the only one).

```
 1. DISCO
 2. begin
 3.   CONTINUOUS class DYNAMICS;
 4.   begin
 5.     X.RATE:=(A+B*Y.STATE)*X.STATE;
 6.     Y.RATE:=(C+D*X.STATE)*Y.STATE;
 7.   end;

 8.   ref(DYNAMICS) EVOLUTION;
 9.   ref(VARIABLE) X, Y;
10.   real A, B, C, D;

11.   DTMIN:=0.0001; DTMAX:=1; MAXRELERROR:=0.00001;
12.   A:= -0.3; B:=0.00002; C:=0.4; D:= -0.0001;
13.   X:-new VARIABLE(1000);   X.START;
14.   Y:-new VARIABLE(100000); Y.START;
15.   EVOLUTION:-new DYNAMICS; EVOLUTION.START;
16.   HOLD(100);
17. end;
```

The system is known to be cyclic. The period can be determined through procedure WAITUNTIL by computing the time interval between two consecutive maximum points of a state variable, say X. In order to achieve this the HOLD-statement of line 16 may be replaced by the following:

```
begin
  real CYCLESTARTTIME;

  WAITUNTIL(X.RATE>0);
  WAITUNTIL(X.RATE<=0);
  comment *** first maximum found ***;

  CYCLESTARTTIME:=TIME;
  WAITUNTIL(X.RATE>0);
  WAITUNTIL(X.RATE<=0);
  comment *** second maximum found ***;

  OUTTEXT("Period =");
  OUTREAL(TIME-CYCLESTARTTIME,5,12);
end;
```

A SIMPLE EXAMPLE

The following example shows the use of class VARIABLE and class CONTINUOUS.

The example has been taken from Reference 21 and has to do with the launch of a three-stage rocket from the earth's surface.

The model has both continuous and discrete elements. During the rocket's flight its motion is governed by well-known physical laws, and its mass decreases continuously as a result of the expulsion of burnt fuel. However, when a stage separates from the rest of the rocket, an instantaneous change of the rocket's mass and acceleration takes place.

A complete simulation program is shown in Figure 1.

The continuous motion of the rocket is described in the class ROCKETMOTION (lines 3-11). The change in mass takes place at a constant rate, MASSFLOW (line 5). There are three forces acting upon the rocket: THRUST, DRAG, and GRAVITY. THRUST is the force generated by the expulsion of burnt fuel (line 6), DRAG is the air resistance (line 7), and GRAVITY is the earth's gravitational attraction on the rocket (line 8).

From Newton's second law of motion the rocket's acceleration can be determined by summing these three forces and dividing by the rocket's mass (line 9). The change of rate of altitude is equal to the rocket's velocity (line 10).

The discrete changes (that is, the separation of stages) are described in the main program (lines 14-27).

The rocket's launch begins when the three VARIABLE-objects MASS, VELOCITY, and ALTITUDE are generated and STARTed together with an object of class ROCKETMOTION (lines 17-20). Each time a stage separates, discrete changes in MASS, MASSFLOW, FLOW-VELOCITY, and AREA take place (lines 23 and 26). These events are scheduled by the procedure HOLD (lines 21, 24, and 27).

This example illustrates one type of continuous-discrete interaction, namely, discrete changes of "continuous" variables.

Two other types may also be modelled with DISCO.

The triggering of a discrete event as a consequence of the fulfilment of a state-condition is one type. This type can be expressed using the procedure WAITUNTIL. In the example it is known at which

```
1. DISCO
2. begin
3.   CONTINUOUS class ROCKETMOTION;
4.   begin
5.     MASS.RATE:=MASSFLOW;
6.     THRUST:=MASS.RATE*FLOWVELOCITY;
7.     DRAG:=AREA*0.00119*EXP(-ALTITUDE.STATE/24000)*VELOCITY.STATE**2;
8.     GRAVITY:=MASS.STATE*32.17/(1+ALTITUDE.STATE/20908800)**2;
9.     VELOCITY.RATE:=(THRUST-DRAG-GRAVITY)/MASS.STATE;
10.    ALTITUDE.RATE:=VELOCITY.STATE;
11.  end *** ROCKETMOTION ***;
12.  ref(VARIABLE) MASS, VELOCITY, ALTITUDE;
13.  real THRUST, DRAG, GRAVITY, MASSFLOW, FLOWVELOCITY, AREA;
14.  DTMIN:=0.00001; DTMAX:=4; MAXABSERROR:=MAXRELERROR:=0.0001;
15.  comment *** FIRST STAGE ***;
16.  MASSFLOW:=-930; FLOWVELOCITY:=-8060; AREA:=510;
17.  MASS:-new VARIABLE(189162); MASS.START;
18.  VELOCITY:-new VARIABLE(0.0); VELOCITY.START;
19.  ALTITUDE:-new VARIABLE(0.0); ALTITUDE.START;
20.  new ROCKETMOTION.START;
21.  HOLD(150);
22.  comment *** SECOND STAGE ***;
23.  MASS.STATE:=40342; MASSFLOW:=-81.49; FLOWVELOCITY:=-13805; AREA:=460;
24.  HOLD(359);
25.  comment *** THIRD STAGE ***;
26.  MASS.STATE:=8137; MASSFLOW:=-14.75; FLOWVELOCITY:=-15250; AREA:=360;
27.  HOLD(479);
28. end *** LAUNCH OF A THREE-STAGE ROCKET ***;
```

Figure 1 - DISCO program for the rocket example

times stage separation occurs, so the *imperative* scheduling procedure HOLD is used to schedule these events. If this were not the case, for example if stage separation were dependent on altitude, then the *interrogative* scheduling procedure WAITUNTIL could be used instead, e.g., WAITUNTIL(ALTITUDE.STATE>25000).

The other type of continuous-discrete interaction allows for dynamic starting and stopping of continuous processes. This capability makes possible simulation of systems in which the differential equations themselves vary with time. In the example we could have represented the rocket's motion by three continuous processes, one for each phase of motion. A stage separation would then cause the current active continuous process to be stopped and replaced by the continuous process that corresponds to the next phase.

THREE EXAMPLES

The descriptive power of DISCO is best appreciated through examples. This section presents three examples illustrating DISCO's capability for modelling combined problems.

Example 1: Fire-fighting

The first example is a simulation of a fire station. The model is formulated by R. W. Sierenberg, Delft University, the Netherlands.

A small city owns one fire station with three fire engines. Fire alarms are given randomly at exponentially distributed intervals with a mean of six hours. Each house on fire contains a certain amount of flammable material. When a fire is discovered, it already has a certain size. The fire increases with a rate which is proportional to its size as long as no extinguishing activity takes place.

At the moment an alarm is given, one fire engine, if it is available, will be sent to the fire. When a fire engine reaches the fire and finds out that its capacity is smaller than the rate with which the fire increases, it will request assistance by sending a second alarm for the same fire. The fire engine returns to the fire station after the fire is put out or when all inflammable material is consumed.

A program for simulating the fire station is shown in Figure 2. The program should be fairly self-explanatory.

Class HOUSEONFIRE (lines 3-18) describes the discrete events associated with a house on fire: alarm call (line 14) and fire termination (line 17).

Class BURNING (lines 19-24) describes, through differential equations, the continuous process associated with a house on fire.

Class FIREENGINE (lines 25-45) defines the fire engines.

An INCENDIARY-object (lines 46-48) sets houses on fire.

The simulation period is one month (line 56).

```
1. DISCO
2. begin
3.   PROCESS class HOUSEONFIRE;
4.   begin
5.     real MATERIAL, EXTINGUISHRATE, TRAVELTIME, C;
6.     ref(VARIABLE) SIZE, DAMAGE;
7.     ref(BURNING) FIRE;
8.     MATERIAL:=UNIFORM(100,600,SEED);
9.     TRAVELTIME:=UNIFORM(5,15,SEED);
10.    C:=UNIFORM(0.01,0.06,SEED);
11.    SIZE:-new VARIABLE(UNIFORM(0,5,SEED)); SIZE.START;
12.    DAMAGE:-new VARIABLE(SIZE.STATE); DAMAGE.START;
13.    FIRE:-new BURNING(this HOUSEONFIRE); FIRE.START;
14.    new ALARM(this HOUSEONFIRE).INTO(ALARMQ);
15.    activate FIRESTATION.FIRST;
16.    WAITUNTIL(SIZE.STATE<=0 or DAMAGE.STATE>=MATERIAL);
17.    FIRE.STOP; DAMAGE.STOP; SIZE.STOP;
18.  end *** HOUSEONFIRE ***;

19.  CONTINUOUS class BURNING(HOUSE); ref(HOUSEONFIRE) HOUSE;
20.  inspect HOUSE do
21.  begin
22.    SIZE.RATE:=C*SIZE.STATE-EXTINGUISHRATE;
23.    DAMAGE.RATE:=SIZE.STATE;
24.  end *** BURNING ***;

25.  PROCESS class FIREENGINE(CAPACITY); real CAPACITY;
26.  while true do
27.  inspect ALARMQ.FIRST when ALARM do
28.  begin
29.    this ALARM.OUT;
30.    if HOUSE.FIRE.ACTIVE then
31.    begin
32.      this FIREENGINE.OUT;
33.      HOLD(HOUSE.TRAVELTIME);
34.      if HOUSE.SIZE.RATE>CAPACITY then
35.      begin
36.        this ALARM.INTO(ALARMQ);
37.        activate FIRESTATION.FIRST;
38.      end;
39.      HOUSE.EXTINGUISHRATE:=HOUSE.EXTINGUISHRATE+CAPACITY;
40.      while HOUSE.FIRE.ACTIVE do HOLD(5);
41.      HOLD(HOUSE.TRAVELTIME);
42.      this FIREENGINE.INTO(FIRESTATION);
43.    end;
44.  end
45.  otherwise PASSIVATE;

46.  PROCESS class INCENDIARY;
47.  while true do
48.  begin HOLD(NEGEXP(1/(60*60),SEED); activate new HOUSEONFIRE; end;

49.  LINK class ALARM(HOUSE); ref(HOUSEONFIRE) HOUSE; ;

50.  ref(HEAD) FIRESTATION, ALARMQ; integer I, SEED;
52.  DTMAX:=10000; MAXRELERROR:=0.00001; SEED:=54521;
53.  FIRESTATION:-new HEAD; ALARMQ:-new HEAD;
54.  for I:=1,2,3 do new FIREENGINE(10).INTO(FIRESTATION);
55.  activate new INCENDIARY;
56.  HOLD(30*24*60);
57. end *** SIMULATION OF A FIRE STATION ***;
```

Figure 2 - DISCO program for the fire-fighting example

Example 2: Chemical reactor system

The simulation of a chemical reaction process is an example used by Hurst and Pritsker[13] to illustrate GASP IV's capability for combined simulation. To facilitate a comparison between DISCO and GASP, the system description given by Hurst and Pritsker is closely followed.

A chemical reactor system consists of a compressor, a surge tank, and four reactors (see Figure 3).

The reactors are charged with reactants which are supplied with hydrogen from the compressor through the surge tank. The reactants react under pressure with the hydrogen which causes the concentration of the reactants to decrease. The effective pressure in each reactor is automatically adjusted to the minimum of the surge tank pressure and critical pressure (100 psia).

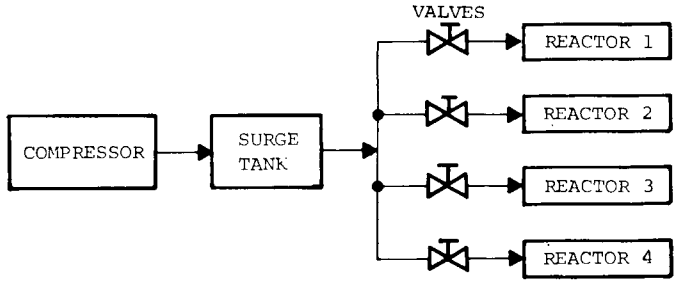

Figure 3 - Schematic diagram of the reactor system

When the concentration of a reactant decreases to 10% of its initial value, the reaction is considered complete and the reactor is turned off and made ready for a fresh batch of reactant.

Initially the surge tank pressure is 500 psia. If the pressure falls below the critical value of 100 psia, the last reactor started will be turned off immediately. The other reactors will continue, but no reactor will be started as long as surge tank pressure is below a nominal pressure of 150 psia.

If two or more reactors can start at the same time, the reactor with the highest value of accumulated batch processing time will be started first.

The reactor system involves both discrete events and continuous state variables. The starting and stopping of reactors are discrete events. Between these events concentration and pressure vary continuously.

The program in Figure 4 gives a precise description of the system.

The discrete processes of the system are described in the classes REACTOR and INTERRUPTER.

Class REACTOR (lines 3-22) describes the reactors and associated events. Starting a reactor is allowed only if the surge-tank pressure is above the nominal pressure (line 13). When the reaction process is completed, the reactor is stopped (line 16), cleaned, and recharged for a fresh batch (line 20).

Class INTERRUPTER (lines 23-30) describes the discrete events occuring whenever surge-tank pressure drops to the critical pressure. An INTERRUPTER-object sees that the last reactor started is turned off when the pressure falls below the critical pressure. Further, the object ensures that the effective pressure in the reactors is the minimum of surge tank pressure and the critical pressure.

The continuous processes of the system are described in the class REACTIONS (lines 31-43). The class is used to define a single continuous process having a variable number of differential equations. The process computes the active reactors' rate of change in concentration (lines 37-38) and the rate of change in surge-tank pressure (line 42).

The main program (lines 47-57) specifies the initial conditions, the step-size and accuracy requirements, and the length of the simulation period. Initially the four reactors are scheduled to be turned on at intervals of half an hour (lines 51-54). The simulation period is 150 hours (line 57).

The whole program takes only 58 program lines. In comparison, the corresponding GASP IV program[13], without reporting and without comments, takes about 220 lines. Note that the DISCO program has no comments; the high degree of structure makes the program on the whole self-explanatory.

Example 3: DOMINO game

This example was suggested by F. E. Cellier[3] as a benchmark problem that can be used to test the capability of a variable-structure simulation, that is, a simulation in which the number of differential equations varies with time.

Fifty-five identical stones of the DOMINO game are placed vertically upright in a sequence with a distance of D space units between any two stones. If the first stone is pushed, a chain reaction is started and all stones fall flat.

The aim of the simulation is to determine the distance (D) between successive stones which maximizes the velocity (V) of the chain.

A precise description of the model and the experiment is given in the program in Figure 5. The program illustrates how the specification of the model (class DOMINOGAME, lines 2-38) can be separated from the description of the experiment.

A STONE-object (lines 5-21) represents a falling stone and the associated discrete events: the pushing of the next stone (line 16) and the termination of the fall (line 20).

The fall is governed by Newton's law. For each stone we have the equation

$$\Theta \cdot \phi'' = m \cdot g \cdot R \cdot \sin\phi$$

where Θ is the moment of inertia, ϕ the inclination, m the mass, R half the side diagonal, and g the acceleration of gravity (see Figure 6).

In class STONEFALL (lines 22-27) this second-order differential equation is formulated by means of two first-order differential equations.

Each falling stone is associated with one STONEFALL-object (line 12). A stone which is not moving, either

```
1. DISCO
2. begin
3.   PROCESS class REACTOR(REACTIONCONSTANT,VOLUME,INITIALCONCENTRATION);
4.   real REACTIONCONSTANT,VOLUME,INITIALCONCENTRATION;
5.   begin
6.     ref(VARIABLE) CONCENTRATION;
7.     real PROCESSINGTIME, STARTTIME;
8.     CONCENTRATION:-new VARIABLE(INITIALCONCENTRATION);
9.   FRESHBATCH:
10.    while CONCENTRATION.STATE>0.10*INITIALCONCENTRATION do
11.    begin
12.      WAITPRIORITY:=PROCESSINGTIME;
13.      WAITUNTIL(PRESSURE.STATE>=NOMINALPRESSURE);
14.      CONCENTRATION.START; INTO(STARTED); STARTTIME:=TIME;
15.      WAITUNTIL(CONCENTRATION.STATE<=0.10*INITIALCONCENTRATION);
16.      CONCENTRATION.STOP; OUT;
17.      PROCESSINGTIME:=PROCESSINGTIME+(TIME-STARTTIME);
18.    end;
19.    CONCENTRATION.STATE:=INITIALCONCENTRATION; PROCESSINGTIME:=0;
20.    HOLD(NEGEXP(1,SEED));  HOLD(MIN(NORMAL(1,0.5,SEED),2));
21.    goto FRESHBATCH;
22.  end *** REACTOR ***;

23.  PROCESS class INTERRUPTER;
24.  while true do
25.  begin
26.    EFFECTIVEPRESSURE:-new VARIABLE(CRITICALPRESSURE);
27.    WAITUNTIL(PRESSURE.STATE<=CRITICALPRESSURE);
28.    activate STARTED.LAST; EFFECTIVEPRESSURE:-PRESSURE;
29.    WAITUNTIL(PRESSURE.STATE>CRITICALPRESSURE);
30.  end *** INTERRUPTER ***;

31.  CONTINUOUS class REACTIONS;
32.  begin
33.    ref(REACTOR) R;  real FLOWTOREACTORS;
34.    R:-STARTED.FIRST; FLOWTOREACTORS:=0;
35.    while R=/=none do
36.    begin
37.      R.CONCENTRATION.RATE:=-R.REACTIONCONSTANT*R.CONCENTRATION.STATE
38.                              *EFFECTIVEPRESSURE.STATE;
39.      FLOWTOREACTORS:=FLOWTOREACTORS+(-R.CONCENTRATION.RATE*R.VOLUME);
40.      R:-R.SUC;
41.    end;
42.    PRESSURE.RATE:=FACTOR*(FLOWFROMCOMPRESSOR-FLOWTOREACTORS);
43.  end *** REACTIONS ***;

44.  ref(VARIABLE) PRESSURE, EFFECTIVEPRESSURE;
45.  real FLOWFROMCOMPRESSOR, FACTOR, NOMINALPRESSURE, CRITICALPRESSURE;
46.  ref(HEAD) STARTED;  integer SEED;

47.  DTMIN:=0.001; DTMAX:=1; MAXRELERROR:=MAXABSERROR:=0.00001;
48.  FLOWFROMCOMPRESSOR:=4.3523; FACTOR:=107.03; SEED:=7913;
49.  NOMINALPRESSURE:=150; CRITICALPRESSURE:=100; STARTED:-new HEAD;
50.  PRESSURE:-new VARIABLE(500); PRESSURE.START;
51.  activate new REACTOR(0.03466,10,0.1) at 0.0;
52.  activate new REACTOR(0.00866,15,0.4) at 0.5;
53.  activate new REACTOR(0.01155,20,0.2) at 1.0;
54.  activate new REACTOR(0.00770,25,0.5) at 1.5;
55.  activate new INTERRUPTER;
56.  new REACTIONS.START;
57.  HOLD(150);
58. end *** REACTOR SIMULATION ***;
```

Figure 4 - DISCO program for the reactor example

```
1. begin
2.   DISCO class DOMINOGAME(NBRSTONES,D,G,M,X,Y,Z);
3.   integer NBRSTONES;  real D,G,M,X,Y,Z;
4.   begin
5.     PROCESS class STONE(INITIALOMEGA); real INITIALOMEGA;
6.     begin
7.       ref(VARIABLE) PHI, OMEGA;
8.       ref(STONEFALL) FALL;
9.       INTO(FALLINGSTONES); STONES:=STONES+1;
10.      PHI:-new VARIABLE(0.0); PHI.START;
11.      OMEGA:-new VARIABLE(INITIALOMEGA); OMEGA.START;
12.      FALL:-new STONEFALL(this STONE); FALL.START;
13.      if STONES<NBRSTONES then
14.      begin
15.        WAITUNTIL(PHI.STATE>=PHIPUSH);
16.        activate new STONE(KO*OMEGA.STATE);
17.        OMEGA.STATE:=KR*OMEGA.STATE;
18.      end;
19.      WAITUNTIL(PHI.STATE>=PI/2);
20.      OUT; FALL.STOP; PHI.STOP; OMEGA.STOP;
21.    end *** STONE ***;
22.    CONTINUOUS class STONEFALL(S); ref(STONE) S;
23.    inspect S do
24.    begin
25.      OMEGA.RATE:=KM*SIN(PHI.STATE)/THETA;
26.      PHI.RATE:=OMEGA.STATE;
27.    end *** STONEFALL ***;
28.    real V, KO, KM, KR, PHIPUSH, THETA, PI;
29.    integer STONES;
30.    ref(HEAD) FALLINGSTONES;
31.    DTMIN:=1.0&-6; DTMAX:=0.2; MAXABSERROR:=MAXRELERROR:=1.0&-4;
32.    KO:=1-(D-X)**2/Z**2; KM:=M*G*0.5*SQRT(X**2+Z**2); KR:=1-SQRT(KO);
33.    PHIPUSH:=ARCSIN((D-X)/Z); THETA:=M*(X**2+Z**2)/3; PI:=4*ARCTAN(1);
34.    FALLINGSTONES:-new HEAD;
35.    activate new STONE(Z/2*0.00001/THETA);
36.    WAITUNTIL(FALLINGSTONES.EMPTY);
37.    V:=D*(STONES-1)/TIME;
38.  end *** DOMINOGAME ***;
39.  real procedure VELOCITY(DIST); real DIST;
40.  inspect new DOMINOGAME(55,DIST,9.81,0.01,0.008,0.024,0.046) do
41.  VELOCITY:=V;
42.  real procedure MAXIMUM(Y,X,A,B,TOL); name Y,X; real Y,X,A,B,TOL;
43.  begin
44.    real X1, X2, Y1, Y2, F;
45.    F:=(SQRT(5)-1)/2;
46.    X:=X1:=B-F*(B-A); Y1:=Y;
47.    X:=X2:=A+F*(B-A); Y2:=Y;
48.    while B-X1>TOL do
49.    if Y1>=Y2
50.    then begin B:=X2; X2:=X1; Y2:=Y1; X:=X1:=B-F*(B-A); Y1:=Y; end
51.    else begin A:=X1; X1:=X2; Y1:=Y2; X:=X2:=A+F*(B-A); Y2:=Y; end;
52.    if Y1>=Y2 then begin X:=X1; MAXIMUM:=Y1; end
53.    else begin X:=X2; MAXIMUM:=Y2; end;
54.  end *** MAXIMUM ***;
55.  real DIST;
56.  OUTTEXT("The maximum chain velocity");
57.  OUTFIX(MAXIMUM(VELOCITY(DIST),DIST,0.008,0.008+0.046,0.001),3,6);
58.  OUTTEXT("m/s"); OUTIMAGE;
59.  OUTTEXT("is reached with a distance of");
60.  OUTFIX(DIST,4,7); OUTTEXT("m between stones.");
61. end *** DOMINO EXPERIMENTS ***;
```

Figure 5 - DISCO program for the DOMINO example

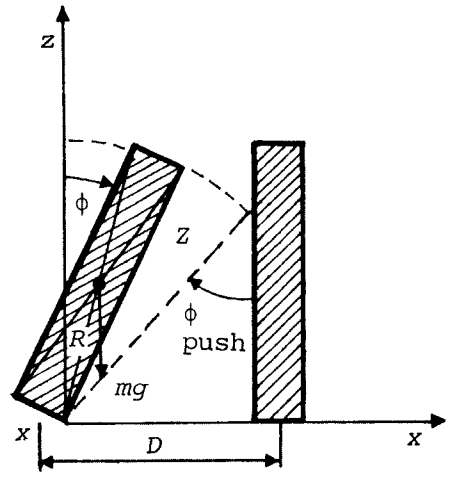

Figure 6 - Graphical description of a falling domino stone

because it has not yet been pushed, or because it has already fallen and lies still, is not associated with such an object. Thus, the number of differential equations will vary with time.

The procedure MAXIMUM (lines 42-54) determines a maximum of the function Y=F(X) on the interval [A,B]. The maximum point is found within the specified tolerance TOL by using the golden section search method.

For the given parameters the program produces the following output:

```
The maximum chain velocity 0.628m/s
is reached with a distance of 0.0204m between stones.
```

This example illustrates DISCO's capability for variable structure simulation. A feature of DISCO which is not demonstrated in the example is its capacity for direct communication between continuous processes. In the model it is assumed that a falling stone does not interact with any other stone, except at the very moment of impact. Thus, friction between stones is not part of the model. If friction were included, the falling stones would interact "continuously" exchanging information about their state variables. Since DISCO permits general communication between processes, including friction would be no problem.

ADDITIONAL FACILITIES

The examples in the preceding section illustrate the application of the essential facilities of DISCO. In this section some additional facilities are briefly mentioned.

Reporting

A class called REPORTER is provided for gathering information about model behaviour. Each REPORTER-object may have its user-defined actions executed with a specified frequency, namely, at uniformly spaced intervals, at the end of each time step, or only at event times. Facilities for producing line-printer plots and histogram are available for displaying information gathered during a simulation.

Integration

The user is offered the choice of six preprogrammed integration methods: Runge-Kutta-England, Euler, Trapez, Adams, Simpson, and Improved Heun. Any of these methods may be chosen at any time during a simulation. If the user does not specify an integration method, the program uses the fourth-order Runge-Kutta-England method.[6] The integration step-size is variable and is automatically adjusted to meet specified accuracy requirements. The user may specify error bounds individually for each VARIABLE-object and may determine the course of action when accuracy requirements can not be met.

In a continuous process the order in which the equations are written is left to the user. Because DISCO does not change the execution sequence of the equations, a correct sequencing is the responsibility of the user. To prevent unintentional delays from being introduced into the model dynamics, the user must make sure that the variables occurring on the right-hand side of an equation have values which reflect the current state of the system. The user can determine the order of evaluation within each continuous process, and the continuous processes themselves may be ranked by giving each a priority. Usually a correct evaluation order can be achieved by these means. An implicit function facility can be used to circumvent algebraic loops in the system of equations.

Additionally, it is possible to describe systems using *difference* equations. This capability may, for example, be used to specify models of the systems dynamics type.

Event sequencing

Discrete processes operate in *quasi-parallel* which means that concurrent events are executed in a certain order. Since the ordering may be important, the user must be able to determine their sequence of execution. SIMULA users are well acquainted with the language's facilities for sequencing time-events (before, after, prior, etc.). State-events, that is, events projected by procedure WAITUNTIL, can be ordered by giving each a priority, WAITPRIORITY (an example of this feature is illustrated in Figure 4, lines 12-13).

Utility software

The utility software includes among other things facilities for handling higher-order differential equations, ideal and exponential delays, and tabulated functions.

IMPLEMENTATION

A simulation is controlled behind the scenes, so to speak, by an object called the *monitor*.

It is the monitor's responsibility to see that

(1) The model state varies "continuously" between events

(2) Discrete events take place at the right time

(3) Information about the model's behaviour is gathered.

The monitor ensures that all continuous parts of the model operate in true parallel and are fully synchronized with the quasi-parallel discrete processes.

TIME in the model is advanced by steps of varying size. The monitor adjusts step-size so that no events occur within a step and so that desired accuracy in updating state variables is maintained. Roundoff errors are reduced by quasi-double-precision summation.[15]

The monitor causes the events to take place at the right time. The event times of state-events are determined with an accuracy of DTMIN using bisection and fifth-order Hermite interpolation.

The monitor controls objects of class REPORTER. Each REPORTER-object has its user-defined actions executed with a specified frequency. Hermite interpolation is also used here to provide an efficient and accurate determination of the model's state at the reporting times.

The working cycle of the monitor is outlined below.

```
while more projected events do
begin
   execute all active CONTINUOUS-objects;
   execute all active REPORTER-objects;

   while no event now do
   begin
      take an integration step fulfilling accuracy requirements;

      if a state-event was passed then
      determine the event time and reduce step accordingly;

      execute active REPORTER-objects when requested;
   end;

   let an event take place now;
end;
```

DESIGN OBJECTIVES

In the construction of DISCO the following design objectives have been emphasized.

Convenience

Most importantly, class DISCO is a convenient tool both for describing and simulating systems.

The class is a logical extension of class SIMULATION, SIMULA's conceptual framework for discrete-event simulation. As such all of class SIMULATION's facilities are available. Few new concepts together with a convenient notation make DISCO easy to learn and use. At Roskilde University Center we have used it successfully in our undergraduate courses having to do with modelling and simulation. The sharp distinction between discrete and continuous processes has aided in the conceptualization of combined systems.

Generality

Another important objective is generality.

DISCO permits general interaction between processes and ensures their synchronous operation. In this respect DISCO differs from a similar SIMULA-class called CADSIM.[20]

Model structures can be changed by the addition, substitution, or deletion of any type of process, thus allowing simulation of systems with a variable structure.

Extendability

SIMULA's class concept has been consistently exploited to facilitate construction of special and general-purpose extensions of class DISCO. For example, through class concatenation (that is, by defining subclasses) the user can build a library of pre-compiled processes.

Security

Protection against and detection of errors are built-in. For example, attempts to interfere with process synchronization are detected; that is, inappropriate process activation and deactivation is discovered and reported to the user.

Efficiency

Reasonable execution speed is important in the light of the well-known fact that system simulation is a notorious consumer of computer time.

Overall, attempts have been made to hold execution time down. In particular, integration and interpolation strategies are chosen with speed in mind.

SIMULA's dynamic storage allocation helps keep storage requirements down. Components enter and leave the system; only those currently present need be represented in the computer.

Portability

The last design objective is portability. Class DISCO (about 2100 lines of SIMULA 67 Common Base Language[5]) is completely machine-independent.

DISCO may be obtained at a nominal cost by writing to:

Department of Computer Science
Roskilde University Center
4000 Roskilde, Denmark
Attention: Keld Helsgaun

REFERENCES

1 BIRTWITSLTE, G. DAHL, O.J. MYHRHAUG, B. NYGAARD, K.
SIMULA BEGIN
Studentlitteratur Lund 1973

2 BLUNDEN, G.P. KRASNOW, H.S.
The Process Concept as a Basis for Simulation
Simulation vol. 9 no. 2 August 1967
pp. 89-93

3 CELLIER, C.F.
Combined Continuous Discrete Simulation by Use of Digital Computers: Techniques and Tools
PhD thesis Swiss Federal Institute of Technology Zurich 1979

4 CELLIER, C.F. BONGULIELMI, A.P.
The COSY Simulation Language
IMACS Congress on Simulation of Systems Sorrento September 24-28, 1979 pp. 271-281

5 DAHL, O.J. MYHRHAUG, B. NYGAARD, K.
SIMULA 67 Common Base Language
Norwegian Computing Center Publication S-22
Oslo 1971

6 ENGLAND, R.
Error Estimates for Runge-Kutta Type Solutions to Systems of Ordinary Differential Equations
Computer Journal vol. 12 May 1969 pp. 166-170

7 FAIRLAND, D.A.
Combined Discrete-Event/Continuous-Systems Simulation
Simulation vol. 14 no. 2 February 1970
pp. 61-72

8 FRANTA, W.R.
The Process View of Simulation
North-Holland New York 1977

9 GOLDEN, D.G. SHOEFFLER, J.D.
GSL—a Combined Continuous and Discrete Simulation Language
Simulation vol. 20 no. 1 January 1973
pp. 1-8

10 HELSGAUN, K.
COMBINEDSIMULATION—a SIMULA-Class for Combined Continuous and Discrete Simulation
Proceedings Sixth SIMULA Users' Conference
Lisbon 1978 pp. 30-35

11 HILLS, P.R.
An Introduction to Simulation Using SIMULA
Norwegian Computing Center Publication S-55
Oslo 1973

12 HOULE, P.A. FRANTA, W.R.
On the Structural Concepts of SIMULA and Simulation Modelling
Proceedings 1974 Summer Computer Simulation Conference Houston, Texas 1974 pp. 55-60

13 HURST, N.R. PRITSKER, A.A.B.
Simulation of a Chemical Reaction Process Using GASP IV
Simulation vol. 21 no. 3 September 1973
pp. 71-75

14 KNUTH, D.E. McNELEY, J.L.
SOL—a Symbolic Language for General-Purpose Systems Simulation
IEEE Transactions on Electronic Computation
vol. EC-13 no. 4 August 1964

15 MØLLER, O.
Quasi Double-Precision in Floating Point Addition
BIT vol. 5 pp. 17-50, 251-255

16 ÖREN, T.I.
Software for Simulation of Combined Continuous and Discrete Systems: a State-of-the-Art Review
Simulation vol. 28 no. 2 February 1977
pp. 33-45

17 PRITSKER, A.A.B. YOUNG, R.E.
Simulation with GASP-PL/I
Wiley New York 1975

18 PRITSKER, A.A.B. HURST, N.R.
GASP IV: a Combined Continuous-Discrete FORTRAN-Based Simulation Language
Simulation vol. 21 no. 3 September 1973
pp. 65-70

19 SIERENBERG, R.W.
Combined Discrete and Continuous Simulation with PROSIM
Proceedings Simulation '77 (M.H. Hamza, editor)

20 SIM, R.
CADSIM Users' Guide and Reference Manual
Imperial College Publication no. 75/23 London
1975

21 SPECHART, F.H. GREEN, W.L.
A Guide to Using CSMP—the Continuous System Modelling Program
Prentice Hall Englewood Cliffs, New Jersey
1976 pp. 35-39