CHAPTER 12.

A SIMPLE SIMULA DESCRIPTION

12.1 A Simple Disease System.

As a first example on a SIMULA system description, let us consider a simple disease system:

Infections occur in a limited population with a time di. tribution given by a Poisson distribution with paramete" "real p' Infected persons do not infect other persons, and they become immune if they recover from the disease.

When a person is infected, no symptoms appear the first 3 days. In the following 7 days (days 4 through 10) the probability of death is given by an array:

```
array mortality [4 : 10] .
```

As mentioned in section 1.4 every entity which carries out actions and/or is a carrier of data in SIMULA will be called a process. All processes characterized by the same data structure and having the same pattern of behaviour (operatior rule) are said to belong to the same activity.

Obviously we have to introduce the infected persons as processes. Since the course of development of the disease is the same for all persons, they all belong to one activity:

```
activity infected person;
```

Apart from the "main program" taking care of the initialization of the system, the introduction of new "infected persons" and final analysis of the results of the disease, we have no other entities executing actions or carrying data, and consequently no more activities.

We start our formal description of the system by writing

```
SIMULA begin                                    (Line 1)
```

telling that SIMULA concepts are to be used.

In the next line we declare the "system variables", variables which shall be available for use everywhere in the system: the size of the population (named "population"), the number of not yet infected persons left in the population (named "nr uninfected"), the parameter "p" of the Poisson disease occurence distribution, the table of real numbers giving the mortality on the n-th day of the disease (named mortality [n]), and finally two integer variables U1 and U2 used in selecting the streams of random numbers to be used in the random drawings performed (see section 7.1).

U1 and U2 are introduced because of the possible simulation of the system. They have no interpretation in terms of the system which is described.

The declarations of system variables then become:

```
integer population, nr uninfected, U1, U2;      (Line 2)
real p; array mortality [4 : 10];               (Line 3)
```

After the disease has disappeared we want to find out how many persons died and how many were cured. For this we need to establish two lists of references to these categories of persons. As mentioned in section 1.4 the SIMULA sets will

serve this purpose. Since all processes must be able to refer to these sets, they are declared as "system sets":

set dead, cured; (Line 4)

Notice that the set declarations introduce empty sets, whereas the values of variables are undefined till an assignment of a value is made.

We may now start the description of the "infected person" activity by writing:

activity infected person; (Line 5)

We have to keep track of how long the person has been infected, since this determines the mortality of the disease at any given stage. Hence the infected persons are characterized by an integer attribute "day":

begin integer day; (Line 6)

(The begin indicates that the description of the endogenous attributes and operation rule of the activity starts here).

When stating the operation rule for a SIMULA activity we are able to take a completely "local" view, and concentrate upon the sequence of actions which may be carried out by an individual process belonging to the activity. Since all actions are carried out by processes during active phases (events) and a process either belongs to an activity or is the unique "main program" process, all actions in the system will be described by giving the operation rules of the activities and the main program. The proper interlacing of events will be achieved by the sequencing statements (see CHAPTER 4).

During the first active phase of an infected person the following statement is executed:
The number of uninfected persons is reduced by 1, since a new infected person has become active.

nr uninfected := nr uninfected - 1; (Line 7)

No more actions are to be executed until 3 days have passed by. We shall "hold" the sequence of actions for 3 days, the process "suspends" itself for 3 days. This is described by the statement

```
hold(3);                                    (Line 8)
```

When the 3 days have expired, actions and decisions have to be made the following 7 days

```
for day := 4 step 1 until 10 do             (Line 9)
```

Each day there is a probability, mortality [day], that the person dies. If this happens, a reference to him is included in the set "dead" and all further actions are stopped, he is "terminated".

In other words there is a probability, mortality [day], that these actions occur. This is equivalent to making a random drawing with probability mortality [day] of getting true as result, and only execute the actions if this turns out to be the outcome. For this purpose we may use the "draw" - procedure described in section 7.2:

```
if draw(mortality [day], U1) then           (Line 10)
begin include(current,dead);
      terminate(current) end                (Line 11)
```

"current" always references the process which currently is active, that is, the process in which the reference is made. From the process' point of view, "current" means "myself".

If the result of the drawing is false, further actions are suspended till the next day:

```
else hold(1);                               (Line 12)
```

Then, if the previous day was not the 10th, "day" is increased by 1 and Lines 10-12 are repeated.

If the previous day was the 10th, we have completed the for - statement starting at Line 9, and we proceed to day nr. 11. Now the infected person has survived and is cured and immune, and he may include a reference to himself in the "cured" set:

include(current, cured) (Line 13)

This is the last action relevant to the disease performed by the infected person, and we may conclude the operation rule by

end; (Line 14)

The infected person has no more actions prescribed for himself. He is terminated, but still remains in the system as a data structure, since he is a member of a set.

Till now we have only made declarations: of system variables and sets, of data structures and operation rules of processes which may appear in the system. No actions have been executed.

The sequence of statements of the SIMULA block, following the declarations, is the operation rule of a unique process, the "main program", always present in a SIMULA system description. Since the first statement after the declarations always is the first statement executed within a block, the first active phase of the main program always is the first event within a discrete event system as described by SIMULA. It is bound to occur, and as first event it is assigned the system time zero.

If other processes are to appear, at least one has to be generated by the main program and the main program must have at least one inactive period if events pertaining to other processes shall be executed.

As the process having the "first event", the main program is used for setting up the initial state of the system. It may also, if desired, be used as a "monitoring" process and for final analysis of information collected during the operation (or simulation) of the system.

The first we have to do to set up the system is to read the parameter values from an input device:

```
read(population, U1,U2,p,mortality);    (Line 15)
```

Then the initial value of "nr uninfected" is set

```
nr uninfected := population;            (Line 16)
```

We now initiate the first "infected person"-process, and since we are to repeat this statement we mark it with a label:

```
infect: activate new infected person;   (Line 17)
```

This statement generates a new "infected person"-process by the "generative" expression

```
new infected person
```

and activates this process immediately. By "immediately" we mean that the main program suspends its own actions to allow the just generated "infected person"-process to execute its first active phase, described in Lines 5-8. This is called "direct scheduling" (see CHAPTER 4).

When this process suspends itself (Line 8), the main program immediately resumes its actions in a new active phase.

This active phase consists of only one statement, suspending the actions of the main program for the time interval between this and the next appearance of an infected person. The length of this interval has a negative exponential distribution, and we may write:

```
hold(negexp(p, U2));                    (Line 18)
```

If there are any more uninfected persons, we repeat the infection:

```
if nr uninfected >0 then go to infect;      (Line 19)
```

If all persons are infected, we proceed to the next statement which has to be

```
hold(11)                                    (Line 20)
```

since no analysis should be made till the last infected person has died or become cured. Then we may write out the results:

```
write(population,cardinal(dead),cardinal(cured))    (Line 21)
```

The procedure "cardinal" gives the number of references ("elements") in a set. This is the last statement, the SIMULA block and the SIMULA description is concluded by

```
end;                                        (Line 22)
```

The complete description becomes:

```
SIMULA begin                                                    (1)
integer population, nr uninfected, U1, U2;                      (2)
real p; array mortality [4 : 10];                               (3)
set dead, cured;                                                (4)
activity infected person;                                       (5)
begin integer day;                                              (6)
     nr uninfected := nr uninfected - 1;                        (7)
     hold(3);                                                   (8)
     for day := 4 step 1 until 10 do                            (9)
     if draw(mortality [day], U1) then                          (10)
     begin include(current,dead); terminate(current) end        (11)
     else hold(1);                                              (12)
     include(current,cured)                                     (13)
end;                                                            (14)
read(population, U1, U2, p, mortality);                         (15)
nr uninfected := population;                                    (16)
infect: activate new infected person;                           (17)
```

12.2 Details of the Element and Sequencing Procedures.

In this section we will discuss in detail the sequencing of events in simulations generated by SIMULA system descriptions by using the example in the last section.

All events which are scheduled, on which we have information, are represented by an "event notice" in a set, "the sequencing set" (SQS).

An event is determined by the system time at which the event is to occur and by the process which is to be active. The event notice must contain these two informations: a system time reference (TR) and a process reference. However, since all references to processes in SIMULA are indirect, through elements the event notice instead of a process reference contains an element reference (ER). The element referred to will in turn refer to the process.

Thus the format of an event notice is

(TR,ER).

The event notices in the SQS will at any time be ordered according to increasing value of the TR. The event notice lying in the front of the SQS refers to the currently active process, the "current" process. When the current event is completed, its event notice is removed. The event notice which occupied the second position now becomes the first, and the next active phase of its (indirectly) associated process becomes the new "current event".

Since the main program is the only process existing initially and it is bound to have an active phase, the initial contents of the SQS is an event notice referring an element which in turn refers the main program. Let us name the element E(MP). The system time is in SIMULA set equal to zero at the start of

system operation. The contents of the SQS then is

(0, E(MP))

This event causes the 3 first statements of the main program to be executed (Lines 15-17).

In Line 17, the expression "new infected person" creates a new process, belonging to the activity "infected person". Let us name this process P1 in our discussion.

The reactivation point (RP) is placed behind the declarations of attributes. This implies that P1 when activated will start its first active phase by executing the first statement of its operation rule. (See CHAPTER 2). During the active phase the RP is not defined. When it is completed, the RP is placed behind the statement concluding the active phase, thus defining the action whereby a new active phase of this process shall start.

Together with P1 an element referring to P1 is generated. Let us name this element E(P1).

As described in CHAPTER 3, an element contains 3 references. One reference specifies a process, the two others serve to specify a set membership, if any. Since the SIMULA sets are ordered and cyclical, these two references specify a successor and a predecessor in the set. The format of an element then is

(successor, predecessor, process)

The element of the main program contains

(none, none, main program),

and the element of P1 contains

(none, none, P1)

In this discussion, the SQS will be ordered as follows:

SQS
E1
E2
E3
:
:
En

E1 is the current event, at the front of SQS. E2 is the next event to E1 etc.

The expression "new infected person" is preceded by the word activate. This implies that P1 shall have its first active phase at the current system time and with priority before all other events, even the current one. ("Direct scheduling": "activate E" is equivalent to "activate E delay 0 prior". See CHAPTER 4).

To achieve this the current event is not cancelled, it only suspends itself by placing the event notice generated by "activate new infected person" in front of itself.

"activate new infected person" then generates an event notice

(0, E(P1)),

and the contents of the SQS becomes

SQS:
(0, E(P1)),
(0, E(MP)).

The consequence is, according to the rules stated, that the first active phase of P1 becomes the current event, and P1 executes the first statements of its operation rule (Lines 7-8). ("new infected person" already has had the effect of generating the process and its attribute "integer day").

hold(3) in Line 8 concludes the first active phase of P1, and at the same time schedules a next active phase for itself to occur 3 time units later, it "suspends" itself.
("hold(3)" is equivalent to "reactivate current delay 3")

This scheduling is done by generation of an event notice (0 + 3, E(P1), which is inserted at its proper place in the SQS. Since the SQS is ordered according to increasing TR, the SQS becomes

SQS:
(0, E(MP)),
(3, E(P1)).

The event notice (0, E(P1)) has disappeared because its associated event is completed.

The main program once more becomes the current process, and the system time still is zero. The only action in this event is "hold(negexp(p,U2))" (Line 18). The main program suspends itself. Let us assume that the random drawing "negexp(p,U2)" gives 2.5 as its result. The hold-statement then passivates the main program and generates the event notice (0 + 2.5, E(MP)). We get

SQS:
(2.5, E(MP))
(3 , E(P1)).

Once more the main program becomes active. The system time is stepped forward to the TR of the current event, 2.5, and the statement of Line 19 is executed. We assumed that the size of the population is large, and we will then execute Line 17 once more: the main program creates a new event notice referring to a new element E(P2) which refers to the new process P2 (generated by the execution of "new infected person"), and suspends itself.

The SQS becomes

SQS:

(2.5, E(P2))
(2.5, E(MP))
(3 , E(p1))

P2 executes Lines 7-8 of its operation rule (which it shares with P1 and all other "infected person"-processes to be generated) and generates the event notece (2.5 + 3, E(P2)) which gives the new contents of the SQS:

(2.5, E(MP))
(3 , E(P1))
(5.5, E(P2))

The main program now executes the hold-statement. Let us assume that the result of the random drawing is 2, which gives

SQS:

(3 , E(P1))
(4.5, E(MP))
(5.5, E(P2))

P1 now gets its second active phase. The reactivation point of P1 indicates that the for-statement of Line 9 now shall be started: "day" is put equal to 4 and the random drawing of Line 10 is executed. Let us assume that the outcome is "false". Then "hold(1)" is executed and the SQS becomes

(4 , E(P1))
(4.5, E(MP))
(5.5, E(P2))

P1 gets its 3rd active phase : "day"is increased by 1 to 5 and once more a random drawing is performed. Let us assume that the outcome this time is "true". Then the statement "include(current, dead)" shall be executed. "current" now is P1, or rather E(P1), and the set membership of the element must be modified:

E(P1) shall be included as the last element of the set "dead". This set has til now been empty: it has only contained its "set head", which has been its own successor and

predecessor:

head(dead): (head(dead), head(dead), "no process")
(as always the process reference of a set head is to "no process")

E(P1) shall become the last element of "dead":
head(dead) shall become its successor element. Since there are as yet no other elements, the elements of "dead" after the inclusion are

head(dead) : (E(P1),E(P1), "no process")
E(P1) : (head(dead),head(dead), P1)

The statement "terminate(current)" ends the active phase of P1 without leaving a reactivation point or an event notice. P1 is terminated, it only exist in the system as an element of the set "dead" and a carrier of the variable "day", having the value 5.

We have

SQS:

(4.5, E(MP))
(5.5, E(P2))

The main program generates P3, E(P3) and the directly scheduled event notice (4.5, E(P3)):

SQS:

(4.5, E(P3))
(4.5, E(MP))
(5.5, E(P2))

The first active phase of P3 gives

SQS

(4.5, E(MP))
(5.5, E(P2))
(7.5, E(P3))

Let us assume that the random drawing in Line 17 gives 1.5 as result:

SQS:

(5.5, E(p2))
(6 , E(MP))
(7.5, E(P3))

Now P2 enters its 2nd active phase at the for-statement at system time equal to 5.5. Let us assume that the random drawing already at day=4 gives true as outcome. Then E(P2) shall be inserted as the last element of "dead". This implies that it gets head(dead) as its successor and E(P1) as its predecessor. The references of the other elements are modified accordingly, and the set becomes:

head(dead) : (E(P1), E(P2), "no process")
E(P1) : (E(P2), head(dead), P1)
E(P2) : (head(dead), E(P1), P2)

Then P2 is terminated and only exist in the system as an element of "dead" and a carrier of the information "day=4".

In this way, the sequencing is continued. Before we leave the example, let us consider some other situations.

Let us assume that

SQS:

(15 , E(MP))
(16 , E(Pn))
(17.8, E(Pn+1))
.
.
.
.

and that the main program executes its hold-statement giving the event notice (16, E(MP)). According to the definition of the hold-statement, the generated event notice has no priority. Therefore it is inserted after all other event notices with TR = 16 already in the SQS, which becomes

SQS:

(16 , E(Pn))
(16 , E(MP))
(17.8, E(Pn+1))
⋮

If we in Line 18 instead had written

reactivate current delay negexp(p,U2)prior

the event notice would have had priority over other event notices with the same TR:

SQS:

(16 , E(MP))
(16 , E(Pn))
(17.8, E(Pn+1))

Notice that we have to write "reactivate" in this case: the main program is already active and therefore "activate" will have no effect.

"reactivate E ----" removes the existing event notice, if any, and substitutes the new one generated by the statement.

Let us consider the situation when an "infected person"-process, Pn, has survived the 10th day. He executes the final hold(1)-statement, and in his next active phase the statement "include (current, cured)" is executed. This inclusion follows the same rules as described for the "dead" set above. No terminate-statement is specified, but the process is automatically terminated since no other statement follows the include-statement. The local "sequence control" of the process' actions has "left through the final end" of the operation rule.

When all members of the population are infected, the statement hold(11) (Line 20) of the main program is executed. The resulting event notice will have a larger TR than any other EN already generated or to be generated in the system operation.

This implies that this event notice will become and remain the last one in the SQS. When it finally is the only one not executed, the main program executes its last active phase, Line 21, and the main program becomes terminated, control leaves through its final end.

Now the SQS is empty, and no more actions will occur in the system. If the SIMULA description is an inner block in an ALGOL program, control will proceed to the next statement in the outer block. All entities declared and generated within the SIMULA block will disappear. However, variables declared in the outer block may have been operated upon within the SIMULA block, which may in this way transmit information to the outer block.

12.3 Details of Scanning and Connection.

As stated above, the "infected person"-processes will end up as terminated members of the sets "dead" or "cured". For those who are elements of "dead", the integer attribute "day" will specify at which day after infection the person died. This we may want to use to establish a histogram of the days of death. Let "nr dead [4 : 10]" be an integer array, declared in the SIMULA block head.

We may perform the calculations inside the operation rule of the "infected person" activity, but we will now show how we may use the connection facilities of SIMULA for this purpose (see CHAPTER 5).

All the "dead" processes are members of the "dead" set. Each of these processes carry the relevant information in the attribute "day". If we want to compute the statistics during the last active phase of the main program, the use of the identifier "day" has no meaning: "day" is not declared in the SIMULA block head, and we have as many values of "day" as we have processes in the system.

In order to get all these values available, one at a time, we must be able to refer to each individual process, and make its attribute available to be operated upon.

If E is an element referring to a process P belonging to the activity A, and S is a statement, the statement

inspect E when A do S;

will "connect" the process P so that its attributes are available in the statement S. There is no ambiguity now, since we have specified an individual process through the element E, and it is the individual values of the attributes of this process P we get access to.

To use this device, we also must be able to let E refer to all members of the set "dead" in succession, we must "scan" the set. This is done by using an "element variable", a variable taking on specific elements as values. Let us call this variable "pointer" and declare it in the SIMULA block head by

element pointer;

We start the scanning by letting "pointer" refer to the element head(dead):

pointer: = head(dead);

The scanning is achieved by repeated substitution of "pointer" by its own successor:

pointer: = suc(pointer)

After the first substitution "pointer" will have the "first" element of "dead" as its value, after the second substitution the second element etc.

Since the SIMULA sets are cyclic, such a sequence of substitutions would never stop. Therefore we have after each substitution to test whether the element now referred to by "pointer" itself refers to a process. For this purpose we use the Boolean procedure "exist". exist(pointer) will have the value true if pointer refers to a process through its element value, false if its element value has no process reference.

Since only the head of a set has no process reference, we may continue the substitution as long as exist(pointer) has the value true.

After each substitution we connect the process referred to and use its "day" attribute for calculations.

The lay out of the SIMULA program becomes:

element pointer; and integer array nr dead [4 : 10];

are declared in the SIMULA block head.

After the statement hold(11) (Line 20) we give all variables in nr dead zero as initial value (k is an integer variable also declared in the SIMULA block head):

```
for k: = 4 step 1 until 10 do nr dead [k]: = 0;
```

Then pointer is given its initial value

```
pointer : = head(dead);
```

and the scanning, connection and calculation is done by

```
for pointer := suc(pointer)while exist (pointer)do
inspect pointer when infected person do
nr dead [day]: = nr dead [day] + 1;
```

(The last statement simply increases by 1 the number of dead for the day of death of the person connected).

As last statement we may specify a printout:

```
write (population, nr dead, cardinal(cured));
```

which will print all components of the array.

The "inspect"-statement will not affect the set membership of the elements referred to. Another connection statement

```
extract E when A do S;
```

will connect the process referred to by E, but at the same time remove E from the set to which it belongs, if any.

The "extract" statement may also be used in our case, since it is irrelevant whether the set membership is kept or not. However, the substitution procedure must be a different one, since suc(pointer) is none after "pointer" has been extracted from the set "dead".

When "pointer" has been extracted and the necessary calculations made, we want to get the next process which now, after the previous extraction, is the first element of the set. In this case we may omit the statement "pointer = head(dead)"; and instead write

```
for pointer: = first(dead) while exist(pointer)do
    extract pointer when infected person do
    nr dead [day]: = nr dead [day] + 1;
```

Notice that the element extracted looses its only reference in the system, through the set "dead", by the extraction. On the other hand it gets another reference added through the connection. After the connection statement is completed this reference vanishes. No other reference is created in the connection statement, it is not possible to refer to the element any more: it disappears from the system. Since this element is the only one referring its process, the process also disappears.

batch of units of a given order waiting in the queue, the next order is tried. The last units of an order are accepted as a batch, even if the number is less than the ordinary batch size. If a machine finds no acceptable batch in the product queue, it will wait until more units arrive.

Although the individual pieces of product are "units", a unit will not be treated as an individual item in the present model. For a given order and a given step in its schedule, i.e. machine group, we define an opart (order part) record to represent the group of units currently involved in that step. The units are either in processing or waiting to be processed the corresponding machine group.

An order is represented by a collection of opart records. The sum of units in each opart is equal to the number of units in the order. Each opart is a member of a product queue. If a machine group occurs more than once in the schedule of a product type, there may be more than one opart of the same order in the product queue of that machine group.

Among the attributes of an opart record are the following integers: The order number, ono, the product type, the step, the number of units waiting, nw, and the number of units in processing, np. The flow of units in the system is effected by counting up and down the attributes nw and np of opart records.

An opart record is generated at the time when the first batch of units of an order arrive at a machine group. It is entered at the end of the corresponding product queue. The opart will remain member of this queue till the last unit has entered processing. It will drop out of the system when the last unit has finished processing. A Boolean attribute last is needed to specify whether a given opart contains the last units of the order involved in this step.

At a given time the units of an order may be distributed on several machine groups. There will be an opart record for each