CHAPTER 5.

CONNECTION

5.1 Connection Statements.

The connection mechanism of SIMULA provides a means of inter-
action between processes. A process can, by connecting another
one, get access to the attributes of the latter.

A connection statement has the following general form.

$$\text{inspect } X \text{ when } A_1 \text{ do } S_1$$
$$\text{when } A_2 \text{ do } S_2$$

$$\bullet$$
$$\bullet$$
$$\bullet$$

$$\text{when } A_n \text{ do } S_n$$
$$\text{otherwise } S;$$

where A_1, A_2, ..., A_n are activity identifiers, and S_1, S_2,
..., S_n, and S are statements. A construction "when A_1 do S_1"
is called a connection clause. There can be any number of
them and must be at least one. The otherwise clause is optio-
nal.

If the PA of X is a process belonging to class A_1, the state-
ment S_1 is executed, and the other statements are ignored.
The connection is said to be effective during the execution
of S_1. The value of the element expression X becomes the
connected element, and the referenced process becomes the con-
nected process. If the process does not belong to any of the
classes listed, or if X has no PA, the statements S_1, ..., S_n
are skipped and the statement S is executed, if present.

Each statement S_i is a <u>connection block</u>. It is interpreted
as if it were a part of the outermost block of the activity
A_i, in the sense that the exogenous and endogenous attributes
of the connected process are immediately accessible through
their local names. The block containing the connection state-
ment acts as an outer block.
Another connection statement is

<center><u>extract</u> X <u>when</u> ... etc</center>

It has the same general form as the one above, and also the
same effect, except that the element X is removed from its
set, if it has a SM and a PA.

If the expression X is a process designator, the class of the
process is known. In this case the alternative connection
clauses can be replaced by the single construction "<u>do</u> S",
where S is a statement serving as a connection block. No
<u>otherwise</u> clause applies in this case.

<u>activate</u> and <u>reactivate</u> statements can be augmented by connec-
tion clauses. If the scheduling is <u>direct</u>, the indicated
active phase is executed <u>before</u> connection becomes effective,
otherwise after, possibly while connection is effective.
Connection may become effective even if no event is scheduled.

Within a connection block connecting a class A process the activity
identifier A, not preceded by the symbol "<u>new</u>" or by the symbol
"<u>when</u>", has the significance of a function designator referencing
the <u>connected element</u>. The same is true for a reference to A
within the body of a procedure declared local to A, when the
procedure is called within the connection block.

Example.

      activate new car do if speed > 50 then include(car, left lane)
                    else include(car, right lane);

where "speed" is an endogenous attribute of the car process,
whose value is defined during the first active phase of the
process.

Because of its use in connection statements an activity identi-
fier may not be represented by a formal parameter. Activity
identifiers are under no circumstances permitted as parameters
to procedures or activities. There is no "activity" specifier
in the language.

5.2 Label Attributes.

The fact that labels, switches and procedures local to the
outermost block of the specified activity body are accessible
within a connection block, makes it possible to "enter" a
connected process. This can happen as the result of an ex-
plicit go to statement within the connection block, or as a
side-effect of a procedure local to the activity called within
the connection block.

Let L be a label local to the connected process. Then "go to L"
has the following effects:

1.   The connected process is terminated, i.e. any event notice
     referring to it is removed from the SQS, any reactivation
     point for the process is deleted.

2.   The current process is terminated without removal of the
     current event notice. Connection is thereby cancelled.

3. The element reference of the current event notice is re-
   placed by a reference to the formerly connected element,
   which thereby becomes "current".

4. An active phase of the formerly connected process commences
   at the specified label L. The system time remains un-
   changed.

By this means the reactivation point of a currently passive
or suspended process can be superceded and a terminated process
can be "revived".

5.3 Examples.

1. Procedure Attributes.

   Cars are traveling on a road. Each car is characterized
   by its velocity V and its position X. The former is a
   step function and the latter a continuous function of time.

   activity car;
   begin real V, Xo, To;
           real procedure X; X := Xo + V$\times$(time - To);
           procedure update (Vnew); real Vnew;
           begin Xo := X; To := time; V := Vnew end;
   ----
   end;

   To is the time when V was last updated, and Xo is the
   position of the car at that time.

   A regular police survey tries to enforce an upper speed
   limit Vmax on a bad portion of the road, between X1 and X2.

```
set road, police file; real X1, X2, Vmax;

activity survey (interval); real interval;
begin element Z;
scan: for Z := first(road), suc(Z) while exist(Z) do
      inspect Z when car do
      if X1 < X' ∧ X < X2 ∧ V > Vmax then
      begin update(Vmax); include(car, police file) end;
      hold(interval); go to scan
end;
```

Notice that the procedures "X" and "update" referenced within
the connection block are those declared for the currently
connected car. The non-local items referenced within the
bodies of these procedures are therefore the attributes of
this particular car.

2. List Processing.

Let T be a set, some of whose elements are "branch" pro-
cesses.

```
set T;
activity branch; begin set subtree;----- end;
```

T can be regarded as a tree structure if elements of a
subtree may be branch processes. The "leaves" of the
tree can be defined as those elements of T or of a subtree
which are not branch processes.

A tree structure like this can be scanned "leafwise" by
means of a recursive procedure. The following activity
is equivalent to a "reader" concept found in well known
list processing languages. It incorporates a recursive
scan procedure.

```
activity leafscan (tree); set tree;
begin element leaf;
        procedure scan(S); set S;
        begin element X;X := head(S);
            for X := suc(X) while exist(X) do
```

```
        inspect X when branch do scan(subtree)
             otherwise begin leaf: = X; passivate end
        end scan;
             scan(tree); leaf: = none
  end leafscan;
```

A reader on T can be initialized by the following statements.

```
    Tsc := new leafscan(T); activate Tsc;
```

where Tsc is an element variable. Later activate statements
on Tsc will step the pointer, which is the endogenous attribute
"leaf" of the leafscan process. When the elements are ex-
hausted, leaf becomes none.

Access to the pointer is by connecting the element Tsc, ex-
plicitly or through one of the following procedures:

```
element procedure reader(Z); element Z;
inspect Z when leafscan do reader := leaf;
element procedure next(Z); element Z;
activate Z when leafscan do next := leaf;
```

The expressions reader(Tsc) and next(Tsc) both evaluate to
the element of the tree T presently under observation. The
latter has the side effect of stepping the pointer.

Notice that the pointer is stepped before the element value is
assigned, since the activation is by direct scheduling.

In general list structures of processes can be formed by means
of set or element attributes. The user has complete freedom
when defining the structures of the various components of a list.

The list processing facilities of SIMULA can be exploited for their own sake with no reference to the discrete event system concept. In such cases it may be natural to define the processes on a list as passive data carriers.

3. Sorting.

Given a set, file1, containing references to "record" processes (and possibly processes of other classes). We want to establish another set, file2, of references to these record processes, sorted against a real attribute "key" with nonnegative values.

```
set file1, file2;
activity record; begin real key; --- end;
```

The following piece of program performs the sorting by means of the SQS. It is assumed that all record processes in the system are passive.

file2 is empty initially.

```
begin element X,Y;
for X := first(file1), suc(X) while exist(X) do
     inspect X when record do activate X delay key;
X := current; Y := none;
for X := nextev(X) while exist(X) do
     inspect X when record do
     begin cancel(Y); Y := X; include(X, file2) end;
cancel(Y) end;
```

The latter inspection clause only serves to skip processes referenced in the SQS not belonging to the "record" activity.