SOME FEATURES OF THE SIMULA 67 LANGUAGE

by

Ole-Johan Dahl, Bjørn Myhrhaug and Kristen Nygaard

Norwegian Computing Center
Oslo, Sept. 1968

SIMULA 67 is a general purpose programming language with a built-in simulation capability similar to, but stronger than that of SIMULA I. SIMULA 67 has been developed by the authors at the Norwegian Computing Center. Compilers for this language are now being implemented on a number of different computers. Other compilers are in the planning stage.

A main characteristic of SIMULA 67 is that it is easily structured towards specialized problem areas, and hence will be used as a basis for special application languages.

SIMULA 67 contains the general algorithmic language ALGOL 60 as a subset, except for some very minor revisions. The reason for choosing ALGOL 60 as starting point was that its basic structure lent itself for extension. It was felt that it would be impractical for the users to base SIMULA 67 on still another new algorithmic language, and ALGOL 60 has already a user basis, mainly in Europe and the Soviet.

The SIMULA I language (1), (2), was conceived strictly as a simulation language or, rather, as a language providing concepts and statements allowing a complete and precise description of discrete-event systems.

In SIMULA I the world is regarded as a collection of programs, interacting and existing in parallel. Instead of splitting e.g. a "machine" into one piece describing the data structure of the "machines" and several other pieces describing various kind of actions executed by these "machines", the SIMULA I approach is to unify all these aspects of the "machines" into one "declaration", called an "activity declaration", (a "process class declaration in SIMULA 67).

SIMULA I has been used in a wide range of problem areas since the beginning of 1965, (3). The SIMULA 67 work started as an effort to improve SIMULA I as a simulation language, but gradually turned into a general purpose programming language and finally also into a language for generating problem-oriented languages. The language is described in (4).

The limited space available does not permit an exposition of the ALGOL 60 features used in this paper. Readers are referred to textbooks and the very short presentation given in (1).

The central concept in SIMULA 67 is the "object". An object is a self-contained program, (block instance), having its own local data and actions defined by a "class declaration". The class declaration defines a program (data and action) pattern, and objects conforming to that pattern are said to "belong to the same class".

If no actions are specified in the class declaration, a class of pure data structures is defined.

Example

```
class order (number); integer number;
   begin integer number of units;
      arrival date;
         real processing time end;
```

A new object belonging to the class "order" is generated by an expression such as

```
"new order (103); "
```

and as many "orders" may be introduced as wanted by such expressions.

The need for manipulating objects and relating objects to each other, makes it necessary to introduce list processing facilities.

A class may be used as "prefix" to another class declaration, thereby building the properties defined by the prefix into the objects defined by the new class declaration.

Examples

```
order class batch order;
   begin integer batch size; end;
order class single order;
   begin real finishing time, weight; end;

single order class plate;
   begin real length, width; end;
```

New objects belonging to the "sub-classes" - "batch order""single order" and "plate" all have the data defined for "order", plus the additional data defined in the various class declarations. Objects belonging to the class "plate" will e.g. consist of the following pieces of information: "number", "number of units", "arrival date", "processing time", "finishing time", "weight", "length" and "width".

If actions are defined in a class declaration, actions conforming to this pattern may be executed by all objects belonging to that class. The actions belonging to one object, may all be executed in sequence, as for a procedure. But these actions may also be executed as a series of separate subsequences, or "active phases". Between two active phases belonging to a given object, any number of active phases belonging to other objects may occur.

The basis SIMULA 67 language contains the fundamental statements necessary for organizing the total program execution as a sequence of active phases belonging to objects. However, in special applications special sequencing principles should be made easily available to the user.

SIMULA 67 may be oriented towards a special application area by defining a suitable class containing the necessary problem oriented concepts. This class is then used as prefix to the program by user interested in the problem in question.

The unsophisticated user may restrict himself to using the aggregated, problem oriented and familiar concepts as constituent "building blocks" in his programming. He may not need to know the full SIMULA 67 language, whereas the experienced programmer at the same time has the general language available, and he may extend the "application language" by new concepts defined by himself.

As an example, in discrete event system simulation the concept of "simulated system time" is the principle commonly used. SIMULA 67 is turned into a simulation language closely resembling SIMULA I by providing the class "SIMULATION" as a part of the language, (in this case provided with the compilers). In the class declaration

```
class SIMULATION;
    begin ..... end;
```

a"time axis" is defined, as well as two-way lists ("sets") which may serve as queues and files, and also the class "process" which gives an object the property of having its active phases organized through the "time axis".

A user wanting to write a simulation program starts his program by

```
SIMULATION begin .....
```

in order to make all the simulation capabilities available in his program.

The SIMULA "processes" are objects which are described by declarations uniting their

- data aspect, through parameters and local declarations
- action aspects, through the description of all their active phases by statements in the class body
- interactions in "system time" with other processes, through the prefix "process"
- other properties, through locally declared procedures.

Example

In SIMULA 67 names may be assigned to individual objects through "references variables", declared by ref (class identifier). "Sets" (circular two-way lists) are introduced by "head" objects which serve as starting and end points of these lists. Subroutines are in ALGOL 60 given by "procedure declarations".

The description below gives an (incomplete) illustration of how a simple "machine" in a job shop simulation may be described.

```
Process class machine (inq. outq. own crane);
        ref(head)inq. outq;ref(crane) own crane;
        virtual: procedure service:
        begin
        ref (order) served; real setup time;
        procedure service;
            hold (setup time + served.processing
            time);
work:   if inq. empty then passivate
        else begin served: - inq. first;
        served. out; service; served. into(outq);
        activate own crane delay message time end;
        go to work
        end machine;
```

Since the procedures "service" is specified as a "virtual quantity", it may be redefined in subclasses to "machines".

Example

```
machine class type A;
        begin procedure service;
              hold(negexp(served.processing time));
        end type A;
```

A process belonging to "type A" will have data and operate like a "machine"process, except that "service" will be as defined within "type A".

If a user wants to generate a special-purpose simulation language to be used in job-shop analysis, he may write:

```
SIMULATION class JOBSHOP;
     begin ..... end;
```

Between "begin" and "end" he defines the building blocks he needs", like

```
process class crane;
        begin ..... end;

process class machine;
        begin
        .....
        end;
etc.
```

The programmer now compiles this class, and whenever he or his colleagues want to use SIMULA 67 for jobshop simulation, they may write in their program

```
JOBSHOP begin .....
```

thereby making available the concepts of both "SIMULATION" and "JOBSHOP", and program in terms of "cranes", "machines" etc.

This facility requires that a mechanism for the incorporation of separately compiled classes is available in the compiler.

SIMULA 67 contains the new types "character" and "text", the latter to provide the desired flexibility in string handling. Input/output is also defined in SIMULA 67, in contrast to ALGOL 60 which has been seriously affected by lack of standardization in these areas.

References

(1) "SIMULA - A language for programming and description of discrete event systems. Introduction and user's manual"
by O.J. Dahl and K. Nygaard.
NCC, Sept. 1967

(2) "SIMULA - an ALGOL - Based Simulation Language"
by O.J. Dahl and K. Nygaard.
Comm. of the ACM, Vol. 9, Sept. 1966

(3) "Users Experience with the SIMULA Language"
by H. Hegna, O.J. Lund and K. Nygaard.
NCC, June 1968

(4) "SIMULA 67 Common Base Language"
by O.J. Dahl, B. Myhrhaug and K. Nygaard.
NCC, May 1968