Ole-Johan Dahl

The Birth of Object Orientation: the Simula Languages
The Simula project

The project was initiated in 1962 by Kristen Nygaard, who saw the need for a language for simulation modelling.

From 1963 he and I cooperated very closely on language development. Implementation issues were my responsibility and issues of economy and organisation were KN's. The work took place at the Norwegian Computing Center (NCC).

General purpose algorithmic capability would be needed. Strategic considerations told that our language must be based on a standard one. Algol 60 was chosen.

There were three stages of language development, here identified as:

- Simula 0 (1962-63),
- Simula 1 (1963-65), and
- Simula 67 (1966-67).
Algol 60

- Has block structure.
- Procedures are specialised blocks.
- Is recursive, textually and dynamically.
- Is orthogonal and conceptually economical.
- There is compiler guaranteed and run time efficient access security to declared quantities, including specified parameters.
- It was well known in Europe.
Simula 0, 1962

The conceptual framework was mainly due to KN: a model would consist of „customers“ flowing through a fixed network of service „stations“.

Since both customers and stations had a superficial similarity to Algol blocks, the initial plan was to exploit that fact through a preprocessor to an Algol compiler.

Unfortunately the strict LIFO structure of Algol program executions is unsuitable for simulation models and could not be circumvented. For that reason the approach was abandoned.
Simula 1, 1963-65

The project was funded by UNIVAC, also providing a 1107 computer for the NCC at half price.

This was a result of negotiations by KN.

The work was to take place at the NCC.
Simula 1, 1963

In order to escape the Algol straitjacket a new runtime system had to be developed which catered for non-LIFO storage management.

Then extension for our purposes of Algol itself became possible.
Simula 1, 1963-64

The concepts of „customers“ and „stations“ were unified as „processes“, able to operate in quasi-parallel over system time.

Processes were referenced by pointers (indirect through elements of circular lists).

They would admit access from outside to their „attributes“, i.e. quantities declared in the outermost block.
Simula 1, 1963-64

For programming security the use of pointers had to be catered for by reference counts, later supplemented by a garbage collector.

Secure attribute access through unqualified pointers necessitated compiler conscious run time testing:

\begin{verbatim}
inspect <process reference> when <process type> do
begin <attributes accessible> end
\end{verbatim}
The Algol language was extended by the following ad hoc mechanisms:

- Procedure-like activity declarations giving rise to quasi-parallel processes
- (only parameters called by value were allowed), \textit{time}, a function procedure returning the system time,
- mechanisms for the explicit scheduling of processes in system time,
- attribute accessing by inspect, and
- a built-in concept of circular lists containing process pointers.
Simula 1, 1964

The new mechanisms were only available within a block prefixed by the keyword **SIMULA**. Thereby backward compatibility to Algol 60 was established.

Programming security was improved by pre-initializing all variables. That was mandatory for pointer variables.
Simula 1, 1965

The following skeleton example is found in the language documentation:

**SIMULA begin**

**activity car;** .. traveling on a linear road

**begin real X0; T0; V ;** .. position at given time, velocity

**real procedure X;** .. the position now

\[ X := X0 + V \times (time - T0); \]

**procedure UpdateV (V new); real V new;** .. change velocity

**begin X0 := X; T0 := time; V := V new end;**

<car behaviour> **end of car;**

**activity police;**

**begin ...; inspect <process reference> when car do**

**if X <within city> and V >50 then UpdateV (50); ...**

**end of police;**

... ... **end of simulation model;**
Simula 1, 1965

The *Car* example shows that the idea of „objects“ in the modern sense was established already in Simula 1: containing variables \((X0; T0; V)\), as well as associated procedures \((X; newV)\).

A comment in the language manual notes that the variable attributes unfortunately could not be hidden from outside view in a subblock, because the procedures too would lose access.
Simula 1, 1965-1966

Our experience with customer applications showed that Simula 1 was indeed a useful tool for simulation modelling. But there was frustration also:

- A simpler class concept of objects not dependent on simulation oriented mechanisms (but able to operate like coroutines) could be important in a general purpose setting.
- The same would hold for list processing facilities based on simple object pointers.
- The inspect mechanism was clumsy at times.
- C.A.R. Hoare had proposed (in an Algol Bulletin) a scheme for “record handling” allowing direct access to record attributes based on pointers qualified by record class, and possibly by record subclasses.
- The Simula 1 compiler was based on an Algol compiler for the UNIVAC 1107 far from optimal for our kind of extensions.
Simula 67, 1966

A new project for the development of an improved version of Simula was established, funded by the NCC.

A condition was that the project must prove profitable within the typical life time of a programming language, estimated to 3 to 5 years.

A new Algol/Simula compiler was planned in cooperation with The Technical University of Trondheim.
Simula 67, 1966

A main problem for us was how to adapt the class/subclass concepts of Hoare for our purposes. The breakthrough came in January '67, just in time for writing our paper for the forthcoming IFIP Conference on Simulation Languages at Lysebu, Oslo, May '67 („Class and Subclass Declarations“, in Ed.: J. Buxton: Simulation Languages, North Holland Publ, 1968).

A mechanism of class „prefixing“ was defined, today referred to as „inheritance“. The prefix of a class was itself a class, which could be separately compiled and reused.
The reuse of separately compiled classes invites to use the language for bottom up program design.

A last addition to our paper was the idea of „virtual procedure attributes“, whose bodies could be (re-) defined in subclasses, and thereby to some extent support top down design.

At the same time virtual procedures could be seen as substitutes for procedure parameters to classes.
Orthogonality dictated that class declarations should be legal in any kind of block head. So-called application languages could now be defined on top of Simula, in the form of a class intended as a prefix to an ordinary block. Such a class would typically contain locally declared classes.

An example is the predefined class `SIMULATION` containing a class `process` and other mechanisms corresponding to those of Simula 1. In particular the declarator activity would now be replaced by `process` class.

One restriction to orthogonality turned out to be necessary for reasons of principle and practice: Prefixing across block levels had to be forbidden. As a reasonable consequence `process class` declarations were confined to the outermost block level of a `SIMULATION` block.
The simulation application language (fragment)

class SIMULATION;
begin class process; .. process-objects operate in simulated time
begin real evtime; ref(process) nextev : - none;
...... end of process;
ref(process) current; .. pointer to the first in time list
real procedure time; time := current.evtime;
procedure hold(DeltaT); real DeltaT; .. suspend for the time DeltaT
inspect current do
begin evtime := evtime+DeltaT;
if evtime nextev.evtime then
begin ref(process)X :-current; current :-X.nextev;
<insert X at new position in the time list>;
resume(current) end
end of hold;
...... end of SIMULATION
Example fragment in Simula 67

```
SIMULATION begin
  process class Car; .. car processes:
  begin real X0; T0; V ; .. position at time T0, velocity
      real procedure X; X := X0+V (time-T0) .. position now
  procedure UpdateV (real V new); .. new velocity
      begin X0 := X; T0 := time; V ; = V new end;
      ...... ; UpdateV (80); hold(<travel time>); ......
  end of Car;
  process class Police;
  begin ref(process) P;
      ...... ;
      inspect P when Car do
          if X <within city>andV >50 then
              begin UpdateV (50); <give fine> end
      ...... end of Police;
  ...... end of SIMULATION block
```
Loose ends

We proposed to add class-like types to the language, whose variables would be permanently named (in-line) objects. The proposal was turned down by the „Common Base Conference“ which took place a few weeks after the Lysebu Conference (for fear of overburdening planned implementation teams).

Three different mechanisms for text handling and I/O were developed by Bjørn Myhrhaug and presented to the Simula Standards Group some months after. The one chosen would have been definable as a class-like type.

A small hierarchy of file classes was also added.

After some time mechanisms for hiding object attributes were propose by Jacob Palme (FOA, Stockholm), and adopted by the SSG.
Language Distribution

Simula 67 compilers were completed during the period 1969 to 1971.

Simula 67 was well received for simulation purposes, but its distribution was hampered by the pricing policy of the government agency behind the NCC.

Lectures at NATO Summer Schools, as well as a chapter in a book* contributed to promoting it as a general purpose programming tool.

Cultural Impact 1

The impact of Simula 67 can be illustrated by the universal acceptance of “object orientation”, OO, as an important paradigm for system development.

Thus, conferences on the theory and practice of OO are held regularly.

OO typically includes some or all of the ideas introduced in Simula 67.
Cultural Impact 2

The most important new idea of Simula is surely that of an “object“, encapsulating a data structure and a set of associated operator procedures (“methods“).

Except in trivial cases the latter will respect a local invariant restricting the state space of the object.

Our early examples indicate an awareness, often vague and never explicit, that an object as presented outward through its operators can be essentially different from its implementation.
Cultural Impact 3

It was Tony Hoare who showed how the different aspects of an object can be explained through an abstraction function* leading from the concrete state space into a more „abstract“ one. At the same time he gave criteria for a correct implementation of the abstract operators.

This insight led to exploration of „abstract types“, as in the language CLU (Liskov et al), as well as to new operating system structures, such as monitors.

Cultural Impact 4

OO was a practical tool in the development of certain graphical applications, in which some objects would be pictorially represented for every user to see.

These pioneering efforts lead to several important operating systems, such as Macintosh Operating System, and then to Windows.
Cultural Impact 5

The first language project directly influenced by Simula 67 was probably Smalltalk, developed by Alan Kay and his colleagues during the latter half of the 1970's.

It exploits the close association between the user and his program which is possible using an interpreted language.
Cultural Impact 6

A large flora of OO programming languages exist today. Some of them are listed and briefly commented upon in the paper. Here we only mention a few particularly in vogue today: C++ (Stroustrup), JAVA, and several LISP dialects.
Cultural Impact 7

The quasi-parallel (coroutine-like) execution of Simula „processes“ has not caught on as a general purpose programming tool. However, a natural development would have been to introduce actual concurrency.

Unfortunately JAVA has done that in a way not compiler controlled. Thereby much of the programming security inherent in the language is lost.

The Microsoft Component Object Model, on the other hand, does use concurrent processes. COM is an important common basis for programming languages such as C#, as well as for other tools.
Finally I would like to try to answer the following fair question: How could it happen that a team of two working in the periphery of Europe could hit on programming principles of lasting importance?

No doubt a bit of good luck was involved. We were designing a language for simulation modelling, and such models are most easily conceived of in terms of cooperating objects. Our approach, however, was general enough to be applicable to many aspects of system development.