Early History of Object Oriented Programming

Per M. Kjeldaas
Louisiana Tech University
CSC 532 Student
October 26, 2004
# TABLE OF CONTENTS

Abstract .......................................................................................................................... 3  
Background .................................................................................................................... 3  
Simula I ............................................................................................................................ 4  
Quasi-Parallel Processing .............................................................................................. 6  
Simula 67: Classes and Objects ...................................................................................... 7  
Subclasses and Inheritance ............................................................................................ 8  
Polymorphism through Virtual Procedures and “inner” ................................................. 8  
Garbage Collection ........................................................................................................ 9  
“Remote” Element (Object) Connection ....................................................................... 10  
Components and Separate Compilation of Classes ....................................................... 10  
Real-Time Simula ............................................................................................................ 11  
Spreading of the Word .................................................................................................. 12  
Finally Catching On ....................................................................................................... 14  
Postscript ....................................................................................................................... 15  
References ..................................................................................................................... 16
Early History of Object Oriented Programming

Abstract

Did object-oriented programming start with C++, or was Smalltalk the earliest object-oriented programming language? Neither of them was first. The first object-oriented programming language, Simula 67, was developed in the northernmost country in Europe. As the name suggests, it was released as early as 1967. Like the Columbi egg, the concept of object orientation is simple and elegant in retrospect, but was not so a priori. It took an operational analysis expert doing Monte Carlo simulations by hand and working with an expert programmer to develop the idea. This paper will explore some of the events that led to object orientation in general and this language in particular. It will also discuss how the concept later spread, including how misguided official policies slowed the adoption of the language and the concept of object orientation in general.

Background

Object orientation grew out of a very practical need: Some computing problems were too complex to easily be programmed in FORTRAN, ALGOL or COBOL. Discrete-event or Monte Carlo simulation was such an area, where no good tools were available to help with the extensive number crunching needed.

In many fields, innovation comes about through the crossing of ideas from different disciplines. Kristen Nygaard, one of the two inventors of object-oriented programming, worked as an operational analyst in the Norwegian Defense Research Establishment, and was
doing Monte-Carlo simulations of nuclear processes by hand. He needed to find a way to automate the process. He connected with a very talented programmer, Ole-Johan Dahl. The sociological history of how these two people came to be working together, how the research was funded, and how it had to go through twists and turns are described in research by Jan Rune Holmevik (1996).

**Simula I**

The first product that came out of their cooperation was Simula I, a preprocessor to Algol-60. “SIMULA was intended from the very outset to be simultaneously a systems description and a programming language” (Holmevik 1996). Using Simula I, there were the concepts of an active process, “station”, and a passive one, “customer”. For example, in a bank simulation, the clerk (station) could be the active process that is doing all the work, while each customer in line is considered a passive process or object. In other words, the station “grabs” the passive customer (object) and does with it what is needed. This system was called a “discrete event network”. Later, this network concept was replaced by a quasi-parallel mechanism. Here is how Holmevik shows it graphically (1996):

Source: (Holmevik 1996)
Nygaard and Dahl describe Simula I in their own historical account (1978):

Using the July 1963 paper … as a platform, we find that the basic concepts of SIMULA at that time were:

1. A system, consisting of a finite and fixed number of active components named stations, and a finite, but possibly variable number of passive components named customers.
2. A station consisting of two parts: a queue part and a service part. Actions associated with the service part, named the station's operating rule, were described by a sequence of ALGOL (or ALGOL-like) statements.
3. A customer with no operating rule, but possibly a finite number of variables, named characteristics.
4. A real, continuous variable called time and a function position, defined for all customers and all values of time.
5. The possible sequence of positions for a given customer implied that a customer was generated by a service part of a station, transferred to the queue part of another (or the same) station, then to the service part of that station etc., until it disappeared by not being transferred to another queue part by the service part of some station.

Since this structure may be regarded as a network, and since the events (actions) of the stations' service parts were regarded as instantaneous and occurring at discrete points of time, this class of systems was named discrete event networks.

It should be noted that at this stage:

1. Each individual station had to be described by a declaration:
   `station <identifier>;<statement>`
   the <statement> being single, compound or a block.

2. Customers were declared collectively by
   `customer <customer list>`
   where the <customer list>-elements had the format
   `<identifier>(<list of characteristics>)[<integer expression>]`
   the <integer expression> indicating the upper limit to the number of this type of customer being present.

3. A system was declared by
   `system <identifier> := <station list>`
   where the elements of the <station list> were identifiers of stations.
From the above one can conclude that even in Simula I, there was separation of concerns in that the stations were programmed separately from the customers.

Quasi-Parallel Processing

In discrete-event simulation, it is important that simulated time progresses as real time progresses, i.e. there is no backtracking. So in order to keep concerns separate, control must be able to pass from one process to another and later to resume where it left off (as adopted by threads today in Java). This was even long before the time-sharing came about with Multics. Stein Krogdahl writes (2003):

Quasi-parallel sequencing is analogous to the notion of coroutines described by Conway in 1963 [1]. In papers on Simula 1 or Simula 67 OJD [Dahl] and KN [Nygaard] always refer to this paper when discussing quasiparallelism, usually by saying something like “a set of quasi-parallel processes will function as coroutines in the sense of [1]”. However, also Simula 0, designed in 1962, had some traces of quasiparallel execution, even if the process concept was not fully developed until February 1964. It is therefore not clear to the current author [Krogdahl] whether OJD and KN developed this concept themselves, or got it from Conway.

In Simula 67 the Simulation system class includes a number of classes and procedures for handling the quasi-parallel sequencing needed for discrete-event simulation. A simulated object is an instance of a subclass of process, which is defined in Simulation. Programmers using Simulation will naturally think in parallel, but can concentrate on writing one process type of class at a time. This provides separation of concerns. The Simulation package class includes procedures or methods for activating a process object, thereby executing a segment of code; suspending or stopping execution; waiting for a specified amount of simulated time; terminating a process; etc. Many of these concepts show up in Java threads.
Simula 67: Classes and Objects

One day, Kristen Nygaard had the idea that he could also turn this relationship between station and customer around. Why not have the customer be the active process that does all the work, and that “grabs” the clerk when the time comes. From that realization there was only a small step to making both the clerk and the customer active processes and then include in each process only the information and activities that “belong” there. This meant that either type of object could be active as well as passive.

This class and object concepts became basic parts of the language with Simula 67 (Dahl, Myhrhaug and Nygaard 1968), and have been essentially unchanged from then till today with Java. The station concept of Simula I was generalized to become an object in Simula 67. The same was true for the customer concept, or later element; they all became objects. The natural language expression “class of objects” was used for the program description of the class, using the class keyword.

This doesn’t mean that they immediately foresaw the opportunities of abstraction that the class concept provided (Nygaard and Dahl 1978):

No mention of the class concept as an abstraction mechanism is made in the Lysebu [a conference in Norway very critical for defining Simula 67] paper. It took several years of slowly growing understanding, …until the fundamental difference between the internal ("concrete") view of an object and an external ("abstract") one finally was made clear by [C.A.R.] Hoare.

Reference variables were also introduced, which worked similarly to how pointers work in C++ today. They were strongly typed like in Java. A special type of assignment operator “:-” was used to distinguish a reference assignment from a value assignment, which used “:=”.
An object was created using a new keyword, and this concept has been adopted almost unchanged in Java today:

\[
\text{ref (Classname) } x; \ x \ :- \ new \ \text{Classname} \ (\text{parameters});
\]

**Subclasses and Inheritance**

The concept of subclasses was introduced with Simula 67 (Dahl, Myhrhaug and Nygaard 1968) in a way that was very similar to today’s usage in Java and C++:

A class may be used as "prefix" to another class declaration, thereby building the properties defined by the prefix into 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;
```

The implementation of subclass objects was done by adding the additional attributes and methods to the superclass. Therefore the term prefix corresponds closely with the physical representation. (Personal recollection.)

**Polymorphism through Virtual Procedures and “inner”**

Stein Krogdahl (2003) has explained well how virtual procedures came about:

Classes in Simula 67 may, like procedures, have formal parameters to which actual values must be given when an object is generated. However, for technical reasons they did not allow procedures as parameters to classes, even if they saw that this could be very useful when one wanted the “same statements” to have slightly different effect in different objects. After much work and discussion, they finally found a mechanism they agreed could regain much of the lost flexibility: By declaring a procedure in a class C as “virtual”, it could be redefined (overridden) in a subclass D, and in objects of class D this redefinition should have effect also when the procedure is called in the code of class C (and when the procedure is called in a D-
object by dot access through a C-typed pointer). Thus, the same procedure call could activate different “versions” of the procedure, at least in objects of different subclasses. And, as we know today, this mechanism turned out to be very useful.

Another mechanism that creates polymorphism is the keyword inner. It works similarly to a virtual procedure in that the statements of the subclass (if any) are executed when the inner statement is executed. And just like for virtual procedures, the actual runtime type of the object (not the type of the reference pointer) determines which statements are executed.

**Garbage Collection**

The block structure of Algol-60 lent itself well to a storage system based upon a stack. However, the quasi parallel control flow of Simula 67 was incompatible with such a structure. With arbitrary object creation and destruction, a new storage system had to be developed. The Algol-60 stack allocation system was not sufficient to handle object storage; a heap-type scheme was needed (Nygaard and Dahl 1978):

> Quasi-parallel execution of processes implied that control would switch from one process to another as the result of special sequencing statements such as "hold". Each temporarily inactive process had a "reactivation point" (represented by a system variable local to the process) which identified the program point where control should resume operations next time it entered the process, with the new storage allocation package quasi-parallel sequencing statements could be allowed at arbitrary program points, e.g. inside procedures called by the processes, since their data stacks could grow and shrink independently. … Furthermore processes could be created and destroyed in arbitrary order.

This storage mechanism was part of Simula 67 from the beginning, as was a garbage collector. However, the garbage collector did not run in parallel with the program is the case
today. Since the major application of the language was simulation, being a batch program, this restriction did not matter at the time. It worked.

“Remote” Element (Object) Connection

Even SIMULA I had a keyword `inspect`, that worked similarly to the

```java
if (x instanceof A) { …}
```

construct in Java. An example of use of this keyword used in conjunction with when and otherwise is shown below (Dahl and Nygaard 1966) with obvious meaning:

```simula
inspect (element expression) when A1 do S1
when A2 do S2
when An do Sn
otherwise S;
```

The local variables of the element expression were visible to the S blocks. In addition, casting could be performed through the keyword `qua`.

Components and Separate Compilation of Classes

The term package was used for a separately compiled class. Such a class would typically include inner classes. The author recollects that the Simula inventors were talking about using separately compiled classes as interchangeable part like in manufacturing, or like components as we call them today.

As a major departure from Simula I, which had simulation integrated into the language, in Simula 67 the special simulation features were collected into a special class or package called Simulation. It had a number of predefined classes and variables; for example, the process class was the superclass of all classes of objects that dealt with simulated time in
quasi parallel. Dahl, Myhrhaug & Nygaard clearly saw the potential for components that could be put together like building blocks (1968):

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.

The concept of a package was discussed in the Simula group, but using a class as the package had the benefit of not introducing another keyword (Krogdahl 2003). One disadvantage was that since Simula 67 did not have multiple inheritance or interfaces; one could only use one package.

Real-Time Simula

As far as the author knows, a real-time version of Simula was never developed. However, in 1970, there was serious discussion in Norway about making an operating system in Simula, with capabilities like the ones in the Control Data 3X00 Master operating system. It was discussed at the time to use Simula to simulate an operating system, and then change the simulation gradually over to actually control the hardware. (Personal recollection.) The simulation facilities were there to handle parallel threads the same way simulation processes were handled, but it was never pursued. In retrospect, that was a lost opportunity.
Spreading of the Word

While Drs. Dahl and Nygaard were well ahead of their time in simulation and computer language innovation, they were not marketing geniuses. Neither was the Norwegian government agency funding the research. Simula 67 and its innovations therefore did not spread through the research community, as one should have expected in retrospect. Here are some of the reasons:

**Simula was expensive.** At the time, a new programming language came out every few years or more often. Special-purpose languages were in vogue at the time. The government agency managing research funds in Norway, which directly or indirectly funded all research in this area, believed that Simula would have a life of up to 5 years. Therefore it was important to get the money out of the project during that time period, and the license fee was accordingly set at a fairly high level. Since some computer manufacturers (Control Data, Univac) appeared to be willing to pay a relatively high price for having Simula implemented on their computers, it appeared to the Norwegian research authorities that potential customers would be willing to pay a high price to get the software.

In reality, however, the “high price” or value did not reflect what the computer manufacturers would be willing to pay out in cash. Quite the contrary, the “purchase” of this software was just used by the sales and marketing departments of the computer manufacturers to justify a discount on the hardware they sold to the Norwegian research agency, without running afoul of the manufacturer’s standard contract with the U. S. General Services Administration, which always included a most-favored clause. With a gross margin of 90% at the time, it was more important for the manufacturer to get the additional
sale without cheapening the product image, than actually getting the list price. (Personal knowledge from working for Control Data).

Unfortunately, “Donald Knuth in 1973 was prevented from introducing SIMULA at Stanford University, partly because of NCC's unwillingness to reduce prices, and give it away free of charge to universities” (Holmevik 1996). It therefore appears that this factor may have been the single greatest obstacle to spreading of concept of object orientation.

_Lack of publications:_ The culture at the Norwegian research institutions at the time did not require “publish or perish”. Indeed, while a number of papers and reports were published locally in Norwegian, only the most revolutionary discoveries would be sent to an English-language international periodical. While Dahl and Nygaard published a paper about Simula I in the Communications of the ACM (1966), Simula 67 was only published in a much less available and more specialized forum such as the Winter Simulation Conference in late 1968, even though Simula 67 was much more revolutionary than Simula I.

_The Simula name:_ In retrospect it would probably have been better if the name of the language had not been Simula. The name of the language suggests that it is a special-purpose language limited to discrete event simulation, and that it would not be useful for programming other types of problems. The Simula name might have turned off people who were looking for a good general programming language.

_Other reasons:_ Sklenar (1997) lists a number of other reasons, such as lack of interfaces, lack of multiple inheritance, too complicated, limited file access and associated data types, “born in a small European country”, and a number of others. Some of the other reasons are more apparent today than they could possibly have been in the late 1960’s, such as the lack of a GUI and a lack of a modern IDE.
Finally Catching On

After a number of years the ideas from Simula eventually caught on. Alan Key learned about it, and used it as a model for the object-oriented aspects of the Smalltalk language. The author was told by Kristen Nygaard (Personal communication at ACM OOPSLA, Oct. 2001) that Alan Kay used a “bootleg” copy of the Control Data implementation of Simula 67 as the basis for developing Smalltalk, which is not surprising given the restrictive and expensive distribution policies of the Norwegian funding agency mentioned above.

Bjarne Stroustrup gave credit to Simula in the first sentence of his overview of C++ history: “C++ was designed to provide Simula's facilities for program organization together with C's efficiency and flexibility for systems programming” (1993).

Of course, it is well known today that the Java language has heavily borrowed from Smalltalk and C++, but less known is the fact is also is directly based upon Simula 67. In a speech at JavaOne, James Gosling of Sun Microsystems (1996) said:

The [Java] language came about as a fusion of four primary different kinds of programming. There's the object-oriented school from languages like SmallTalk; or one of my favorites, but mostly forgotten, Simula 67. …

It is interesting to note that there are still a number of people who prefer to program in Simula to any of the newer languages, such as the members of the Association of Simula Users. During the 1980’s and 1990’s a number of universities used Simula as a language for teaching object-oriented programming. Vaucher writes (2000):
At the Université de Montréal, we recently (1994-1998) used Simula as the basic language to teach programming. … Since then, our department got overwhelmed by the Java Tidal Wave and Simula is only used peripherally. It is small consolation to us Simula fans that all students who were exposed to the language feel Simula would have been a much better base than C++ from which to develop "Java".

Postscript

If the inventors had known then what we know today about what works in spreading new ideas, they probably would have made Simula open-source freeware, and would have received the return from their investment in services or general academic recognition.

Such international recognition came very late to Messrs. Dahl and Nygaard, as they jointly in 2002 were awarded the “2001 A.M. Turing Award, considered the "Nobel Prize of Computing," … for their role in the invention of object-oriented programming, the most widely used programming model today” (ACM 2002).

That same year, they both passed away.
References


