THE BIRTH OF SIMULA

Stein Krogdahl

Department of Informatics, University of Oslo, Norway; steinkr@ifi.uio.no

Abstract: When designing Simula, Ole-Johan Dahl and Kristen Nygaard introduced the basic concepts of what later became object-orientation, which still, 35 years later, has a profound impact on computing. This paper looks at the background for the Simula project, the way it developed over time, and the reason it could became so successful.

Key words: Computing history, programming languages, Simula

1. INTRODUCTION

On many occasions, people have told the history of how the programming language Simula came into being. Perhaps the foremost of these was the paper on Simula at the "History of Programming Languages" conference in 1978 [5] by Ole-Johan Dahl (OJD) and Kristen Nygaard (KN) themselves. However, people have given many other accounts, e.g. one by J.R. Holmevik [9] in 1994 and one by OJD [2] in 2001.

Most of these papers provided chronological accounts of what happened during the years when OJD and KN developed the two Simula languages, first the special purpose simulation language Simula (now usually referred to as Simula 1) and then the general-purpose language Simula 67. The latter is now officially renamed "Simula", but to make a clear distinction, we shall refer to it here as Simula 67.

This paper will take a slightly different view in that it will first give a rather short chronological overview of what happened when they developed the Simula languages during the years 1961 to 1967. After that, we will look at different aspects of Simula 67, and try to find where they originated, when

they came into the development of the Simula languages, and how and why they got their final form in the language. The papers [5] and [9] contain much more extensive lists of references than this paper does.

2. A CHRONOLOGICAL OVERVIEW

This section gives an overview of the development of the Simula languages during the years 1961 to 1967. Four phases define this development.

2.1 The early design phase (1961-August 1962)

The start of the Simula story is, as most such stories, not clearly defined. Sometime around 1960/61 KN started working on ideas for a language that could naturally describe complex dynamic systems. The background was that he had worked with so-called "Monte Carlo simulations" while at Norwegian Defense Research Establishment (NDRE). People performed these simulations both "by hand" and on early computers. KN felt a strong need for a language or a notation in which he could describe such systems at a much higher level than was usual at the time. From the nature of the problems he had worked with, his main interests were discrete event models, where they modeled a dynamic system as a set of variables operated upon by a sequence of instantaneous events.

In January 1962, we find the first written account of this effort. In a letter to a French operations research specialist, Charles Saltzmann, KN writes:

"The status on the Simulation Language (Monte Carlo Compiler) is that I have rather clear ideas on how to describe queuing systems, and have developed concepts which I feel allow a reasonably easy description of large classes of situations. [...] The work on the compiler could not start before the language was fairly well developed, but this stage seems now to have been reached. The expert programmer who is interested in this part of the job will meet me tomorrow. He has been rather optimistic during our previous meetings."

The expert programmer was OJD, whom KN knew very well from NDRE (from where KN had left in 1960, to work at the Norwegian Computing Center (NCC)). From then on, KN and OJD were working together to obtain a draft of the language, and in May 1962 they had a first specification. By then they had also coined the name SIMULA (SIMUlation LAnguage) for the language, and their idea was to implement it as a

preprocessor to a general Algol compiler. They presented their ideas for this language at the IFIP 62 World Conference in Munich, in August 1962 [11].

In this early version of Simula, they modeled a system by a (fixed) number of "stations", each with a queue of "customers". The stations were the active parts, and each was controlled by a program that could "input" a customer from the station's queue, update variables (global, local in station, and local in customer), and transfer the customer to the queue of another station. Stations could discard customers by not transferring them to another queue, and could generate new customers. They could also wait a given period (in simulated time) before starting the next action. Custom types were declared as data records, without any actions (or procedures) of their own.

We shall refer to this language as "Simula 0", but it probably never had a stable definition and never had an implementation.

2.2 The Simula 1 design phase (Autumn 62 - March 64)

After the presentation of the ideas for Simula 0 in Munich, KN occupied himself with organizational and financial matters. This resulted in the purchase of a UNIVAC 1107 to NCC for a discount price, and, in the same contract, in funding for the implementation of Simula on UNIVAC 1107.

OJD went on working with the language, and in May 1963, he joined the NCC as a full time employee. However, it became more and more apparent that implementing the new language as a preprocessor to Algol was not the right way to go. The LIFO (stack) nature of the Algol procedure calls turned out to be too restrictive for implementing the "stations" and "customers" of Simula 0.

Thus, still determined to build upon an Algol compiler, they had to dig deeper into it, and replace the simple LIFO allocation mechanism with what today we would call a heap. For this purpose, OJD designed a scheme for automatically regaining unused memory (which did not compact the retained objects).

Inspired by the new freedom obtained from this allocation scheme, they also saw that a generalization of the language was possible. In Simula 0, stations were the active parts and customers the passive ones. They now replaced these with *one* concept that could describe all the different "objects" that should participate in a simulation. This concept was called a "process" (declared with the keyword "activity"), and could play both an active and a passive role in simulations. They kept track of the processes by (un-typed) pointers, and they did not restrict their lifetime by any LIFO discipline. The whole system had a built-in notion of simulated time that controlled the execution of the processes when they were active.

In March 1964, they had a well-defined language. It resembled Algol, with the addition of the process concept and the accompanying apparatus for simulation programming.

2.3 Implementation of Simula 1 (April 1964 - January 1965)

The implementation started immediately. Since they already understood much of the complexities surrounding the garbage collector, the whole task of reworking the UNIVAC 1107 Algol compiler to a Simula 1 compiler was finished already in January 1965. OJD led the work and to large degree carried it out. However, Bjørn Myhrhaug and Sigurd Kubosch from NCC (both of which were central in later Simula 67 implementations) also participated. They had occasional help from the team around the UNIVAC 1107 Algol compiler (Joseph Speroni, Ken Walter, and Nicholas Hobacker).

2.4 The design phase of Simula 67 (January 1965 - summer 1967)

When OJD and KN started using Simula 1 in various projects, they soon observed that people could profitably use it for other programming tasks than simulation. Thus, they started to explore what a more generalized version of Simula 1 should look like. However, the spark that really made the development of a new general-purpose language take off was probably a paper on record handling by C.A.R. Hoare [8]. It proposed to describe records by record classes and to introduce "record subclasses", described as extensions of other classes. It also argued that with such a scheme one could type pointers without losing too much flexibility.

Even if OJD and KN saw that Hoare's proposal had ideas that fitted well with a generalized version of Simula 1, it also had aspects that were difficult to combine with other ideas they had for the new language. A breakthrough came late in 1966, when they saw that they could declare classes and subclasses syntactically, independent of each other, instead of in one closed construct as in Hoare's proposal.

With this insight, other aspects of the new language (which they now had named Simula 67) naturally found their place, and they worked intensively the first months of 1967 to finish a paper for the IFIP TC 2 conference to be held in May at Lysebu in Oslo (chaired by OJD). The theme here was Simulation languages, and their paper [6] was finished in March, just in time for submission to the conference. Towards the conference they still worked with new ideas, and after much discussion, they came up with the (now well-known) concept of "virtual quantities".

In May 1967, they signed a contract with Control Data to implement the (still unfinished) language Simula 67. The implementation should be carried out at KCIN (the "Kjeller Computer Installation" serving NDRE) for CDC 3600, at the University of Oslo for CDC 3300, and in Paris (by Control Data itself) for the CDC 6000 series. To establish a first basic definition of Simula 67, a meeting (called the "Simula 67 Common Base Conference") took place at NCC in June, where people from Control Data, KCIN, University of Oslo, and NCC met, together with a few invited specialists.

Also at this conference KN and OJD proposed new additions to the language, notably a proposal that would make it possible to define named "inline objects", that is, to use classes as types in traditional variable declarations. However, as they did not fully work through this proposal, either conceptually or implementationally, the implementers voted it down. Probably they must have felt that the implementation task was large enough as it was.

This conference also set up a body called "Simula Standardization Group" (SSG), with representatives from NCC and the different implementers. Its first task was to establish a formal language definition. The main remaining issues were text handling and mechanisms for input/output and in the definition of these Bjørn Myhrhaug from NCC played a central role. They formally froze the language in the "Simula 67 Common Base Language" report, accepted by SSG on February 10, 1968 [4]. Bjørn Myhrhaug also, together with OJD, wrote an implementation guide for Simula 67 [3], which was important for later implementations of the language.

All the Control Data compilers were finished during the spring of 1969. In 1969, they also decided that NCC itself should complete a compiler for the UNIVAC 1100 series and implement Simula 67 for IBM 360/370. These implementations were finished in March 1971 and May 1972 respectively.

3. THE DIFFERENT ASPECTS OF SIMULA 67

The previous section gave a brief chronological overview of the development of the Simula languages. In this section, we shall take an orthogonal view, look at different aspects of Simula 67, and try to find their roots and the way they developed throughout the design process of Simula 1 and Simula 67.

Most of the interesting aspects of Simula 67 are associated with the class concept and the way one can generate objects of classes. When defined in 1967, the class concept represented a revolutionary new combination of

properties. Some of these came from the long development of the Simula 1 language, some came from other researchers, and some were developed by OJD and KN in the last months before they defined the language.

3.1 The object as a process

We shall first look at an aspect of Simula 67 which is not well known, and which is taken over by very few of its successors. This aspect is the ability of objects to act as processes that can execute in a "quasi-parallel" fashion. This special type of parallelism is in fact a form of sequential execution, but a form where you can get some of the benefits obtained by organizing the program as a set of "independent" processes. The idea is that classes in Simula 67 (and processes in Simula 1) have statements of their own (like procedures), and that these start executing as soon as an object of that class is generated. However, unlike procedures, objects may choose to temporarily stop their execution and transfer the control to another process. If one returns control back to the object, it will resume execution where the control last left off. A process will always retain the execution control until it explicitly gives it away. When the execution of an object reaches the end of its statements, it will become "terminated", and can no longer resume (but one can still access local data and call local procedures from outside the object).

The quasi-parallel sequencing is essential for the simulation mechanism in Simula 1. Roughly speaking, it works as follows: When a process has finished the actions it should perform at a certain point in simulated time, it decides when (in simulated time) it wants the control back, and stores this in a local "next-event-time" variable. It then gives the control to a central "time-manager", which finds the process that is to execute next (the one with the smallest next-event-time), updates the global time variable accordingly, and gives the control to that process.

The idea of this mechanism was to invite the programmer of a simulation program to model the underlying system by a set of processes, each describing some natural sequence of events in that system (e.g. the sequence of events experienced by one car in a traffic simulation).

Note that a process may transfer control to another process even if it is currently inside one or more procedure calls. Thus, each quasi-parallel process will have its own stack of procedure calls, and if it is not executing, its "reactivation point" will reside in the innermost of these calls. This means that not only processes and other objects, but also procedure calls must be stored on "the heap". (With the memory sizes of 60ies, they did not consider the later practice of allocating a large continuous chunk of memory to the stack of each process/thread). Thus, the traditional stack of procedure

calls disappeared in implementations of both Simula 1 and Simula 67, everything was on the heap (even if we could optimize this to some extent for specific programs).

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 and KN always refer to this paper when discussing quasi-parallelism, usually by saying something like "a set of quasi-parallel processes will function as coroutines in the sense of [1]". However, Simula 0, designed in 1962, also had some traces of quasi-parallel execution, even if they did not fully develop the process concept until February 1964. It is therefore not clear to the current author whether OJD and KN developed this concept themselves, or got it from Conway.

Many of the uses OJD and KN found for Simula 1 in other areas than simulation used the quasi-parallel sequencing for some other purpose (e.g. to traverse two search trees "in parallel"). Thus, there was no doubt that Simula 67 should retain this mechanism. Another reason for this was that they wanted the simulation aspect of Simula 1 to be expressible within Simula 67, as a separate package. However, the concept of simulated time then naturally belonged in this package, while a more basic mechanism for quasi-parallel sequencing was included in the Simula 67 language.

3.2 Dynamic generation of objects

A ubiquitous property of today's object-oriented languages is the ability to generate objects dynamically. This was also a basic mechanism in Simula 1 and indeed in Simula 67. However, the decision to allow this, taken in the transition from Simula 0 to Simula 1 was probably not quite straightforward. When they designed Simula 0 in 1962, the idea was to implement the language as a preprocessor to an Algol compiler, thus having to stick to the LIFO allocation discipline of procedures and blocks of that language.

This hampered the design of Simula 0 so that, for example, the number of "stations" (the active components) had to be fully determined at compile time, while the number of "customers" (the passive elements) had to be given at the start of each execution. Thus, when they saw that even this scheme was cumbersome to implement as planned, they looked for other solutions. Moreover, when forced to go deeper into the Algol compiler, they sought a solution that gave full freedom to generate objects at run time.

To obtain this they had to abandon the simple LIFO scheme for allocating procedure calls in Algol. This meant digging into the Algol compiler and its run time system, and implement a scheme where both objects and procedure calls could come and go (that is, become inaccessible) in any order. For security reasons, they also wanted the system itself to find

which objects and procedure call instances it could delete. For this purpose, OJD designed a garbage collector scheme based on reference counts and free-lists, and with a last resort search mechanism that could also find loops of objects that it could delete. It never moved any objects. This scheme was part of the Simula 1 implementation.

Later, when the Simula 1 implementation was finished, they learned (through their NDRE mentor Jan V. Garwick, once he returned from USA) about a four-phase compacting garbage collector that could do away with all the fragmentation they experienced with the Simula 1 implementation. They used this algorithm in the first Simula 67 implementations and it has since been a reliable element in most Simula 67 implementations.

3.3 Objects as "records", and access from outside

The decision that one single concept, the process concept of Simula 1, should cover all needs from quasi-parallel processes to simple data records was probably one of the most important ones in the history of the Simula project. As seen in the previous section, this decision was made possible by another important decision: to abandon the simple stack allocation scheme of Algol.

The significance of the "single concept decision" lies in the following:

- From this decision, it follows that also the need for simple data records (with no actions of their own) should be covered by the process concept. Thus we obviously have to be able to access such records from outside (e.g. by "dot access") to utilize their local data at all. Then, again by the single concept decision, one should allow access from outside also for processes in general.
- The process concept of Simula 1 is modeled after the Algol block. Therefore, it allows local procedures in addition to local variables and actions of the process itself. Thus, records with procedures and variables, but with *no* actions of their own, follow as a natural special case. Then again, to use such procedures we obviously have to be able to call them from outside the record. Thus, one should also allow calls to local procedures from outside processes in general.

We now recognize the latter point as one of the basic characteristics of object-orientation. The former point was very important for obtaining flexible communication between the processes in a simulation.

For reasons of flexibility and simplicity, they decided that processes should only be accessible through pointers. In Simula 1 there were no concept of subclasses, and the pointers were therefore not typed (typed pointers would have led to problems e.g. when processes of different kinds should be kept on a linked list). To keep the language type-safe, Simula 1

therefore had a rather cumbersome way of accessing the "inside" of a process referenced by a pointer 'pntr':

inspect pntr **when** P1 **do begin** ... **end when** P2 **do begin** ... **end** ...;

Here P1, P2, ... must be names of process declarations ("process types"). The effect of the above statement is first to determine the actual type of the process referenced by 'pntr', and then to enter the corresponding block, where the attributes of this process can be accessed directly.

When they introduced subclasses in Simula 67, the absolute need for such a construction disappeared (see below). However, the inspect statement could still be convenient in many situations, and they included it in Simula 67 (in a slightly different form).

3.4 Classes and subclasses

OJD and KN saw a number of cases where processes had many common properties when programming in Simula 1. However, because there were also differences, two fully separate processes had to be described. They had no way of utilizing the similarity. This all changed when C.A.R. Hoare published a paper on record handling [8], where he proposed classes and subclasses of records, but in a scheme where a class and all its subclasses were described in one closed syntactic unit.

He also proposed to type pointer variables with record classes (or subclasses) and only to allow them to point to records of the typing class and to records of all its subclasses. This gave full access security when allowing, for example, "pointer.attribute" only when the attribute is defined in the class typing the pointer. Hoare wrote "attribute(pointer)", as was common at the time, but Simula 67 turned to the now ubiquitous "dot-notation" (which was probably first used for this purpose in PL/1).

Equally important, with this scheme typing of pointers was almost not felt a nuisance at all. All records that should enter lists could be "subrecords" of a suitable record class "element", but otherwise be different.

However, even if this seemed promising, it did not really fit in with all the requirements they had for the new Simula 67. Hoare's records had no actions of their own, and, most importantly, the closed syntax did not fit in with other ideas they had. Nevertheless, late in 1966 they finally found a principle that made all the pieces fit together. The main idea was to allow the declaration of subclasses of a class C independently of where one declared C. That is, we can declare a new subclass D of C wherever the declaration of C is visible (with certain block level restrictions). The syntax chosen was to write C as a "prefix" to the declaration of D: "C **class** D (...); **begin** ... **end**" (which is why, in Simula literature, we often find the notion "prefix class" instead of the now more usual "superclass" or "base class").

The great advantage of this scheme is that we could write general-purpose classes separately and, for example, place them in public libraries (in textual or compiled form). Later, different users could fetch these classes, and could write subclasses to satisfy their own specific needs. This scheme was essential when they later wanted to program the simulation properties of Simula 1 as a package in Simula 67.

3.5 Virtual procedures

Virtual procedures are now an integral part of object orientation, even to the degree that in some languages, *all* procedures are virtual (e.g. in Java). However, the story of virtuals started with a very specific problem that OJD and KN wanted to solve. It was only after submission of the Lysebu paper in March 1967 that they had time to look at the following problem:

As with procedures, classes in Simula 67 may 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. As we know today, this mechanism turned out to be very useful.

The proposal for virtuals just reached the Lysebu conference, and was included in the final version of the paper [6].

3.6 Classes as packages

A basic idea with Simula 67 was that we could implement the simulation aspects of Simula 1 as a separate "package" in Simula 67. OJD and KN saw that implementing a variety of specially tailored library packages for different application areas would be important for the usability of the new language. As implementing the simulation aspects of Simula 1 was no simple task, the ability of Simula 67 to describe such a package became an important yardstick for its quality.

They immediately saw that such packages often would contain more than one class, and they therefore needed some sort of container to keep these

classes together. Their solution was to use classes also for this purpose and this worked fine because they had retained the full block structure of Algol in Simula 67. They could therefore have classes within classes, and could use the outermost class as a container representing the package as such. The innermost classes would then be those offered for normal use (either for direct object generation or for making subclasses that are more specific).

With this scheme, they could also make one package build upon another by simply making one "package class" a subclass of the other. To bring packages into the user's program Simula 67 uses "block prefixing". By prefixing a normal block with the name of a package class, the content of that class became visible in the block, just as if the block was the body of a subclass of the package class (also so that virtual procedures of the package class could be redefined in the prefixed block). Thus, e.g. for a simulation package, where the processes should be able to enter linked lists, they could have a scheme like this (where "!...;" is a comment, and the classes are slightly simplified):

```
class LinkedList; ! A package class implementing list handling ;
begin
   class Head; begin <a first-pointer, and suitable procedures>; end;
   class Elem; begin <a next-pointer, and suitable procedures>;  end;
end;
```

Using this package, one can define the simulation package as follows:

```
LinkedList class Simulation; ! This package uses package LinkedList ;
begin
   Elem class Process; ! All processes can be elements in lists ;
   begin <initial statements>;
      inner;     ! See explanation below ;
      <terminating statements>;
   end;
   <Further classes (or other declarations)>;
end;
```

We can then use such a package as follows:

```
Simulation begin
   Process class Car; begin ... end;
   Process class Bus; begin ... end;
   ...
end;
```

Note that package classes are totally normal classes, except that a class referring to "this" (the current object) is not legal as a block prefix.

One mechanism they saw the need for through examples like the one above, is the "inner" mechanism used in the class Process in the Simulation package class. This keyword has the effect that statements of subclasses of Process are positioned where the "inner" is, which means that the Process class gets control before and after the statements of a subclass. Thus, we can properly initialize and terminate any object of a subclass of Process (e.g. Car or Bus) by statements given in the prewritten class Process.

One may notice that the inner mechanism has similarities with the virtual mechanism, as the keyword "inner" represents the *statement part* of the actual subclass (if any), while the virtual mechanism correspondingly picks the *procedure redefinition* of the actual subclass. These mechanisms entered the Simula project for quite different purposes, and one may wonder whether a unifying mechanism had been found if the project had proceeded (as indicated earlier, the virtual mechanism came very late). However, it is interesting to note that in the language Beta (designed by KN and three Danish colleagues as a further development of the Simula ideas, see e.g. [10]) both the inner and the virtual concepts are retained as separate mechanisms.

It is impressive that OJD and KN saw the need for a package concept already in 1967. It is told (but not confirmed) that they discussed if a special concept like "package" or "module" should be introduced (instead of package classes), but the chosen solution has a pleasant economy of concepts, which might have decided the matter. However, it has the disadvantage that you can only bring in one package in each block. As multiple prefixing for classes (which later became known as "multiple inheritance") was not included in Simula 67, it followed naturally that also multiple *block* prefixing was excluded. With a separate concept for packages (for which you were *not* allowed to dynamically create new "package objects") it would have been easier both implementationally and conceptually to allow more than one package to be brought into a given block.

4. CONCLUDING REMARKS

In this final section, we shall look at the following question: How could a small project with (for most of the time) only two participants produce a language and a set of concepts so powerful and pervasive that they today, 35 years later, underlie most software development? Obviously, it is impossible to give a complete answer to this, but we shall look at some key

circumstances around the Simula project that the current author thinks were important for its success.

- First and foremost, both KN and OJD were extraordinary talented researchers in the field of language design. They were well oriented in the relevant literature of the time and had contact with a number of other researchers during the design of the Simula languages. They were persistent in their work, and never stopped looking for more general and elegant solutions. Last but not least: they had an excellent judgment concerning the balance between efficiency and elegance when they had to make a choice, but they always strove to obtain both.
- Even if OJD and KN were similar in the sense that they both had the qualities described above, they were quite different in other respects, and could thereby complement each other during the work. KN had the idea to start with, and even if OJD indeed participated in developing the concepts, KN probably remained the one that most naturally expressed the high-level dreams and hopes for the project. On the other hand, OJD was the one who immediately saw the implementational consequences of the different proposals, and could thereby mould them into a form that, to the largest possible extent, took care of both implementational and conceptual aspects.
- Both KN and OJD were robust personalities, and critique of new ideas did not at all drown in politeness. In [5] they wrote, "In some research teams a new idea is treated with loving care: How interesting! Beautiful!" This was not the case in the Simula development. When one of us announced that he had a new idea, the other would brighten up and do his best to kill it off." In such an environment, they accepted no new concept without deep and lengthy discussions.
- They were part of a European tradition in language design that strongly emphasized concepts such as economy, orthogonality, and simplicity. The first and foremost result of this proud tradition was the Algol language, and also, as described above, the elements of that language had a direct and profound influence on the Simula project.
- If we consider the object-oriented concepts as the main outcome of the Simula project, we can see that starting the project by designing a language for simulation was of utmost importance for that outcome. A central aspect of simulation programming is to represent the part of the world that we want to simulate by a suitable data structure inside the machine. Thus, KN and OJD saw that advanced and flexible mechanisms for describing and setting up data structures were of vital importance for a successful simulation language, and they worked hard to obtain such mechanisms. Through this they were led to a language design for Simula 1 that invited the programmer to first choose the data structure, and then tie the actions (or action sequences)

of the model to the part of that structure where they most naturally belong (and not the other way around, as was natural in "procedural languages" like Algol). We can view this change of programming strategy as the main paradigm shift towards object-orientation. It naturally first appeared in the world of simulation, but OJD and KN could later take their experience with this principle from the simulation world to the design of the general-purpose language Simula 67 (and here it paved the way for the later concept of abstract data types, see [7]).

- It was very fortunate that Hoare's ideas on records and subrecords in [8] appeared exactly at the time when KN and OJD were working on generalizing the ideas of Simula 1 to fit in a general-purpose setting. Thus, they could immediately recognize the potentials of Hoare's ideas, and after intense work, combine them very successfully with the ideas they already had. This merge of ideas was probably the most important event in the development of the object-oriented concepts.
- It was probably also important that the Simula project spanned such a long period. If it had been a project with full funding from the start and a strict time limit, it would probably have ended up with an implementation of the Simula 0 concepts. Instead, the funding was not at all clear, and KN had occupied much of the time with other matters (which to some extent resulted in funding for the Simula project). However, OJD could keep the project going, having discussions with KN whenever possible. Thus, the ideas developed over many years and run through many phases. It was probably also vital for the outcome that they implemented a full-fledged language (Simula 1) along the way, so they could study its potentials and shortcomings in real use.
- Finally, with the shift from the special purpose simulation language Simula 1 to forming a general-purpose language they got the problem of designing Simula 67 so they could naturally express the simulation aspects of Simula 1 as a separate package in Simula 67. As these simulation aspects relate to many different mechanisms in the language (e.g. the coroutine mechanism) this was no easy task. Thus, the ability of Simula 67 to describe such a package in a simple and direct way became an important yardstick for measuring the quality of different proposals.

ACKNOWLEDGMENTS

While preparing this paper I have been in contact with a number of people, and in particular, I would like to thank the following persons for help and advice: Karel Babcicky, Håvard Hegna, Donald E. Knuth, Dag F. Langmyhr, Arne Maus, Bjørn Myhrhaug, Birger Møller-Pedersen, Olaf Owe, Jo Piene, and Wilfried Rupflin.

REFERENCES

[1] M.E. Conway. Design of a separable transition-diagram compiler. *Comm. ACM*, 6, 1963.

[2] Ole-Johan Dahl. The Roots of Object Orientation: The Simula Language. *Software Pioneers' Conference, Bonn, June 2001. In "Software Pioneers", Springer, 2002.*

[3] Ole-Johan Dahl and Bjørn Myhrhaug. *SIMULA 67 Implementation Guide*. Norwegian Computing Center, Oslo, Norway, Publ. S-9, June, 1969.

[4] Ole-Johan Dahl, Bjørn Myhrhaug, and Kristen Nygaard. *SIMULA 67 Common Base Language*. Norwegian Computing Center, 1968.

[5] Ole-Johan Dahl and Kristen Nygaard. The development of the Simula language. In Wexelblat, editor, *History of Programming Languages*, pages 439-493, 1981.

[6] Ole-Johan Dahl and Kristen Nygaard. Class and subclass declarations. In *Proceedings from IFIP TC2 Conference on Simulation Programming Languages, Lysebu, Oslo, ed.: J. N. Buxton*, pages 158-174. North Holland, May 1967.

[7] C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica, Vol 1, no 4*, pages 271-281, 1972.

[8] C.A.R. Hoare. Record handling. *Algol Bulletin No. 21*, November 1965.

[9] Jan Rune Holmevik. Compiling SIMULA: A Historical Study of Technological Genesis. *IEEE Annals of the History of Computing*, Vol 16, No. 4, 1994.

[10] B.B. Kristiansen, O.L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley, 1993.

[11] Kristen Nygaard. SIMULA: An Extension of ALGOL to the Description of Discrete-Event Networks. *Proceedings of the IFIP congress 62, Munich, Aug 1962*. North-Holland Publ., pages 520-522.