IX

SIMULA Session

Chairman: Barbara Liskov
Speaker: Kristen Nygaard
Discussant: Ole-Johan Dahl

PAPER: THE DEVELOPMENT OF THE SIMULA LANGUAGES

Kristen Nygaard

Norwegian Computing Center and University of Oslo

Ole-Johan Dahl

University of Oslo

Preface

The organizers of this conference have told us that we should write at least 25 pages of manuscript, but that we may produce as many pages more as we wanted. Perhaps they did not envisage the possible consequences, but we have taken their words at face value.

This paper has implied a vast amount of work and archeological activities. We are grateful to SIGPLAN for defining a task to which resources had to be allocated by our institutions and which forced us to write down an account of our work from 1961 to 1971. While we are writing this preface, those years are very much alive to us. We realize that we lived through this period in a state of semimadness, a combination of very hard work, frustration, and euphoria.

The observations which have impressed us most are:

that the results of the SIMULA effort were completely dependent upon the joint beliefs, work, ideas and advice of a very large group of people, and

that at many stages the whole effort continued just because of lucky circumstances.

Have we told the truth about SIMULA's history? Yes, to our best knowledge, we have. But—have we told the whole truth? No, we have not.

SIMULA's history is intertwined with that of the Norwegian Computing Center. And

ISBN 0-12-745040-8

the emergence of NCC in the 1960s is an important part of the history of informatics in Norway. It is too early to tell that history, but our professional society is now starting at least to collect all the stories remembered. In this paper we are deliberately vague when we want to avoid accusations against persons and institutions, and yet indicate problematic situations.

In our time travel during the last few months, many of those involved have put their memories at our disposal. We want to express our gratitude. In the actual writing we have been assisted by Birger Møller Pedersen and Paul Wynn of the NCC, and the typing has been very competently done by Lise Tschudi assisted by Eileen Schreiner.

1. Background

The development of the SIMULA I and SIMULA 67 languages was carried out at the Norwegian Computing Center (NCC). The early background for the development is, however, our work at the Norwegian Defence Research Establishment (NDRE) in the 1950s.

KN started his conscript service at the NDRE in 1948 as assistant to Jan V. Garwick—the father of computer science in Norway. Their first main task was to carry out resonance absorption calculations related to the construction of Norway's first nuclear reactor. After extensive work had been invested in a traditional numerical approach, Monte Carlo simulation methods (by ''hand'') were successfully introduced instead in 1949–1950. KN headed the ''computing office'' until mid 1952, and then became a full time operational research worker.

OJD joined the NDRE in 1952, also as a soldier doing his conscript service. Garwick and his assistants had, since 1949, followed the development of electronic computers. A BULL punched-card calculator had, in 1951–1953, been extended at the NDRE into a card-programmed electromechanical computer. In 1954 it was decided that NDRE should acquire a Ferranti MERCURY computer, then at the design and construction stage, and in the following years the NDRE milieu developed basic software which was fairly advanced for its time.

In the late 1950s, the NDRE milieu started work in language design, in which Garwick and OJD were particularly active.

From 1956 on the operational research (OR) activities expanded rapidly. In the large scale OR jobs, simulation once more turned out to be the only feasible tool for analysis of sufficiently realistic models. Also, it became evident that no useful and consistent set of concepts existed in terms of which the structure and interaction in these complex systems could be understood and described. Since the task of writing simulation programs for the MERCURY computer became important, the lack of tools for this task was a serious obstacle.

In May 1960 KN left the NDRE to build up the NCC as a research institute in computer science, operational research and related fields. Many of the civilian tasks turned out to present the same kind of methodological problems: the necessity of using simulation, the need of concepts and a language for system description, lack of tools for generating simulation programs. This experience was the direct stimulus for the ideas which in 1961 initiated the SIMULA development.

NCC is a semigovernmental research institute, established in 1958, working within informatics, operational research, numerical analysis, and applied statistics. The task of the institute basically is to assist in disseminating new methods and tools in these fields to the

user environment in Norway. In fulfilling this task, NCC is supposed to take on practical jobs and to develop new knowledge through pure and applied research.

NCC is supervised by the Royal Norwegian Council for Scientific and Industrial Research. The institute is funded by research contracts with customers and contributions from the Research Council. Today (1978) approximately 60% of NCC's income stems from customers, 40% from the Research Council. Of this 40%, 75% is grants to the institute's activities in general, and 25% (10% of the total) earmarked contributions to specified research projects. The staff amounts to 76, of which 55 are research workers. The customers of NCC come from private commerce and industry, public agencies, other research institutions, and (since 1971) trade unions. NCC has played an important part in the Norwegian and later Scandinavian development of workers' influence on planning, control and data processing systems.

2. SIMULA I

2.1. Early History

The ideas for a language which could serve the dual purpose of system description and simulation programming originated at the NCC in the spring of 1961. The first written reference to SIMULA is found in a letter dated January 5, 1962, from KN to the French operational research specialist Charles Salzmann (Nygaard, 1962a):

> The status of the Simulation Language (Monte Carlo Compiler) is that I have rather clear ideas on how to describe queueing systems, and have developed concepts which I feel allow a reasonably easy description of large classes of situations. I believe that these results have some interest even isolated from the compiler, since the presently used ways of describing such systems are not very satisfactory.
>
> I hope that later developments, for which I have a number of ideas, will include e.g. stochastic inventory situations amongst the situations which may be described by the language.
>
> 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 naïve optimism evidenced by this quotation was perhaps an asset. Had we at that time known the magnitude of the efforts ahead, we had at least had a more difficult task in getting the necessary resources for the project.

The ''expert programmer'' was of course OJD, who had been contacted by KN in December 1961 when the first set of apparently powerful ideas had appeared. Then it became evident that the ''SIMULA project'' could only be carried out successfully if it combined:

experience in operational research, particularly related to large, complex systems of many different kinds, with

experience and knowledge in computer programming language design and implementation.

OJD immediately got interested and participated in a long series of discussions with KN during the spring of 1962. In May we felt that we had a language proposal which we could present to other people. At that time OJD had become involved to such an extent that he

decided to leave NDRE for NCC. He started his work at the NCC in March 1963, after having finished his current project at NDRE (implementing a homemade ALGOL-like programming language).

In May 1962 Univac launched a big campaign for their new computers, the famous UNIVAC 1107 and the (now forgotten) UNIVAC III. KN was invited to join the "Univac Executive Tour" to the US, and accepted with the intention of reversing the sales mission by selling SIMULA to Univac. To the surprise of all involved, both parties succeeded: The NCC got a UNIVAC 1107 in August 1963. Univac and NCC entered a software contract, and NCC had completed the SIMULA I compiler in January, 1965. The politics surrounding these events was complex, often tough, and gave some of those involved wounds which were difficult to heal.

There was no initial enthusiasm for SIMULA in NCC's environment, and we were told that:

1. There would be no use for such a language as SIMULA.
2. There would be use, but it had been done before.
3. Our ideas were not good enough, and we lacked in general the competence needed to embark upon such a project, which for these reasons never would be completed.
4. Work of this nature should be done in countries with large resources, and not in small and unimportant countries like Norway.

We had, however, the support of the board of NCC, and SIMULA was linked to the acquisition of NCC's UNIVAC 1107 computer.

2.2. The Main Development Stages

The SIMULA I language went through four main stages:

1. The summer of 1961–the autumn of 1962: The initial ideas based upon a mathematically formulated "discrete event network" concept and a programming language reasoning which had no specific implementation situation in mind. The main references are a working document (Nygaard, 1962b, in Norwegian) and the SIMULA presentation at the IFIP World Congress in Munich, August 1962 (Nygaard, 1963a).
2. The autumn of 1962–September 1963: Development of the early approach, increased flexibility introduced by the possibilities of ALGOL 60, at the same time restricted by the assumption of SIMULA being implemented by a preprocessor to ALGOL 60 combined with a "procedure package". The main references are Nygaard (1963d) and Dahl and Nygard (1963), the latter being the language definition of the SIMULA I software agreement between Univac and NCC (Agreement, 1963).
3. September 1963–March 1964: Decision to implement SIMULA I through a modification and extension of Univac's ALGOL 60 compiler, based upon a new storage management scheme developed by OJD. The introduction in February 1964 of the "process" concept, utilizing the new possibilities available. The main reference is Dahl and Nygaard (1964a).
4. March 1964–December 1964: The implementation of the SIMULA I compiler. Minor language modifications and extensions based upon implementation experience and programming test cases. The main reference to the resulting SIMULA I language is Dahl and Nygaard (1965).

In this section we will describe the SIMULA I of stage 2 and its transition into the SIMULA I of stage 3. The reasoning which motivated this transition and thus the final SIMULA I language will be presented in Section 2.3.

Using the July 1963 paper (Nygaard, 1963d) 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.

The best way to give the reader a feeling for the language at this stage is to present some fragments of an example—an "airport departure" model.

```
system Airport Departure := arrivals, counter,
    fee collector, control, lobby;
customer passenger (fee paid) [500]; Boolean fee paid;
```

```
    ⋮
station counter;
    begin accept (passenger) select:
            (first) if none: (exit);
    hold (normal (2, 0.2));
    route (passenger) to:
    (if fee paid then control else fee collector)
    end;
station fee collector, etc.
```

The language elements "accept-select-if none," "hold," and "route-to" describe the nodes of the network and the interconnections. The syntax is that of ALGOL procedure calls, but according to Dahl and Nygaard (1963) "these are not procedures in the strict ALGOL sense. . . . Various details must be added behind the scenes by a preprocessor." For instance, the "hold" statement represented a time delay in the operation of the station, which implied that control would have to leave the station and return to it at a later time. Each new life cycle of a station would be started automatically at the appropriate times.

At the next stage, March 1964 (Dahl and Nygaard, 1964a), the simple network idea had been replaced by the more powerful concept of models consisting of interacting processes operating in "quasi-parallel" (see Section 2.3.2). The processes were declared collectively by **activity** declarations. At this stage the preprocessor idea had been abandoned, the language would be implemented by extending the ALGOL compiler and changing parts of the run time system (Dahl, 1963, 1964).

Process queues were now represented by explicitly declared ordered "sets." They were manipulated by "wait" and "include" statements and a special **extract–select** construct which also made quantities declared local to the selected process, the "attributes," accessible from the outside. Thereby processes were also data carriers, like the "customer" objects one year earlier.

```
SIMULA begin comment airport departure;
set q counter, q fee, q control, lobby (passenger);
    counter office (clerk); . . .
activity passenger; Boolean fee paid;
    begin fee paid := random (0, 1) < 0.5; . . . ;
        wait (q counter) end;

activity clerk;
begin
counter: extract passenger
    select first (q counter) do
    begin hold (normal (2, 0.3));
        if fee paid then
        begin include (passenger) into: (q control);
            incite (control office) end
```



```
        else
        begin include (passenger) into: (q fee);
            incite (fee office) end;

    end
    if none wait (counter office);
    goto counter
  end
    ⋮
end of SIMULA;
```

Curiously enough the wait-incite operator pair makes our set concept of 1964 act as a waiting mechanism not unlike the concept of condition variables introduced by Hoare for process synchronization in monitors (Hoare, 1974). (Hoare never saw our paper.) Our operator pair is analogous to Hoare's wait-signal.

At this stage all the major features of SIMULA I were present. Some important details were adjusted, however, along with the language implementation. In order to increase flexibility, sets were allowed to contain processes of different kinds, and process pointers were introduced as explicit language elements. Consequently the mechanism for accessing the attributes of a process had to be remodelled.

```
inspect ⟨process reference⟩
    when passenger do . . .
    when staff do . . .
    otherwise . . .
```

The **incite** operator was replaced by the more generally applicable construct

activate ⟨process reference⟩

The SIMULA I compiler was finished at the end of 1964, and the language definition user's manual appeared in May the following year (Dahl and Nygaard, 1965).

2.3. The Development Process

2.3.1. System Description

From the very outset SIMULA was regarded as a *system description language,* as evidenced by the title of the IFIP 1962 Munich paper, "SIMULA—an extension of ALGOL to the description of discrete event networks" (Nygaard, 1963a). The design objectives were stated in Nygaard (1963d, pp. 2–3).

1. The language should be built around a general mathematical structure with few basic concepts. This structure should furnish the operations research worker with a standardized approach in his description so that he can easily define and describe the various components of the system in terms of these concepts.
2. It should be unifying, pointing out similarities and differences between various kinds of network systems.
3. It should be directing, and force the operations research worker to consider all aspects of the network.
4. It should be general and allow the description of very wide classes of network systems and

other systems which may be analyzed by simulation, and should for this purpose contain a general algebraic and dynamic language, as for example ALGOL and FORTRAN.

5. It should be easy to read and to print, and should be suitable for communication between scientists studying networks.

6. It should be problem-oriented and not computer-oriented, even if this implies an appreciable increase in the amount of work which has to be done by the computer.

Two years later, in May 1965, the design objectives were restated in the SIMULA I Manual (Dahl and Nygaard, 1965, pp. 4–5):

1. Since simulation is the method of analysis most commonly to be used for the systems in which we are interested, SIMULA is a dynamic language:

It is designed to describe sequences of actions, not permanent relationships. The range of variation in decision rules and interactions between components of systems is so wide that it is necessary to let the language contain a general algorithmic language. An important reason why ALGOL has been chosen is that its block structure is similar to what was needed in SIMULA.

2. SIMULA should be built around a few basic concepts, selected to provide the research worker with a standardized approach to a very wide class of problems and to make it easy to identify the various components of the system.

3. Attempts have been made to make the language unifying—pointing out similarities and differences between systems, and directing—forcing the research worker to consider all relevant aspects of the systems. Efforts have also been made to make SIMULA descriptions easy to read and print and hence a useful tool for communication.

4. Taking the above objectives into account, SIMULA descriptions (supplemented by the necessary input, output and data analysis statements) should without any rewriting be able to produce simulation programs for electronic computers through compilers.

Comparing the two versions of the design objectives, it is seen that the three main differences are:

1. In 1963 SIMULA was related to "discrete event *network* systems." In 1965 the term "network" had disappeared.

2. In 1963 it was stated that SIMULA "should be built around a *general mathematical structure* with few basic concepts."

Also it was said in Nygaard (1963d) that "our present idea is that the first SIMULA compiler should be ALGOL-based, and ALGOL is used here. A later version may be FORTRAN-based, using the same basic concepts." In 1965 SIMULA's nature as a "*dynamic language*" (i.e. algorithmic language) and its relationship to the block structured programming language ALGOL was strongly emphasized.

3. In 1963 it was said that SIMULA "should be problem-oriented and not computer-oriented, even if this implies an appreciable increase in the amount of work which has to be done by the computer." In 1965 SIMULA's problem orientation was still stressed, but its computer orientation was also underlined.

Let us examine the reasons for each of these modifications of the design objectives.

When we started the development of SIMULA, we felt that we had a rather wide range of system examples available to test our ideas against. The situation was, however, that all these systems could be conceived as consisting of components of two distinct kinds: permanently present *active* components, and a variable number of transient *passive* components moving between and being acted upon by the active ones. Such a system could in a natural way be regarded as a network.

First we observed that, e.g., the airport departure system could be considered from a

"dual" point of view: It could be described by active passengers, grabbing and holding the passive counter clerks, fee collectors, etc. Then we realized that it was also possible to adopt an "in-between" or "balanced" point of view: describing the passengers (customers) as active in moving from station to station, passive in their interaction with the service parts of stations. These observations seemed to apply to a large number of situations. Finally, in our search for still wider classes of systems to be used to test our concepts, we found important examples of systems which we felt could not naturally be regarded as "networks" in the sense we had used the term (e.g., the "epidemic system" described in Dahl and Nygaard, 1966.) The result of this development was the abandonment of the "network" concept and the introduction of processes as the basic, unifying concept.

The second modification of the design objectives was related to the first. We no longer regarded a system as described by a "general mathematical structure" and instead understood it as a variable collection of interacting processes—each process being present in the program execution, the simulation, as an ALGOL stack.

From then on the program execution, existing as a dynamic system within the computer's store, became prominent in our discussions. Graphical representations of simplified (but structurally identical) versions of the program executions were used as *models* of the systems described by the language. More and more our reasoning on language properties was related to desirable features of these model systems. (The textbook *SIMULA BEGIN* is a later example of the systematic pedagogical use of such graphical models (Birtwistle *et al.*, 1973).)

It turned out that this approach was essential in the teaching of SIMULA I, and it was an important mode of thinking when SIMULA 67 was developed and later taught. Instead of deriving language constructs from discussions of the described systems combined with implementation considerations, we developed model system properties suitable for portraying discrete event systems, considered the implementation possibilities, and then settled the language constructs. An obvious consequence was that we abandoned the idea of a FORTRAN-based version of SIMULA I (see Section 2.5). ALGOL's stack structure had, in its generalized form, become an essential feature of SIMULA I and a main reason for its success.

Finally, let us consider the modification of the 1963 design objective, that SIMULA "should be problem-oriented and not computer-oriented, even if this implies an appreciable increase in the amount of work which has to be done by the computer." This design objective caused much discussion and disagreement between us. But, as the language gradually developed, we felt that the expected conflict between problem orientation and computer orientation diminished and to some extent disappeared. Instead we often discovered that, with the language approach chosen, good system description capabilities seemed to result in a more simple and logical implementation. Another reason for the modification was that we realized that the success of SIMULA would, regardless of our insistence on the importance of problem orientation, to a large extent depend upon its compile and run time efficiency as a programming language.

2.3.2. Storage Allocation

The initial plan was that the simulation facility should be implemented as a procedure package and a simple preprocessor on top of ALGOL 60. One idea that looked promising at the time came from the observation that ALGOL, by its recursive block mechanism, did cater for multiple occurrences of user defined data structures rather like the "custom-

ers'' that would go from one ''station'' to the next in our simulation models. Also the station descriptions had block format. It turned out, however, that the block structure of ALGOL was not very helpful, in fact the preprocessor would have to fight against it by breaking up the blocks, making local variables nonlocal, etc. There was no hope of integrating special purpose operations like ''hold (time interval)'' completely into the language, since it implied a kind of ''parallel'' processing foreign to ALGOL and conflicting with ALGOL's strict dynamic stack regime of procedure calls and storage allocation.

During the spring of 1963 we became more and more convinced that the project was on a wrong track, and started toying with the idea of making nontrivial changes to ALGOL by breaking with the stack regime. Since that would have grave consequences for the storage management of the ALGOL run time system, we had to dig from that end.

During the summer and autumn of 1963 a storage allocation package was designed, based on a two-dimensional free area list (Dahl, 1963). The inner lists contained areas of the same size, which we felt would be numerous in typical steady state situations; the outer list contained the inner ones ordered according to area size. Each area had a ''used'' bit in its first and last words to facilitate the recombination of neighbouring free areas. Thus the system had stack allocation as a special case, and could at the same time utilize the entire noncontiguous store of our computer.

With this solution to the storage allocation problem the search space for useful dynamic structures was drastically increased, and in February 1964 the SIMULA process concept was born, ranging from pure data structures to quasi-parallel ALGOL programs. 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 (Dahl, 1964). Furthermore processes could be created and destroyed in arbitrary order.

2.3.3. Security and Consistency

During the summer of 1963 Bernard Hausner, then working at RAND Corporation, Santa Monica, U.S., was employed by the NCC (see section 2.5). Hausner was one of the fathers of SIMSCRIPT (Markowitz *et al.,* 1963), and through him we got to know the SIMSCRIPT language and its implementation rather well. This was our first encounter with the pointer concept in a high-level language. To some extent, however, our reaction to SIMSCRIPT can best be described as that of a ''cultural clash.'' It helped to make an important design goal conscious and explicit.

Our language had to provide programming ''security'' to the same extent as ALGOL 60 itself: Any erroneous program must either be rejected by the compiler (preferably), or by run time checks (if unavoidable), or its behavior must be understandable *by reasoning based entirely on the language semantics,* independent of its implementation.

A main aspect of security was to achieve compiler controlled data access. As far as processes only interacting through nonlocal data were concerned, the problem was solved merely by applying the usual ALGOL access rules, with user convenience and computer efficiency thrown into the bargain (no subscripts needed to distinguish ''my'' local variables from those of other processes of the same kind). However, there was a need to obtain access to the contents of an object from outside the object. In a model containing ''customers'' and ''clerks'' the active agent would need access to its own data as well as those of the partner during ''service.''

The **inspect . . . when . . . when . . . otherwise** construct did provide the required compiler control, at the expense of run time tests to determine the type of the inspected object. However, the testing was turned into a potentially constructive language mechanism, rather than unproductive run time checking. Compiler control required that outside access be limited to the outermost block level of a process. This had the advantage that a process could prevent outside interference by hiding local data in inner blocks.

Another aspect of security had to do with de-allocation of storage. For reasons of implementation simplicity and efficiency one would like de-allocation to be explicit, say through a ''destroy'' statement, or self-destruction by control going through process **end.** However, the only way this could be combined with security would have been a process referencing regime essentially ensuring one pointer at a time to any process. Unable to find such a scheme providing sufficient programming flexibility we implemented a reference count scheme, an idea borrowed from Weizenbaum (1962), and also added a ''last resort'' garbage collector.

Automatic data retention was felt to be a fairly dangerous approach, in the sense that bad programs might easily lead to the flooding of memory by useless data, the resulting error message not giving many clues as to which pointers were responsible. To reduce that danger we insisted that procedures and ordinary subblocks should be self-destructive on exit, as they are in ALGOL 60. Combining these two different de-allocation strategies led to two possibilities of conflict with respect to data accessing security.

1. A process could outlive its dynamic parent, i.e., the block instance containing the generating expression which gave rise to the process. As a result the process might access nonexisting data through its formal parameters. The remedy chosen was to forbid all kinds of call by name parameters to processes (including procedures, labels, and switches), only excepting arrays which had to be allocated as separate objects anyway. The restriction was sometimes painful, but it did conform with the notion that a process breaks loose (becomes ''detached'') from its parent upon generation and starts a life of its own.
2. A process could outlive its textually enclosing block instance (by being pointed to by data nonlocal to the latter), thereby accessing nonexisting nonlocals. A remedy was to require that all processes be declared (by **activity** declarations) local to a special block, identified by the prefix **SIMULA.**

SIMULA begin . . . end

Furthermore the SIMULA block must be the outermost one or must be embedded in an *ordinary* ALGOL program. This latter requirement was enforced by the following compiler stratagem: the special SIMULA vocabulary was part of the compiler dictionary only inside a SIMULA block, with the single exception of the word **SIMULA;** the latter was removed from the dictionary inside a SIMULA block. (It would still be possible to have dynamically nested instances of a SIMULA block, by embedding it in a recursive procedure. But to our knowledge nobody has ever found a use for this construct, or even checked that the compiler could handle it properly).

The SIMULA block would correspond to the simulation model as such. It was not unnatural to require that processes, running in quasi-parallel, should also be declared "in parallel" and as direct components of the model.

A final aspect of security concerned the ALGOL rule that the value of a variable is undefined upon declaration. With reference variables in the language (see below) it was very clear that "undefined values" would have to be treated explicitly and cautiously by the implementation. The best way we could find of combining reasonable efficiency with full programming security was to revise ALGOL on this point and assign neutral initial values to all variables. Fortunately this revision could not change the behavior of any correct ALGOL program.

2.3.4. Process Referencing

The concept of "process sets" was planned to be a prominent feature of the language, together with associated scanning mechanisms and with "selector expressions" as the only means of process identification. In this respect we were influenced by the design of SIMSCRIPT; certainly the term "set" was borrowed from that language, in SIMULA I meaning "ordered set of processes." It was questions of efficiency and algorithmic generality that made us abandon that approach and settle for "process pointer" as a basic data type.

As the language and its implementation took form, efficiency issues came more into the foreground. Not only should the language implementation be efficient, but the language itself must be such that users were *invited to make efficient programs*. In particular all built-in mechanisms ought to have execution times independent of model size.

One of our standard examples was a queue of high- and low-priority customers. Rather than having any easy-to-use selector expression that would search the queue for a priority customer, it would be better to maintain separate queues for the two kinds of customers. Thereby customer selection would become independent of queue size.

Still, some kind of built-in list mechanism was required for queueing and similar purposes, so an abstract ordered set concept was included as a second new data type, implemented as two-way circular lists. It seemed attractive at the time to emphasize the "abstract" nature of the set concept and enable processes to be members of arbitrarily many sets at the same time. This could be achieved by using auxiliary "element" objects to represent a process in different sets. At the same time all process references were made indirect through element objects by providing only "element pointers" in the language. Thereby "successor" and "predecessor" functions could be implemented without implicit searching.

No special scan/select mechanisms were included, except the natural extension of the ALGOL **for** statement to control pointer variables. Unfortunately the ALGOL **while** construct insists on advancing the controlled variable before testing it, which meant that the "set head" had to come out into the open and make our sets look less abstract than we had intended.

```
set S; element X; . . .
X := head(S);
for X := suc(X) while exist(X) do
      inspect X when . . . do . . .
```


Admittedly the inspect construction was a clumsy and fairly inefficient device for taking brief looks at objects in passing. However, we rationalized by referring to the "invitation-to-efficiency" principle: the clumsier the better, searching is abominable anyway.

In retrospect the introduction of "multimembership" sets was a mistake. First the "sets" were really process sequences allowing multiple process occurrences, whereas simple process chains would have been more appropriate for most purposes. Second, natural abstract set primitives like $P \in S$ and $S := S - \{P\}$ for given process P and set S were not efficient operations for the chosen set representation. So, contrary to good principles, functions like "member (P, S)," searching S for an element representing P, found their way into the language as built-in procedures. Third, there was an ever-present overhead in process referencing caused by the fact that all process pointers were directed through separately allocated "element" objects.

2.3.5. Process Scheduling

If (avoidable) searching in space is bad, then searching for the correct point in time is even worse. The May 1963 language specifications (Dahl and Nygaard, 1963) contain a statement

PAUSE (⟨Boolean expression⟩)

to that effect (awaiting the truth of the Boolean expression). The idea was quietly buried, and we made do with *passivate, activate, hold,* and *cancel* statements as described in Dahl and Nygaard (1964a). There was *direct* activation for immediately invoking the next active phase of a process, comparable to a procedure call and activation with *time delay* (including *hold*) not unlike the CAUSE–AT mechanism of SIMSCRIPT.

A simple way to implement model time scheduling is to maintain a list of scheduled processes sorted by time attributes. The list (we called it the "sequencing set," SQS) was made to look like a generalized ALGOL activation stack by an invariant stating that the currently operating process is at the end of the SQS. (And its time attribute is the current model time). Admittedly scheduling with a time delay is then not an $O(1)$ operation, and to improve efficiency, the SQS was represented by a kind of binary tree (left heavy, postordered). The tree preserved the order of elements with equal time values and had $O(1)$ algorithms for removal and for LIFO and FIFO insertion. Worst-case insertion was $O(n)$, but according to experiments (Myhrhaug, 1965), the "average" behavior was much better.

Recently the insertion algorithm was analyzed with respect to exponentially distributed time delays (Jonassen and Dahl, 1975), and the average performance in that case was found to be $O((\ln n)^2)$.

The decision, made early 1964, not to include any mechanism for "interrogative sequencing" (Dahl, 1968b), was a fairly natural one at the time. It was based on considerations of efficiency and ease of implementation, and on the growing conviction that the language must give the user full sequencing control in order to be a sufficiently general purpose modeling tool.

As a result of discussion with colleagues, especially those from the CSL group (Buxton and Laski, 1962), it became clear that the case for interrogative sequencing was stronger than we had originally realized (see, e.g., Blunden, 1968), compared to the "imperative" sequencing of SIMULA (and SIMSCRIPT). The dilemma may be explained as a choice between a statement

await (⟨Boolean expression⟩)

in process **P**, and a **passivate** statement in P together with matching occurrences of **activate** P in other processes. Advantages of the former are:

1. It is obviously easier to use, and more self-documenting.
2. It leads to better program decomposition in terms of processes; process P is made more self-contained.
3. Properly implemented it protects the user from making the error of forgetting necessary occurrences of **activate** P. This kind of programming error is especially nasty since the observable consequences are negative: events that should have happened in the model do not take place.

Yet, we feel that our design decision was the right one, for the following reasons:

1. The notion of waiting until a given condition becomes true is not well defined within the framework of quasi-parallel processes, since only one process may run at a time. Thus, any realization can only approximate the idea, which means that sequencing decisions that ought to be in the hands of the user, will have to be arbitrarily made by the implementation. A ''good'' implementation is likely to be so complicated that the exact model behavior is hard to predict from the program text in complex cases.
2. There is no a priori upper limit to the cost of executing an **await** statement. The cost is likely to increase with the size of the model (as is true for searching in space too).

From 1 and 2 we draw the conclusion that interrogative sequencing should not be among the language primitives. Whether it should be added as an auxiliary mechanism on top of the language is perhaps a question of time, money, programming skill, and taste. Fortunately one has learned to achieve program decomposition (cf. point 2 above) by isolating sequencing strategies as separately verifiable *class*-like program components, representing such concepts as abstract resource schedulers, communication channels, or even interrogative schemes. (Compare the monitor concept of Hoare (1974) for parallel programming.)

Our attitude and feelings towards interrogative sequencing has been conditioned by experience and perhaps cultural factors. Two incidents from the year 1965 are worth recounting. At the IFIP Congress 65 in New York, May 24–29, 1965 (Nygaard and Dahl, 1965), we were able to show results from our first real-life simulation model (of a paper mill lumbering yard). For the first time we could exchange information on typical simulation execution speeds and found that SIMULA I's performance was very much better than what a number of people in that audience were accustomed to.

Some of the first SIMULA programs written outside our center were part of Univac's acceptance tests for our compiler. One of the programs, a model of the job flow through an operating system, appeared to go into a tight loop. After running it for 15 minutes while nothing happened we were convinced that another bug had been discovered in the run time system. It turned out that the program was built around a loop of the form

```
wait: if nothing has happened then
      begin hold (one drum cycle);
      goto wait end
```

According to the program, the computer was switched on at 8 A.M., and the data provided the first job arrival at 9 A.M. Since the drum cycle was 34 milliseconds, approximately

100,000 events had to be executed before anything happened. After some simple reprogramming a whole day's work was simulated in a few minutes of computer time, which goes to show that the ''invitation-to-efficiency'' may well be turned down by language users.

2.4. Relation to Other Languages

SIMSCRIPT was the only simulation language that we were closely acquainted with during the design phase of SIMULA. From the preceding sections it will be evident that it had a considerable impact through its list processing and time scheduling mechanisms. It also contained a set of random drawing and other utility routines, which served as a model for our procedure library. Information on GPSS (Gordon, 1962) was available to us through IBM Norway, but the GPSS system looked more like a generalized simulation model than a programming language. Consequently we did not study it very closely. Only later did it occur to us that the ''transactions'' of GPSS could in fact be looked upon as processes in quasi-parallel. Tocher's work on GPS (e.g., Tocher, 1963), gave us a greater awareness of some of the practical considerations and difficulties of large scale simulation. However, his system design appeared to be too specialized. SOL (Knuth and McNeley, 1964a, b) came to our attention too late (July 1964) to have an effect on our design, but we were impressed with its beauty and realized that others before us had had the idea of quasi-parallel processes in an ALGOL-like setting.

At the time when we offered our SIMULA introduction paper (Dahl and Nygaard, 1966) to the ACM, October 1965, Don Knuth was serving as the programming language editor. He wrote us a very generous letter, typical for him, which added to our pride and became the starting point of a long and lasting friendship. Other good friendships resulted from contacts and discussions with the CSL designers John N. Buxton and John Laski.

By far the most important language ancestor of SIMULA I is of course ALGOL 60 itself. Through its orthogonal design, concept economy, strictness of definition, and degree of compile time error control it set a standard of quality that we could only hope to approach, but which was well worth striving for. The concept central to us was the dynamic block structure. In retrospect the two main lessons were:

1. The distinction between a piece of program text and an execution, or ''dynamic instance'' of it.
2. The fact that data and operations belong together, and that most useful program constructs contain both.

In ALGOL, blocks (including procedures) are seen externally as generalized operations. By introducing mechanisms for quasi-parallel sequencing, essentially the same construct could play the role of processes in parallel, and through mechanisms for naming block instances and accessing their contents they could function as generalized data objects. The essential benefits of combining data and operations in a single construct were already there to be explored.

One result of this exploration was the discovery that ''procedure attributes'' might be useful. The following example of a class of ''abstract'' car objects is quoted from the Language Definition document (Dahl and Nygaard, 1965), Section 5.3:

```
activity car;
begin real V, X0, T0;
      real procedure X; X := X0 + V * (time - T0);
      procedure update (Vnew); real Vnew;
      begin X0 := X; T0 := time; V := Vnew end;
      ⋮
end;
```

The attributes X, V and update were used by a police survey process to enforce a speed limit on a given road section. (It is perhaps a pity that the representation variables X_0 and T_0 could not be hidden away in a subblock.) Another discovery was the fact that SIMULA had become a powerful list processing language. A second example in the same document defines a process for scanning the leaves of a tree, advancing to the next leaf with each activation.

It should be mentioned that D. Ross, through his AED project (Ross and Rodriguez, 1963), independently and unknown to us had developed an ALGOL-like language also exploiting the idea of adding operators to data structures.

One factor which contributed to the comparative success of the SIMULA I project was the fact that we had a good ALGOL compiler to start with. It had been developed at Case Institute of Technology in Cleveland by J. Speroni, W. Lynch, N. Hubacker, and others. They had extended the ALGOL language by a rather nice I/O system, including an elaborate generalized parameter list mechanism. The compiler was finished and installed in Oslo late spring 1964 (which was in the nick of time for our project schedule). Our implementation effort amounted to about one man-year's work on the run time system, including built-in procedures, and one man-month for extending the compiler.

2.5. The Software Agreement between Univac and NCC

In the second half of May 1962, Univac arranged the "Univac Executive Tour." A Douglas DC-8 was filled with prospective customers who were shown Univac's new computers and production facilities. KN was invited to participate by Stig Walstam, Director of Univac Scandinavia. KN came into contact with James W. Nickitas, then Assistant to Luther Harr, director of Univac Europe. Nickitas was told about SIMULA and also about another NCC software project: a linear programming package based upon a new algorithm developed by Sverre Spurkland at NCC. Nickitas immediately arranged a meeting at a hotel in New York for himself, KN, and three important people within Univac's software activities: Alvin M. Paster (Manager, Systems Research), his boss Robert W. Bemer (Director, Systems Programming), and Bemer's boss, William R. Lonergan. In the setting of the Executive Tour, a certain measure of polite interest was of course to be expected. But Paster, Bemer, and Lonergan turned out to be really interested, both in SIMULA and the LP package. (They knew that NCC recently had contracted a Danish GIER computer and would not buy Univac equipment.) Bemer at once invited KN to present SIMULA at the session which he was to chair at the IFIP 62 Congress in Munich. When the Executive Tour arrived at St. Paul, KN was very impressed by the brand-new UNIVAC 1107, and this both pleased Nickitas immensely (he was deeply in love with that computer himself) and also gave him another idea which he started to work on.

On May 29 the Executive Tour had arrived in Washington, D.C. After a dinner at the

Mayflower, Nickitas, who is of Greek descent, invited KN to a Greek nightclub. While they were listening to bouzouki music, watching a beautiful belly dancer, Nickitas presented the following informal proposal: Univac needed in the near future a good UNIVAC 1107 demonstration site in Europe. If NCC would be willing to provide Univac with SIMULA and the LP package, Univac would be willing to sell the 1107 at a substantial discount. When KN returned and told this story, most people raised serious doubts as to the state of his mind. In the end of June, however, Luther Harr, Nickitas and Walstam turned up in Oslo and presented the offer officially at a meeting with the NCC board. During the meeting, it became clear that Luther Harr either was in a very generous mood, or he had not read his lesson sufficiently well and did not realize that SIMULA and the LP were to be a part of the payment for the 1107. KN then asked him if Univac was definite in their decision to offer NCC a software contract for these items, and this was confirmed. Nickitas was foaming, but could do nothing. Afterwards, Nickitas took the incident with grace and became a very close friend of us as well as NCC. But Univac was from then on not too happy when money was discussed in relation to SIMULA.

After a summer of intensive computer evaluation studies the GIER contract was cancelled and UNIVAC 1107 purchased by the NCC. Alvin M. Paster became Univac's man in the subsequent software contract negotations. Univac's letter of intent was dated October 12, 1962, and it also contained Univac's first contract proposal (Paster, 1962). The contract work was delayed because of the long communication lines (the contract was with Univac headquarters, New York) and because NCC had internal troubles associated with the very rapid expansion necessary to take over a large scale computer.

Another factor also complicated the picture: Univac decided to support an alternative basic software package for the 1107 ("Package B" as opposed to the original "Package A"). KN was sent to the US in November 1962, as a representative for the European 1107 users to make an evaluation of the situation, which was critical also for SIMULA: what would the quality be of the proposed new Package B ALGOL 60 compiler, to be developed at Case Institute of Technology in Cleveland? (The Package A ALGOL 60, later to be dropped, was contracted from General Kinetics Inc.).

KN returned as a convinced Package B supporter, and had established very useful contacts with Joseph Speroni and others participating in the Case ALGOL 60 project. In Los Angeles he had lectured on SIMULA at the RAND Corporation and got Bernie Hausner interested. Hausner was the chief implementor in the SIMSCRIPT team (Markowitz *et al.*, 1963). He was promptly offered a job at the NCC, which he later accepted, and arrived on July 1, 1963.

At a meeting at NCC on May 2, 1963, Univac to our surprise confronted us with a proposal for an extension of the contract: we should also implement a SIMSCRIPT compiler. The reason was that Univac had met a US market demand for SIMSCRIPT. They did not lose interest in SIMULA, but used Hausner's presence at NCC to force us into SIMSCRIPT as well, as an "initial step" towards implementing SIMULA. (It was believed, also by us, that we could use important parts of a SIMSCRIPT compiler as a platform for SIMULA.) Another awesome new person now also entered the scene: Mr. C. A. Christopher, Univac's Director of Procurement. We never met him, but his presence was very much felt as the representative of the world of law, money and vast administrative hierarchies. Our first encounter with this new world was the first page of his contract text proposal, where we read:

WITNESSETH:

WHEREAS, Univac desires to have a program developed and implemented which will solve Linear Programming Problems, a Simscript Compiler developed and implemented, and a SIMULA Compiler developed and implemented, all for a UNIVAC 1107 Computer System, and

WHEREAS, NCC is willing and able to design, develop, implement, check out and document such aforementioned programs for the stated purposes on the terms and conditions hereinafter set forth.

NOW, THEREFORE, in consideration of the premises and of the mutual convenants herein contained, the parties hereto agree as follows.

The contract text was worked out and agreed upon. It specified Robert L. Hengen as the Technical Supervisor and C. A. Christopher as the General Contract Administrator for Univac, KN serving in both these functions for NCC. Hengen was, however, substituted by Alvin M. Paster. Paster gradually got other responsibilities and Hengen returned in February, 1964. The contract was dated June 1, 1963, signed June 1 by the NCC and July 3 by Univac (Agreement, 1963).

The SIMULA part of this contract clearly describes (in Article III B) the SIMULA compiler as a preprocessor to the ALGOL compiler. It also has our language specifications of May 18, 1963 (Dahl and Nygaard, 1963) as an appendix, defining the SIMULA language. It states that variations in the language have to be approved by Univac. The payment was $35,000, according to a payment schedule which assumed that the SIMSCRIPT part of the contract should be completed first. It is also implicitly assumed that the SIMULA compiler should be ready for acceptance tests by July 3, 1964 (366 days after the contract was signed). The contract gave NCC 60 days maintenance responsibility after it had been accepted by Univac, and it described the details of the acceptance procedure.

The SIMSCRIPT compiler was to be completed very quickly, in five months, for the sum of $25,000. Already in August–September 1963 it became clear that it would not be sensible to transform the SIMSCRIPT IBM 7090 compiler into an 1107 compiler. NCC immediately told this to Paster during a visit at NCC late September, and Univac agreed to terminate this section of the contract. During this visit Univac got our proposal of making a SIMULA compiler with a new storage management scheme, and not only a preprocessor and procedure package in ALGOL. Communication between Univac and NCC was problematic, and for various reasons our work was delayed. In February 1964 the terms of the contract were modified (Christopher, 1964). SIMSCRIPT was officially terminated. The delivery date was changed to January 1, 1965. The payment schedule was revised: $33,000 had already been paid ($8000 for SIMSCRIPT), $20,000 was to be paid by March 31, 1964, and $30,000 was to be paid upon Univac's acceptance of the SIMULA and LP products. (The main reference to the history of the contract work is Nygaard (1965a).)

In February 1964 the SIMULA language could be reconsidered with the freedom made available by the new storage management scheme developed in the autumn by OJD. The process concept was developed and reported in March (Dahl and Nygaard, 1964a). From then on only minor language revisions were made.

A major problem encountered was the late arrival of the Case ALGOL 60 compiler (May–June 1964) and the lack of suitable documentation, particularly relating to its interface with the EXEC II operating system. Two American visitors during the summer of 1964 provided useful assistance in this situation: Ken Walter from Purdue and Nicholas Hubacker from Case. At the NCC Bjørn Myhrhaug and Sigurd Kubosch were members of the SIMULA team. The work during the summer and autumn went on with much effort

and little drama. Progress reports were forwarded to Univac as before, but we only got a reaction once (through a Telex) when we urgently requested an acknowledgment of the arrival of our reports. Our impression was that the personnel with knowledge of the contracts left Univac and did not inform those who took over their responsibilities. (We got, however, the scheduled March 1964 payment of $20,000).

When the SIMULA I compiler was ready, on the date specified by the revised contract, January 1965, Univac was notified, but we got no reaction. During our visit to the US in May 1965, in connection with the IFIP 65 World Congress, we had a meeting with Ira A. Clark (Technical Coordinator for Systems Programming, Univac, St. Paul) and W. J. Raymond (Systems Programming Manager, Univac International Operations, New York).

Clark's attitude toward SIMULA was initially very negative, and for good reasons. It turned out that he, who had got the responsibility for handling the contract, had never seen our letter. Later on it was discovered, we were told, in an abandoned desk at the Univac New York headquarters.

Clark later on was very helpful as our contact during the acceptance tests which followed. Univac forwarded their test problems to be run in Oslo on October 27, 1965. On January 10, 1966, a Telex informed us that Univac was accepting SIMULA I, and a letter from Ira A. Clark (dated January 27, 1965) said

> We have examined the results of test cases submitted as acceptance runs for the SIMULA compiler. In every case, the compiler performed successfully and useful results were obtained. The personnel in the Center were very helpful in doing everything necessary to analyze our test cases, insure the accuracy of our work, successfully compile these SIMULA tests, and provide helpful comments and listing annotation of output.

SIMULA I was for some time Univac Category III Software. This implied that Univac distributed the compiler, but NCC had the maintenance responsibility. When NCC had completed a comprehensive technical documentation, SIMULA I was (in May 1967) made Category I software. (NCC had a maintenance agreement with Univac until Univac adapted the SIMULA I compiler to its EXEC-8 operating system).

In spite of our efforts, Univac had lost interest in our LP package, and a last letter from C. A. Christopher (1965, Dec. 15) stated: "To date we have made payments to you of $53,000. I realize that time and effort were put into the other projects but with negative results. We feel that the total paid NCC to date represents full and adequate reimbursement for the SIMULA Compiler."

2.6. The Response to SIMULA I

At the NCC the use of SIMULA I for simulation programming started immediately and spread rapidly. In 1965 three SIMULA I courses were given at NCC. The use of SIMULA I up to December 1965 is reported in Nygaard (1965b). A later version of this report covers the period up to June 1968 (Hegna *et al.*, 1968).

A visit to Stockholm in February 1965 triggered off an important development. We were lecturing three days at the Royal Swedish Institute of Technology and were pleased to note the first day that two representatives from the big Swedish company ASEA seemed to be very interested. The second day we were disappointed to find that the ASEA people did not turn up. The third day they returned however, Niels Lindecrantz of ASEA bringing an offer: ASEA would consider using SIMULA for a number of jobs if we were willing

to do a test job free of charge. We were requested to program and run a large and complex job shop simulation in less than four weeks, with a simulation execution efficiency which was at least four times higher than that job on the FACIT computers. We accepted, succeeded and got the other jobs. (A simplified version of this program is one of the main examples used in Dahl (1968b).)

In addition to the money from Univac, NCC got a number of other benefits. Our Operational Research Department could take on jobs which otherwise would have been impossible to accept. Customers got important problems solved. The first SIMULA-based application package, for simulation of logic circuits, was implemented (Stevenson, 1967). The NCC developed international connections, and we got into direct contact with many of the other research workers developing programming languages, particularly simulation languages, at the time. We had at an early stage ideas for using SIMULA I as a tool in real-time programming, but these plans never materialized (Nygaard, 1963e, in Norwegian).

Burroughs was the first of the other computer manufacturers to take an interest in SIMULA I, for two reasons:

the advocacy of Don Knuth and John L. McNeley, the fathers of SOL,
the general ALGOL orientation within the company.

In 1968 SIMULA I was made available to Burroughs B5500 users. The language was modified and extended, the reasons being discussed in a letter from John S. Murphy of Burroughs, Pasadena (Murphy, 1968).

Another early interest in SIMULA I developed in the Soviet Union, reported in Nygaard (1968b). The main center of activity was the Central Economical Mathematical Institute of the Russian Academy of Sciences in Moscow (CEMI). The CEMI Computer Science department was headed by E. I. Yakovlev. KN was invited to the Soviet Union twice in the summer of 1966, and gave one-week courses in Moscow, Leningrad, Novosibirsk, and some lectures in Kiev. A number of reciprocal visits resulted and a cooperative agreement was in effect for some time. OJD and Myhrhaug lectured on SIMULA I and SIMULA 67 implementation in Moscow, and a team in Yakovlev's department later implemented SIMULA I on a URAL 16 computer. The project leader was K. S. Kusmin. Implementation on the BESM 6 computer was discussed but never carried out. The SIMULA I manual (Dahl and Nygaard, 1965) was translated into Russian.

During the first years of SIMULA's life, the NCC had to do most of the teaching of the language. We soon discovered that this was not a trivial task. As a result we developed the pedagogical approach in which the process concept was the first one introduced, then the reference (''element'') concept and informal versions of the statements followed. Procedures and formalized versions of the statements were introduced later. Graphical models were used extensively (see Section 2.3.1).

Those familiar with FORTRAN very often had difficulties in unlearning that language. Those familiar with ALGOL were better off, but still found it problematic to substitute ALGOL's single-stack picture of the world with SIMULA's multistack picture. Quite often the newcomers to programming seemed to absorb SIMULA faster than the oldtimers.

When SIMULA I was put to practical work it turned out that to a large extent it was used as a system description language. A common attitude among its simulation users seemed to be: Sometimes actual simulation runs on the computer provided useful information. The writing of the SIMULA program was almost always useful, since the development of this program (regarded as a system description) resulted in a better understanding of the system. Semiformalized SIMULA programs, with the input, output, and data analysis statements omitted, proved to be useful in discussing the systems' properties with people unacquainted with programming.

SIMULA was intended to be a system description and simulation programming language. Some users discovered that SIMULA I also provided powerful new facilities when used for other purposes than simulation. After the first introduction phase we became more and more interested in this use of SIMULA I, and we soon discovered a number of shortcomings within the language. The resulting discussions on the possible improvements to SIMULA I in 1965–1966 initiated the development of SIMULA 67.

3. SIMULA 67

3.1. From SIMULA I to SIMULA 67

During 1965 and the beginning of 1966, most of our time went to using and introducing SIMULA I as a simulation language. Tracing facilities were designed and implemented (Dahl *et al.*, 1966), but were never used very much (at the NCC). When the first large jobs had been successfully completed and a number of users had learned the language, we became more and more interested in the possibilities of SIMULA I as a general purpose programming language. A first reference to a ''new, improved SIMULA'' occurs in a letter dated September, 1965 (Nickitas, 1965). We explored SIMULA I's list structuring and coroutine capabilities, the use of procedure attributes etc. Gradually we realized that a number of shortcomings existed in the language.

1. The element/set concepts were too clumsy as basic, general purpose mechanisms for list processing. Even for simulation modeling our experience showed that simple process pointers might be better, combined with an inherent set membership capability of processes, restricted to one set at a time.
2. The inspect mechanism for remote attribute accessing turned out to be very cumbersome in some situations. (Try to compute X.A+Y.A using inspect statements.) Then Hoare's record class proposal appeared (Hoare, 1965, 1966, 1968), which showed how full security could be obtained in constructs like X.A by compile time reference qualification, and how reasonable flexibility of run time referencing could be obtained by the idea of record subclasses.
3. We were beginning to realize that SIMULA I's simulation facilities were too heavy a burden to carry for a general purpose programming language. Certainly the multistack structure was a big advantage, but quasi-parallel sequencing had many applications independent of the concept of simulated time.
4. We had seen many useful applications of the process concept to represent collections of variables and procedures, which functioned as natural units of programming although they had no ''own actions.'' It occurred to us that the variables of such an object could play the role intended for the not very successful **own** variables of ALGOL 60, since they survived individual procedure activations. In our experience, however, such data would often be associated with a group of procedures rather than just one. The difficulties inherent in the **own** variable concept were related to generation and initialization. However,

SIMULA objects were generated by explicit mechanisms, and initializing actions could be naturally assigned as ''own actions'' of the object.

5. When writing simulation programs we had observed that processes often shared a number of common properties, both in data attributes and actions, but were structurally different in other respects so that they had to be described by separate declarations. Such partial similarity fairly often applied to processes in *different simulation models,* indicating that programming effort could be saved by somehow preprogramming the common properties. Parametrization could not provide enough flexibility, especially since parameters called by name, including procedure parameters, had been banned for processes (for good reasons, see Section 2.3.3). However, the idea of subclasses, somehow extended to apply to processes, might prove useful.

6. We were itching to revise the SIMULA implementation. The Univac ALGOL compiler, although efficient for most ALGOL programs, was terribly wasteful of storage space whenever the number of process activation records was large, as it would be in most simulation models. This made memory space our most serious bottleneck for large scale simulation. Jan V. Garwick, back from work in the US, had shown us a nice compacting garbage collector (now well known, said to have been designed for the implementation of a language called LISP 2). Some experimentation indicated that it was more efficient than our combined reference count/garbage collecting scheme. Furthermore it could take advantage of active deallocation at exit from procedures and blocks, simply by moving the free pointer back whenever the deletion occurred at the end of the used memory. Thus, ALGOL programs could be run with hardly any overhead at all, and most SIMULA programs would benefit too.

Our discussions during the spring and summer of 1965 had been rather academic: what should be revised *if* a revision was to be made. In the autumn of 1965 the Technical University of Norway in Trondheim contacted NCC and expressed its interest in implementing a new ALGOL 60 compiler on the UNIVAC 1100 series. The possibilities of basing a SIMULA compiler upon an ALGOL 60 compiler designed with SIMULA in mind seemed attractive. From February 1966 the term ''SIMULA II'' started to appear in our correspondence.

We started at that time a cooperation with a team headed by Knut Skog and Kristen Rekdal at the Technical University. NCC's part of that work faded out from December 1966 onwards, when the SIMULA 67 ideas were developed. The compiler was completed by the Technical University and marketed under the name ''NU ALGOL.''

This was the background for our language discussions during the autumn of 1966. All six points listed above were motivating us, but in retrospect it appears that points 2 and 5—attribute accessing and common properties of processes—were the most important ones. That is, important in the sense that our work to resolve these problems resulted in the class/subclass concepts which structured the rest of the new language. The subclass idea of Hoare (1968) was a natural starting point, but there were two difficulties:

1. We needed subclasses of processes with own actions and local data stacks, not only of pure data records.
2. We also needed to group together common process properties in such a way that they could be applied *later,* in a variety of different situations not necessarily known in advance.

Much time was spent during the autumn of 1966 in trying to adapt Hoare's record class

construct to meet our requirements, without success. The solution came suddenly, with the idea of ''prefixing,'' in December 1966. We were thinking in terms of a toll booth on a bridge, with a queue of cars which were either trucks or buses. (This example reappears in Dahl and Nygaard, 1968.) A ''stripped'' list structure, consisting of a ''set head'' and a variable number of ''links,'' had been written down, when we saw that both our problems could be solved by a mechanism for ''gluing'' each of the various processes (trucks, buses) on to a ''link'' to make each link-process pair *one block instance*. Such a language feature would not be difficult to implement. Now each of the processes in the example would be a block instance consisting of two *layers:* A *prefix layer* containing a ''successor'' and ''predecessor,'' and other properties associated with two-way list membership, and a *main part* containing the attributes of either a truck or a bus. In order to obtain compiler simplicity and, at the same time, security in attribute referencing, it was necessary that the two-layer property of these processes be known at compile time and that the prefix and main part be permanently glued together into one block instance.

The syntax for this new language feature was easy to find. The ''links'' could be declared separately, without any information about the other process classes which used link instances as a prefix layer. Since the processes of these other process classes were at the same time both ''links'' and something more, it was natural to indicate this by textually prefixing their declarations with the process class identifier of this common property, namely, ''link.'' These process classes would then be ''subclasses'' of ''link.''

It was evident that when prefixing was introduced, it could be extended to multiple prefixing, establishing hierarchies of process classes. (In the example, ''car'' would be a subclass of ''link,'' ''truck'' and ''bus'' subclasses of ''car.'') It was also evident that this ''concatenation'' of a sequence of prefixes with a main part could be applied to the action parts of processes as well. Usually a new idea was subjected to rather violent attacks in order to test its strength. The prefix idea was the only exception. We immediately realized that we now had the necessary foundation for a completely new language approach, and in the days which followed the discovery we decided that:

1. We would design a new general programming language, in terms of which an improved SIMULA I could be expressed.
2. The basic concept should be *classes* of *objects*.
3. The prefix feature, and thus the subclass concept, should be a part of the language.
4. Direct, qualified references should be introduced.

The development of the first SIMULA 67 proposal, based upon these decisions, is described in the next two sections.

3.2. The Lysebu Paper

The IFIP Technical Committee 2 (on programming languages) had decided in the autumn of 1965 that an IFIP Working Conference on simulation languages be held in Oslo in the spring of 1967. The decision was the result of a proposal made by the Norwegian representative to TC2, at the time, OJD. At New Year 1967 preparation had been under way for more than a year, and the time and place of the conference had been fixed for May 22–26 at Lysebu, a resort in the hills outside Oslo. We naturally hoped to be able to complete the design of our new language in time for the Lysebu conference. According to the conference schedule, preprints of all papers were to be distributed to the participants at

the end of March. Consequently we were in a hurry, but fortunately the most difficult work had already been done, and the initial version of the paper (Dahl and Nygaard, 1967a) was ready in time. We had a fairly clear idea of how to unify the old processlike objects and the new concept of self-initializing data/procedure objects (Section 3.1, point 4), and at the same time remove the special purpose ''model time'' concept from the former. Since the term ''process'' could not apply to the unified concept, we introduced the more neutral word ''object'' as a technical term.

An object would start its life like an instance of a function procedure, invoked by the evaluation of a generating expression. During this phase the object might initialize its own local variables. Then, on passage through the **end** of the object or as the result of a new basic operation ''detach,'' control would return to the generating expression delivering a reference to the object as the function value. In the former case the object was ''terminated'' with no further own actions, in the latter case it had become a ''detached object'' capable of functioning as a ''coroutine.'' The basic coroutine call **resume** (⟨object reference⟩) would make control leave the active object, leaving behind a reactivation point at the end of the resume statement, and enter the reference object at its reactivation point. (The Lysebu paper mentions a third primitive operation ''**goto** (⟨process reference⟩)'' terminating the active object, but on this point revisions were made later.)

A declaration giving rise to a class of objects might well have been called an *object class* (in analogy with Hoare's *record class*). In choosing the shorter term **class** we felt that we had a good terminology which distinguished clearly between the declared quantity (the class) and its dynamic offspring (the objects). Our good intentions have not quite worked out, however. Many users tend to use the term *class,* or perhaps *class instance,* to denote an object, no doubt because *object* does not occur as a reserved word of the language. (As an afterthought, around 1978, we might have insisted that all class declarations must be prefixed, and defined a primitive outermost prefix *object* containing detach and resume as local procedures. (See also Wang and Dahl, 1971.))

The idea of class prefixing and concatenation was the most important step towards our goal. It had become possible to define classes primarily intended to be *used as prefixes*. Our idea was that some of the special purpose concepts of SIMULA I could be expressed by such ''prefix classes'' available to the programmer as plug-in units.

It was easy to describe ''circular list processing'' (like ''sets'' in SIMULA I) by means of a little class hierarchy for list elements (**class** link) and list heads (**class** list, later called ''head''), with a common prefix part containing forward and backward pointers. Now any class prefixed by ''link'' would give rise to objects that could go in and out of circular lists, using procedures like ''into'' or ''out'' declared within its prefix part together with the list pointers. The lists themselves would be represented as ''list'' objects, possibly augmented by user defined attributes.

In order to explain the process concept as a prefix class it became necessary to extend the concatenation mechanism slightly. The original rule was that the operation rule of a concatenated class consisted of the operation rule of the prefix class followed by that of the main part. For a process object it was necessary to have predefined actions both at the beginning and at the end of its operation rule. So the prefix class had to be given a ''split body'' whose operation rule consisted of initial actions and final actions textually separated by a special symbol **inner.**

Now the task was simple. The prefix class was named ''process,'' which meant that the term **activity** of SIMULA I would be represented by the much more descriptive term

''process **class.**'' The Lysebu paper shows a class ''process'' containing a **real** variable ''evtime'' representing model time, and a **ref** (process) variable ''nextev'' used to describe the SQS as a one-way list of process objects. (In an actual implementation we intended to use the logically equivalent binary tree technique, mentioned in Section 3.2.5). Obviously the sequencing primitives of SIMULA I could be represented by procedures manipulating the SQS, keeping it sorted with respect to evtime values, and passing control by resume operations. The ''final actions'' of a process were needed to remove the now dying object from the SQS and pass control to its successor on the SQS. Now only two problems remained: where to store the SQS pointer, and how to represent the ''main program'' of a simulation model.

We had independently, while exploring the linguistic consequence of the prefix notation, considered block prefixes as a natural extension. The reason for this was the fact that an ordinary in-line block could be viewed as the body of a very specialized anonymous class. We were quite pleased to discover that this very idea had been used before, through the ad hoc notation

SIMULA begin · · · **end**

of SIMULA I for making available the non-ALGOL simulation facilities.

Now everything fell into place. We only had to collect the various bits and pieces like prefix classes, procedures, and nonlocal variables (the SQS pointer) into one big class, appropriately named SIMULA and intended to be used for block prefixing. Its ''initial actions'' were used to initialize the SQS, containing a specialized process object impersonating the main program, whereupon control would proceed as it should: to execute the first active phase of the latter.

One ad hoc rule was needed to make the whole thing run: ''an instance of a prefixed block is a detached object by definition.'' Thereby the main program could function as a coroutine in quasi-parallel with its local process objects. (It was later discovered that this effect could have been achieved in a somewhat more natural way (Wang and Dahl, 1971).)

It goes without saying that the class/subclass constructs had not been fully explored as general purpose programming tools at the time of the first version of the Lysebu paper (March 1967). However, the possibility of using class declarations to define ''language dialects oriented towards special problem areas'' is pointed out, and so is the importance of ''making user defined classes generally available.''

The generalization to hierarchies of ''language dialects'' was fairly obvious. In June 1967 the ''SIMULA class'' had been renamed and reorganized as a two level hierarchy,

class SIMSET, and
SIMSET **class** SIMULATION,

reflecting the fact that circular list handling could be useful for other purposes than simulation modelling. (We never quite got rid of the term ''set'' from SIMULA I.) This kind of hierarchy points towards a technique of level-by-level bottom-up program design not unlike that of Dijkstra in constructing his THE operating system (Dijkstra, 1968).

No mention of the class concept as an abstraction mechanism is made in the Lysebu paper. It took several years of slowly growing understanding (see, e.g., Dahl, 1970; Dahl and Hoare, 1972), until the fundamental difference between the internal (''concrete'') view of an object and an external (''abstract'') one finally was made clear by Hoare (1972).

It is worth noticing that the Lysebu paper, as distributed prior to the conference, does not mention the concept of virtual attributes. In fact, a preliminary version of that mechanism was developed in April–May and is described in a paper dated May (Dahl and Nygaard, 1967b), also available at Lysebu. The concept was discussed at the conference, and a short section on virtuals has been added to the paper as it appears in the conference proceedings (Buxton, 1968). (This conference report is still today quite refreshing reading.)

We had seen that the class/subclass facility made it possible to define generalized object classes, which could be specialized by defining subclasses containing additional declared *properties*. However, the concatenation mechanism was not quite flexible enough for adding details to the *operation rules*. Something like procedure parameters still seemed to be needed for classes.

As discussed in Section 2.3.3 the ALGOL-like call-by-name parameters were out of the question for reasons of security and storage allocation strategy: the actual parameter could be lost during the lifetime of an object. The problem then was to find a name-parameter-like mechanism that would guarantee a safe place for the actual parameter. After much trial and error we hit on the virtual quantity concept where the actual would have to be declared *in the object itself,* but at a deeper subclass level than that of the virtual specification. Now generalized objects could be defined whose behavior pattern could be left partly unspecified in a prefix class body. Different subclasses could contain different actual procedure declarations.

The implementation efficiency of virtual quantities was good, since no change of environment was needed to access a virtual from within the object. Unfortunately we chose to model the virtual specifications after the parameter specifications of ALGOL, which meant that the parameters of a virtual procedure had to be run time checked.

It has later been shown that virtual quantities make it possible to directly use class hierarchies for top-down programming, but in a fairly clumsy way. Consequently this way of using classes has not received much attention.

3.3. The Common Base Conference

As reported in Section 3.5 a SIMULA 67 Common Base Conference (CBC) was arranged June 5–9, 1967, at the NCC, two weeks after the Lysebu conference.

The following papers were submitted to the CBC:

1. The Lysebu paper (Dahl and Nygaard, 1967a, Mar.),
2. "SIMULA 67 Common Base Proposal" (Dahl and Nygaard 1967b, May), and
3. "Proposals for Consideration by the SIMULA 67 Common Base Conference" (Dahl and Nygaard, 1967c, June).

The most controversial subject discussed at the CBC was what we called "in-line" declarations. We had realized that the indirect naming of objects through reference variables often represented wasteful overhead for the programmer as well as the program execution (not to speak of the garbage collecting effort). It would be useful to have a direct naming mechanism, in fact treating objects as ("in-line") compound *variables*. The Common Base Proposal, Section 8.3 reads:

8.3 In-line declarations

8.3.1 Syntax

⟨in-line object⟩ ::= ⟨identifier⟩ ⟨actual parameter part⟩

⟨in-line declaration⟩ ::= ⟨class id⟩ ⟨in-line object⟩|
⟨in-line declaration⟩, ⟨in-line object⟩

8.3.2 Semantics

A class identifier, underlined, may be used as a declarator. The identifier of a declared ⟨in-line object⟩ is a qualified ref variable, initialized to refer to a generated object of the stated class, and with the given actual parameters.

Assignment to the variable is not allowed. (Procedure declarations local to the class may be used to simulate en-bloc assignment of attributes.)

The implementor has the option to represent the declared variable by the associated object itself, rather than by the associated ref value.

(The important problem of reference variables elsewhere in the system pointing to an in-line object was not mentioned. To allow that would have grave consequences for the garbage collector.)

The proposals of (2) were, however, to be replaced by a set of much more ambitious proposals, (3), intended to unify the concepts of "class" and "type." These ideas were to some extent inspired by discussions at the Lysebu conference, and in particular the papers of Garwick (1968) and McNeley (1968). (Actually the paper presented to the CBC was an iteration of (3) produced during two hectic days and nights prior to the CBC. This paper has been lost.) Essentially the proposals were as follows:

1. Let C be a class identifier. Then

def(C) V ⟨actual parameter part⟩

is a declaration which defines a variable named V of "type" C.V is an (in-line) object of class C initialized according to its operation rule and the actual parameter part. Thus, the above declaration is comparable to

ref(C) X; followed by X := **new** C ⟨actual parameter part⟩

except that the latter generates an "off-line" object. The types C and **ref**(C) are distinct and an operation like X := V is therefore illegal. An ad hoc restriction on the class body (that it must contain no occurrence of the reference expression "**this** C") ensures that no reference values pointing to an in-line object can ever occur. Given a reference value X of type **ref** (C) to an object, then "X." is an expression of type C denoting that object. (Since it is mentally difficult to distinguish correctly between a reference to, or "name on," a thing and the thing itself, we felt there should be a notational difference forcing the programmer to precise thinking.)

2. Each basic type is a predefined class with local data and procedures defined outside the high level language. There is a one-to-one correspondence between operator symbols and a certain preselected set of identifiers (such as "becomes," "plus," greater," etc.). Thus a declaration line "**integer** a" means

def(integer)a;

where "integer" is a predefined class. And a statement like "a := b*c+d" means

a.becomes(b.times(c).plus (d))

where the closing parentheses are positioned according to operator priority, and the identifiers refer to procedures local to the class "integer." A consequence of this transformation rule was that operations such as :=, =, +, etc., would be available for a user defined type if and only if the class body contained procedures with the corresponding names.

3. Whenever a formal parameter is specified to be of type C, a corresponding actual parameter must belong to C or to a subtype (subclass) of C. This proposal opened up for a lot of interesting possibilities such as introducing side effects to assignment operations on variables belonging to subtypes of the basic ones (McNeley, 1968), without changing their meaning as operands in basic type expressions. It also made it possible to unify the concepts of function procedure and prefixed class; an instance of the procedure would be an object whose prefix part played the role of the function value.

No doubt the above proposals were prematurely presented. Although the final proposal document has been lost, we are sure that we had not managed to work through all the consequences for the language and its implementation during the two short weeks available between the Lysebu conference and the CBC.

Anyway, we made a valiant attempt to get our new proposals accepted by the CBC, but failed. The parties who had committed themselves to implementing the new language felt they were too complicated and cautiously rejected them, (NCC, 1967). We were in no position to overrule these parties. It was crucial for the continuation of the whole SIMULA 67 effort to get at least one major manufacturer (in this case Control Data, see Section 3.5) to commit itself to implementations. Unfortunately the simpler proposal of (2) was swept aside too.

The concept of virtual quantities was thoroughly discussed by the CBC. The discussion led to an interesting and nontrivial improvement of our own proposal. By a slight change of the binding rule for actuals it became possible to *redefine* previously bound virtuals. An actual declaration provided at a deep subclass level of an object would have precedence (throughout the whole object) over definitions at more shallow levels. This meant added flexibility in designing application languages. "Default" actuals for virtual procedures, provided at an application language level, now were replaceable in user defined subclasses. For instance, default error printouts might be replaced by corrective actions applicable in user defined situations.

One beauty of the principle of required qualification of references lies in the fact that it solves completely, at least for disjoint classes, one of the problems of language consistency discussed in Section 2.3.3. No object can lose its textual environment by active deletion of block/procedure instances, because any reference to an object must occur within the textual scope of the object class declaration and is therefore embedded in that environment.

However, in order to utilize the full capabilities of class concatenation it was deemed necessary to allow prefix sequences to cross block levels. In particular, predefined classes declared nonlocally to a user program, must be available for prefixing within the program. But then consistency is lost, since an object belonging to a local subclass may lose part of its textual environment if pointed to by a nonlocal reference. In order to prevent this phe-

nomenon we had to devise an ad hoc rule restricting reference assignments (and class prefixing), the so called "Rule *R*" (see, e.g., Dahl and Nygaard, 1967d), Section 4.1.2:

> A reference assignment is legal only if the left hand quantity is declared within the scope of the class qualifying the right hand side and all its subclasses, scopes defined after effecting all concatenations implied by class and block prefixes.

(It must be admitted that the wording of this rule is more compact than understandable). The rule would (as far as we know!) restore full security and consistency, even if unqualified references were allowed. However, it was very unnatural, too implicit, and therefore very difficult to grasp, to observe, and to enforce. A simpler rule of "same block level," called "Rule *S*," was apparently too restrictive:

> All classes of a prefix sequence must belong to the same block head (which may be that of a concatenated class or a prefixed block). Qualification is required for all references.

These matters were discussed by the CBC, but no recommendation was given, except to forbid unqualified references. Soon afterwards it occurred to us, however, that the essential capabilities of the prefixing mechanism could be salvaged and full security retained by applying the Rule *S* to class prefixes, but not to block prefixes. Sufficient flexibility could now be obtained by embedding predefined classes in a textually enclosing class *C*, say, (also predefined). Now these classes could be made available for class prefixing in any desired block head by prefixing that block by the class *C*. Since that technique would be the standard way of implementing application languages in any case, we were fairly happy. Still, there are situations in which Rule *S* is a nuisance.

Our various proposals to the CBC contained nothing about string handling and input–output. The CBC unanimously stressed the importance of having such facilities included as well defined parts of a "Common Base Language." Consequently a working group was established consisting of members of the implementation teams and persons from the NCC. One hoped that the class/subclass concept might lead to fruitful constructs. The proposals of the group should be submitted for approval to the "SIMULA 67 Standards Group" (see Section 3.5), which was also established by the CBC. Of the remaining recorded decisions of the CBC (NCC, 1967) one notes a recommendation on the syntax of "external class" declarations (for the inclusion of separately compiled classes). Everything considered, the CBC played a very constructive role during the birth of SIMULA 67. Among other things it helped the first implementation efforts to get off the ground by rejecting our premature proposals. However, if the CBC had been delayed a couple of months, SIMULA 67 might well have contained a general type concept. As an immediate follow-up of the CBC we produced a document called "The SIMULA 67 Common Base Definition," dated June 1967 (Dahl and Nygaard, 1967d), which, except for the missing string and I/O facilities, gives a surprisingly accurate description of the language as it exists today. In the preface are listed those whom we felt at that time had given us useful advice: C. A. R. Hoare, Jan V. Garwick, Per Martin Kjeldaas, Don Knuth, John Buxton, John Laski, Pat Blunden, and Christopher Strachey.

The Rule *S*, which later was made part of the Common Base, is mentioned as an "optional restriction." The introduction of the document refers to plans for a "full SIMULA 67 language," as opposed to the Common Base, which would contain a unification of the type and class concepts. But this dream was never to be realized.

3.4. The SIMULA 67 Common Base.

According to the CBC the SIMULA 67 Common Base (SCB) had been frozen in June, except for string handling and I/O facilities. The responsibility for the latter was assigned to a working group reporting to the SIMULA Standards Group (SSG). In actual fact a few adjustments were made to the SCB during the autumn of 1967, along with the development of the new facilities. But all decisions and proposals were made in contact with the implementation teams headed by P. M. Kjeldaas in Oslo and J. Newey in Paris (see Section 3.5).

At NCC Bjørn Myhrhaug was given responsibility for the development of the string and I/O proposals. Myhrhaug had for a long time been most useful as a partner in the implementation work and as a ''sounding board'' in our language discussions. Now he became a member of the design team and thus a co-author of SIMULA 67.

In October Myhrhaug had his first string handling proposals ready (Myhrhaug, 1967b). There were three alternatives, two of which introduced in-line strings of compile time defined lengths as a new primitive type. We rejected both of them for two reasons: insufficient flexibility, and the fact that they were based on concepts unrelated to those of the SCB.

The third proposal was based on classes and a new basic type *character*. There were two classes, ''string descriptor'' and ''string,'' the latter containing a character array, the former identifying a substring of a string object and a ''scan pointer'' for sequential character access, together with various operators declared as procedures. We felt that these constructs would provide good flexibility and could be implemented with reasonable execution efficiency, also on computers without direct character accessing capability. The price was a certain amount of overhead in run time data structure, and a syntactic overhead which was unacceptable. The latter could, however, be alleviated by defining ''string descriptor'' as an ''in-line object type'' using operator notation for operations like ''becomes'' and ''equal.'' True enough, this new construct could not be defined within the SCB, but it was consistent with our (still evolving) plans for a ''complete'' SIMULA 67 with classlike types. The new type was later called *text* in order to avoid confusion with ALGOL's string concept.

The input–output facilities were in some ways easier to design. The proposals converged on a hierarchy of classes corresponding to different kinds of files. (However, some details in the ''printfile'' class kept worrying the SSG for several years.)

The first meeting of the SIMULA 67 Standards Group was held in Oslo, February 10, 1968. The last of a series of proposals by Myhrhaug (1968) was presented at the meeting, together with recommendations by the Working Group. The proposals were approved (SIMULA Standards Group, 1968), and the NCC was instructed to ''provide a new SIMULA 67 Common Base Definition paper within 6 weeks.'' The NCC actually worked twice as long to produce it. The first complete and authoritative Common Base Definition is dated May 1968 (Dahl *et al.,* 1968a).

It so happened that the first batch of the new SCB documents came out of the printing mill the day before the ten-year anniversary conference for ALGOL in Zurich. Myhrhaug arrived late for the conference with 45 kg overweight baggage and reported difficulties in passing the customs control. The inspector had become very suspicious when, on opening one of the books, his eyes fell upon the following example (Dahl *et al.,* 1968a, pp. 22–27):

class hashing; **virtual: procedure** hash; . . .

One adjustment to the old SCB, approved by the February meeting of the SSG, deserves to be mentioned, since it is related to basic ways of thinking of objects, values and references. We had discussed during the summer of 1967 whether references were so different from other kinds of operands that they needed a special set of operator symbols. The answer had been a tentative ''yes, perhaps,'' but the possible gain in clarity had not seemed important enough to warrant the expense. Faced with our new text type the old question appeared in a new light. A text could either be thought of as a string descriptor (''text reference'') or as a character sequence (''text value''), and we needed some notation for distinguishing between them. The operators '':–,'' ''==,'' and ''=/='' were chosen for reference assignment and reference equality/inequality, respectively. It was then natural to apply these operator symbols to object references as well. A similar distinction was made between parameter transmission ''by reference'' and ''by value.'' This had only syntactic consequences for the old SCB, since an object transmitted by reference is the same as an object reference transmitted by value.

At the beginning of 1968 we regarded the Common Base as a statue with one leg missing. Our plan was to ''complete'' the language and provide an experimental in-house Univac implementation by extending the Common Base compiler now in progress on our UNIVAC 1107 (Nygaard, 1967). We were now thinking in terms of class declarations giving rise to in-line references to off-line objects, and analogous ''type'' declarations giving rise to in-line objects. (*text* could be seen as a type prefixed by a class, whose objects were references with additional descriptive information.) Some of our views at the time are expressed in a letter from OJD to Tony Hoare (Dahl, 1968a).

We had had some correspondence with Hoare during the preceding autumn, and through him we had seen some of his correspondence with Niklaus Wirth. This contact had been very stimulating and refreshing. We invited Hoare and Wirth to a three day meeting, February 4–6, to discuss types and classes (Nygaard, 1968a). We also wanted to consult them on our string handling and I/O proposals, about to be submitted for approval by the SSG. They kindly accepted our invitation, and we had a series of discussions which were very useful to us, since they clarified our own thoughts considerably. It turned out that their views and ours differed so much (Wirth, 1968) that we could not follow their advice in specific matters (which implies that neither of the two is responsible for any shortcomings of SIMULA 67's text and I/O facilities).

One concrete result of the meeting was that the **while** statement of PASCAL was included in the Oslo compilers and later proposed as a ''recommended extension'' of the SCB. It is now a part of the SCB.

The Common Base paper consumed most of our time during the spring. From the summer of 1968 on there were no resources for a continuation of work on the ''complete SIMULA 67.'' OJD moved to the University of Oslo, KN had to concentrate on ''SIMULA politics,'' and the implementors had enough problems to cope with within the Common Base.

3.5. The Fight for the SIMULA 67 Compilers.

Fairy tales end with the heroine getting her prince, and everybody (except the witch) living happily ever after. As language designers we often feel that people believe that one has ''succeeded'' and that a language ''exists'' as soon as it is defined, and that the subse-

quent implementation is a mere triviality, even if time consuming. The implementors very often are only briefly mentioned in a preface to some report.

In fact it is when the language is reasonably well described and thus the implementation task defined, that the main, tedious, and frustrating part of the struggle for a language's existence really starts. The actual writing of compilers is one aspect of this struggle. To obtain the resources necessary for this work is another aspect, to establish an organizational framework for distribution, maintenance, language standardization and development, is a third aspect. Finally, a fourth one is to make the language known and used.

We believe that these aspects of the histories of the various programming languages are usually quite dramatic and very often without a happy ending. We also suspect that this is the case even when large, powerful organizations are involved.

In the case of SIMULA 67 very scarce resources were at hand. The NCC is a small institute in a small country, without a worldwide organizational network to support its activities. We are not presenting the true history of SIMULA 67 unless we tell this second part of the story, at the same time mentioning a few of those who contributed to SIMULA 67's survival.

SIMULA I was originally only considered as a system description and simulation language, not as a general programming language. SIMULA I was implemented for UNIVAC 1100 series computers, later on also for Burroughs B5500 and URAL 16.

Our ambitions for SIMULA 67 were much higher: we wanted SIMULA 67 to become an ''existing'' general programming language, ''existing'' in the sense that it was available on most of the major computer systems, being used over a long period of time by a substantial number of people throughout the world and having a significant impact upon the development of future programming languages. We felt, of course, that SIMULA 67 was worth fighting for. What were our resources?

In 1968 NCC entered a difficult reorganization period. (The NCC is described in Section 1). From 1963 on NCC had been operating a UNIVAC 1107 and acted also as a computer service bureau. Now the 1107 was sold, and NCC was itself to become a user of other organizations' computers. A number of employees associated with our function as supplier of computing power left the institute. Even if NCC now was supposed to focus upon research and development work, large new long-range projects were not particularly popular in this turbulent period.

As mentioned earlier, the SIMULA I ideas were not received with enthusiasm in NCC's environment. The reception of SIMULA I in the professional world had made it difficult to maintain that the language was a poor one and its designers incompetent. This time it was stated (without any knowledge of the new language) that

1. Obviously SIMULA 67 was a wonderful language, but
2. Programming languages would have an average life time of about five years. Consequently, SIMULA 67 should not be implemented by NCC unless we could be certain of covering the total expenses by sales income over a five year period.

We were convinced that the language situation would be different in the coming decade from what it had been in the previous one. Neither users nor computer manufacturers would be willing to accept a stream of new languages. Also, in the research world it would be demanded that what had been achieved in the sixties should now be consolidated and evaluated through practical use of the most advanced high-level languages which had been developed.

We felt that in the next decade a large number of languages would cease to ''exist,'' and that only about a dozen would survive: COBOL and FORTRAN because of conservatism and the colossal investment in programs written in these languages, PL/I because of IBM support, and then, at a lower scale of usage, some ten other languages. We wanted SIMULA 67 to be in that group.

SIMULA I had been developed during NCC's expansion years in the early 1960s, within KN's research department, established in 1963 and named the ''Department for Special Projects.'' This department consisted in 1967 of six persons, the total staff of NCC being about 120 persons at that time.

Even if KN had a decisive influence on the work in the department, it was not possible to allocate more than at most four persons to SIMULA 67 implementations, and not possible to get additional resources from other departments unless such implementations were set up as projects intended to be profitable from a strictly economic point of view.

Another argument against SIMULA 67 implementation projects was a valid one, but greatly exaggerated: A modern, commercial compiler would require investment on a much greater scale than the SIMULA I compiler, and we were told that IBM was using about five hundred man-years on their PL/I effort. We realized that our objectives for SIMULA 67 would require much larger resources than SIMULA I, but could not accept that this should rule out SIMULA 67. (We often quoted the story about the two businessmen debating whether to locate an important software project in the US or in Europe, the question being settled by the remark: ''We have to locate the project in Europe, since in the US it is not possible to put together a sufficiently small team.'')

To sum up: our resources in manpower and money were very modest, and we had to provide economic arguments for the implementations of SIMULA 67. We had other resources, however: the reputation of SIMULA I, and the fact that SIMULA was linked to ALGOL 60. Simulation was becoming more and more important, and in Europe people started asking for SIMULA when computer purchase contracts were negotiated.

We had defined our objective as making SIMULA 67 a member of the small group of programming languages which in ten year's time would still ''exist'' in the sense defined earlier. Obviously, we had very great odds against us, and we had to plan very carefully. Some aspects of our strategy and tactics will be described below.

In practice, the language could be regarded as ''existing'' only if implementations of a high standard were available on the most important computers. In our environment that implied giving top priority to implementations on Control Data, IBM and Univac computers.

By ''high standard'' we understood:

1. Compilation and run time execution speeds comparable with the most efficient ALGOL 60 compilers.
2. Availability of comprehensive and well written documentation and educational material.
3. The existence and effective operation of distribution and maintenance organizations for the compilers.

We also felt, based upon the fate of ALGOL 60, that the implementations should be compatible as far as possible, and continue to be so. This implied:

1. String handling and input/output facilities should be defined as a part of the SIMULA 67 language.

2. The establishment of an organization which had the exclusive power to state what was "legal SIMULA 67" and adopt changes to that SIMULA 67. This organization had to be set up to provide conservatism, but also such that its members had genuine common interests in the spreading and development of the language.

Compatibility considerations also were important in other respects. Should SIMULA 67 contain ALGOL 60 as a subset? We disagreed with some basic design features of ALGOL 68, and compatibility with that language was ruled out. We also doubted that ALGOL 68 would be accepted by a large proportion of the ALGOL 60 user community and we felt that we could improve certain features of ALGOL 60 in SIMULA 67. On the other hand, ALGOL 60 is a beautiful language and the advantages of staying compatible were indeed very great. We decided that the possibility of running ALGOL 60 programs on SIMULA 67 compilers and of "growing" from ALGOL 60 to SIMULA 67 would attract many ALGOL 60 users. We needed their support, even with the limited size of the ALGOL community, and made only small modifications in the ALGOL 60 part of SIMULA 67.

A minor, but not unimportant point was the name of our new language. SIMULA is an abbreviation of "simulation language," and very obviously so. The new language was a general, high-level programming language and a system description language. In the short run the language would benefit from a name presenting it as an improved version of the simulation language SIMULA. In the long run the SIMULA name possibly might slow down the language's acceptance as a general purpose language. We decided that we were forced to give priority to the short term advantages and named the language SIMULA 67. Today the language suffers from the predicted long range effects.

In the spring of 1967, we did the basic groundwork in the design of the SIMULA 67 run-time system features. At the same time Control Data decided that they wanted SIMULA implemented both on their 3000 and 6000 series of computers because of customer demands. Among the customers were the Kjeller Computer Installation [KCIN, serving the Norwegian Defence Research Establishment (NDRE)] and the University of Oslo.

Negotiations between NCC and Control Data (acting through Control Data Norway) were started on March 1, 1967, and a contract was signed on May 23, 1967 (Agreement, 1967). According to this contract and discussions with Control Data:

1. Control Data intended to implement SIMULA 67. The 6000 series compiler would be implemented by Control Data, Europe. The 3000 series compiler would be implemented by KCIN.
2. At the insistence of Svein A. Øvergaard, director of KCIN, a firm "SIMULA 67 Common Base" language definition should specify what was to be implemented. A SIMULA Common Base Conference should meet "within the first ten days of June 1967."
3. A new organization named the "SIMULA 67 Standards Group" (SSG) was to be established. Eligible for membership of the SSG would be organizations which were responsible for development and/or maintenance of SIMULA 67 compilers. NCC would be an ex officio member and have the right to veto decisions in the SSG. Control Data should apply for membership. (The statutes of SSG are found in Statutes, 1967.)
4. NCC would provide SIMULA 67 implementation guidance to Control Data.

This was the initial platform for the SIMULA 67 implementation efforts. In June the University of Oslo, through its Computer Department headed by Per Ofstad, joined the

project. The 3000 series work was carried out in a cooperation between KCIN ("upper" 3000 series) and the University ("lower" 3000 series).

In order to make the initial and basic definition of SIMULA 67, named the "SIMULA 67 Common Base Language", and to set up the SSG, the "SIMULA 67 Common Base Conference" convened at the NCC in the period June 5–9. Present at the conference were representatives from Control Data, KCIN, people from the University of Oslo, and some individual computer scientists invited by the NCC. The conference succeeded in making the necessary basic decisions, deferred certain decisions for further study, and established the SSG, with Control Data and NCC as the initial members, KN being the first chairman. (Today the SSG has ten member organizations representing active implementations). Some of the decisions at the conference were rather important and have been discussed earlier in this paper.

The *real* freezing of SIMULA 67 happened during the autumn of 1967. *Formally* the freezing was achieved by decisions made at the SSG meeting in Oslo on February 10, 1968 (see Section 3.4).

We also planned from the outset that an "Association of SIMULA Users" (ASU) should be the framework for contact between the end users of SIMULA and for channelling their demands and complaints to the SSG and its members (having compiler maintenance responsibility). The ASU was established five years later, in September 1972, with Robin Hills being Chairman the first two years. ASU has since then had annual conferences and a series of workshops on a wide range of issues. (The first conference was in Oslo, the second in Monte Carlo). The present active membership is approximately 500.

The last element of the organizational strategy was the SIMULA Newsletter. After a few abortive attempts (NCC, 1968b) the Newsletter has been published regularly by the NCC since May 1972 and is now distributed to approximately 1000 subscribers. The reasons for the delays in setting up the ASU and the Newsletter were simple: we had to economize carefully with our scarce manpower resources and we did not get SIMULA users in any quantity until 1971.

In the beginning of 1968 the SIMULA 67 Common Base Language was quite well defined and the initial stage of the organizational plan in operation. Later on that year OJD became the first Professor of Informatics at the University of Oslo. He participated in the Control Data 3000 series implementations, but was mainly absorbed by the task of building up informatics at the University of Oslo. Also, in the beginning of 1968, the "battle for the compilers" started and lasted till the summer of 1969 when it was finally decided that NCC should implement and market SIMULA 67 on the IBM 360/370 computers and complete and market SIMULA 67 for the UNIVAC 1100 series computers.

The Control Data projects started in 1967 and were carried out in Paris under the direction of Jacques Newey (6000 series). In Norway the two 3000 series implementations were run as a joint KCIN/University project under the direction of Per Martin Kjeldaas of KCIN. There was some contact between the Oslo and Paris teams. In Oslo the work on the lower 3000 series compiler was pushed ahead, since the test facilities were better at the University. Both compilers were, however, ready in the spring of 1969 and turned out to satisfy the "high standard" criteria for efficiency stated earlier.

The Norwegian teams had some financial support from Control Data Europe. In return Control Data Europe obtained the right to use and distribute the 3000 series compilers.

The University and KCIN had maintenance contracts for their respective compilers. The 3000 series team directed by Kjeldaas consisted of Dag Belsnes, Ole Johnny Dahle, Øivind Hjartøy, Øystein Hope, Ole Jarl Kvammen, Hans Christian Lind, Åmund Lunde, Terje Noodt, Tore Pettersen, and Arne Wang.

At the NCC the situation was more complex. When the class/subclass concepts were invented, SIMULA 67 emerged and the "SIMULA II" ideas were dropped. Our work in the Department for Special Projects in 1967 and early 1968 was, in addition to the language definition, mainly directed towards the development of the basic design of SIMULA 67 compilers. We were always running implementation studies in parallel with the language design. A concept was not incorporated in the language until we had a sensible way of implementing it. One of the few exceptions was the "type proposal" (Section 3.3). A result of this work was the "SIMULA 67 Implementation Guide" (Dahl and Myhrhaug, 1969). This report contained the results of a quite substantial investment and was regarded as a commercial secret until it was released in 1971. The report was sold as a part of NCC's consultancy contracts with SIMULA 67 implementors. [See KN's letter to Hoare, then working at Elliott–Automation Computers Limited, dated November 3, 1967, for conditions offered (Nygaard, 1967).]

Bjørn Myhrhaug, Sigurd Kubosch, Dag Belsnes, and OJD were active in these design studies. Gradually Sigurd Kubosch (originally from Germany) became more and more involved with the UNIVAC 1100 series compiler, later on joined by Ron Kerr (from Scotland), and they did the main bulk of work on that implementation.

Kubosch and Kerr worked mostly alone and without the major support which the IBM compiler project later received. Their task was made even more difficult because NCC changed from using UNIVAC 1107 (with EXEC II) to UNIVAC 1108 (with EXEC 8) in the middle of the project. A first, rather restricted version was ready in the summer of 1969, being released to Univac, St. Paul, for evaluation purposes and to the Technical University in Trondheim. The compiler was gradually extended and improved. When it was clear that we had a marketable product, and that we had to market it ourselves, we were able to allocate Kubosch and Kerr to write comprehensive documentation. There had been no resources for that earlier. The first commercial delivery of the UNIVAC 1100 series SIMULA System took place in March 1971 to the University of Karlsruhe, West Germany.

In the spring of 1968 it was made clear to us that NCC could only support the production of Univac and IBM SIMULA 67 compilers if we could establish a financial arrangement which secured NCC the same payoff on these products as on our strictly commercial projects. Preferably, NCC should not run any economic risk.

The financial pressure was brought to bear upon SIMULA 67 at a time when NCC, as mentioned earlier, had large reorganizational problems. From a narrow SIMULA 67 point of view this was a lucky circumstance, since the insistent pressure was never followed up by administrative decisions. Time passed by and in the summer of 1969 the work on the Univac compiler and the support for the IBM compiler had developed to a stage beyond "the point of no return."

In the autumn of 1968 Harald Omdal was employed by NCC to assist KN in finding suitable financial arrangements for the two compiler projects. Omdal, former director of the Joint Computing Center of the four largest Norwegian commercial banks, was working as a private consultant. After initial work by Omdal on designing alternatives, he and KN

visited in the spring of 1969 a number of large Scandinavian companies to obtain their financial partnership in the production of an IBM 360/370 compiler. We got a pleasant reception, interest in later use of SIMULA 67, and some, but not many positive responses. We were not too surprised. Why should these companies pay in advance for something they could get without risk later? The whole idea of this type of project support had to be abandoned.

We also contacted IBM in Norway and Sweden. Both IBM and we were rather careful in these discussions. IBM wanted to support advanced national programming research in Scandinavia. To accept SIMULA 67 as an IBM-supported language would, however, be a major decision with far-reaching economic implications. Such a decision could only be made at IBM headquarters in the U.S. On our side, we were afraid of giving IBM control over the 360/370 compiler because of the risk of the language and its compiler being put on the shelf.

The results of these contacts were in the end positive and very important. IBM agreed to support the project by granting NCC a very substantial amount of computer time for developing and testing the compiler (40 hours on a 360/75 in Stockholm and 200 hours on a 360/40 in Oslo).

The event which finally triggered off the IBM compiler project occurred in the summer of 1969: the Swedish Agency for Administrative Development (Statskontoret) decided, with the support of the Swedish Defence Research Establishment, to participate in the project through two highly qualified programmers. Jacob Palme played an important role in this decision. The IBM 360/370 SIMULA 67 compiler project was headed by Bjørn Myhrhaug, and the team consisted of Lars Enderin and Stefan Arnborg (from the Swedish Defence Research Establishment), and the NCC employees Francis Stevenson, Paul Wynn, Graham Birtwistle (all from the United Kingdom) and Karel Babcicky (from Czechoslovakia). When Myhrhaug got a leave of absence, Babcicky was project leader for a period. Myhrhaug also was coordinator for both the Univac and IBM projects, being succeeded by Birtwistle. The first public release of the IBM compiler took place in May 1972 (to the governmental data center in Stockholm).

Univac had mixed reactions towards SIMULA 67. From a commercial point of view SIMULA I was a useful but not very important part of their software library. They felt no market demand for an improved language and, in particular, no reason to share SIMULA with other manufacturers From a professional point of view, however, many within Univac were actively interested in getting SIMULA 67. A long series of discussions and contract negotations with various Univac agencies followed, but a contract was never signed.

We think it can be safely said that the UNIVAC 1100 series and the IBM 360/370 series compilers both satisfy the criteria for "high standard" described earlier. It is interesting to observe that they were developed in two completely different ways. The two-man UNIVAC 1100 series compiler team worked their way with little support, using a long time, and were asked to provide comprehensive documentation at a late stage. The seven man IBM 360/370 series compiler team worked in a well supported and carefully planned project, documenting as they went along. The end result was that both compilers proved efficient and successful and both consumed approximately 15 man-years. Our initial estimate had been 8–10 man-years, assuming no suitable ALGOL 60 compiler available (Nygaard, 1967). We underestimated to some extent the design and programming work and to a great extent the documentation effort.

At the NCC a team of four persons, headed by Karel Babcicky, is now constantly employed in handling our SIMULA activities. For many years Eileen Schreiner has been our "SIMULA secretary" keeping all threads together and serving on the board of the ASU.

We have mentioned that the attitude toward SIMULA I and SIMULA 67 was rather negative in certain parts of NCC's environment. In other parts of that environment, among professional people, the attitude has been mainly positive. Within the NCC itself, SIMULA has had wholehearted and generous support. The period in which (in our opinion) too shortsighted economic considerations threatened the development was quite brief, atypical and had its reasons. Anyhow, an institute organized as the NCC is forced to take economics into account, and the compiler projects represented in the years 1968–1971 a significant economic burden for the institute.

Has SIMULA 67 then been an economic success or a failure for the NCC? That is a difficult question to answer, since it is not easy to measure the economic side effects of the SIMULA efforts. The experience gained has been the platform for later, straightforwardly profitable jobs, e.g., other language projects. SIMULA itself has made it possible to do jobs within operational research and data processing which otherwise would have been much more costly or beyond our capabilities. The international acceptance of SIMULA 67 has contributed to the Institute's reputation. In direct money terms, SIMULA 67 has not produced a profit. On the other hand, distributed over the eleven years since 1967, serious losses have not been incurred.

Today it is generally accepted that SIMULA has been a worthwhile effort, both for NCC and its environment. We have since then, in 1973–1975, developed a new type of language—a pure system description language—called DELTA (Holbæk-Hanssen *et al.*, 1975), starting from the SIMULA platform. From DELTA we are now deriving a new systems programming language, called BETA (Kristensen *et al.*, 1977) in cooperation with research workers from the Universities in Århus and Ålborg in Denmark. Whereas DELTA cannot be compiled, BETA of course can. Will the NCC embark upon the implementation of BETA, having had the SIMULA experience? This remains to be seen.

4. Concluding Remarks

The organizers of this conference have suggested that we should discuss our own languages' "implications for current and future languages." We find this difficult because of our personal involvement and think that other research workers are better judges on this subject. However, we are in the lucky situation that we may refer to Peter Wegner's recent article, "Programming Languages—The first 25 years" (Wegner, 1976), which contains many comments on SIMULA 67. Instead, we would like to conclude our paper with some reflections on our experiences from the process of developing a programming language.

In the spring of 1967 a new employee at the NCC in a very shocked voice told the switchboard operator: "Two men are fighting violently in front of the blackboard in the upstairs corridor. What shall we do?" The operator came out of her office, listened for a few seconds and then said: "Relax, it's only Dahl and Nygaard discussing SIMULA". The story is true. The SIMULA 67 language was the outcome of ten months of an almost continuous sequence of battles and cooperation in front of that blackboard—interrupted by intervals when we worked in our neighboring offices (communicating by shouting through the wall if necessary) or at home. (The people arranging this conference asked us

to provide material relating to the development of our respective languages. We felt that the best thing we could have done was to bring along that blackboard. But we did not know for certain whether we would be flying a wide-body aircraft.)

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. Assuming that the person who got the idea is willing to fight, this is a far better mode of work than the mode of mutual admiration. We think it was useful for us, and we succeeded in discarding a very large number of proposals. The class/subclass concept was perhaps the only one which was immediately accepted, whereas the virtual concept perhaps was the one which went through the longest sequence of initial rejections before it finally was given a definition acceptable to both of us.

When we started working together, we had quite different backgrounds. KN had quit programming in 1954, at a time when an essential aspect of the art was to program an algorithm with the minimum number of machine code instructions. His reasoning was always related to suitable language mechanisms for the description of systems in the real, physical world. OJD had been working on typical programming tasks and programming language design and implementation, with little experience from the problem area of operational research. In the initial stages of our cooperation we had communication problems. Gradually the area of common knowledge and understanding increased. We believe that our differences turned out to be a useful resource in our work, since each of us developed his own "role" in the team. In this way we were more likely to create ideas together which none of us would have created alone. We have later on both been working in close-knit teams with other people and we have found that we have had to develop other roles in these teams, the resource situation not being the same.

Sometimes we are asked questions like: "Who invented the virtual mechanism?" or "Who got the prefix idea?" Even if an answer could be given it would only tell who brought an idea forward the last of a long sequence of steps, and thus be of little interest. We tried once (when OJD was applying for his current position at the University of Oslo) to sort out at least certain rough areas of "ownership" in our relations to SIMULA I and SIMULA 67. When we found that each of us owned one half of the "reactivation point," we discontinued the effort.

We have been criticized for "dropping" SIMULA 67 after it had been developed. It is said that other people, e.g., Tony Hoare, Jean Ichbiah, Don Knuth, Jacob Palme, have done the real job of promoting the language. This is partially true, and we are grateful for their interest. One reason for the increased use of SIMULA 67 in recent years, especially within the United States, is undoubtedly the very successful DEC 10 implementation produced by a Swedish team in 1973–1974. Arnborg and Enderin, who also took part in the IBM implementation, were key members of that group. Ingrid Wennerström was another important member. Jacob Palme again played a decisive role in initiating the work. OJD's work on structured programming has been based on a SIMULA 67 platform, and has contributed to making SIMULA 67 known in the scientific community. Graham Birtwistle took the main burden in writing a comprehensive SIMULA 67 textbook (Birtwistle *et al.*, 1973). NCC and its staff has invested a substantial effort in promoting SIMULA 67 in many other ways: courses in many countries, publication of the SIMULA Newsletter, contacts with users in general. It was SIMULA 67's simulation capability which made it possible to get support for the implementation of the first set of compilers and to sell these

compilers to customers. If we had used our very scarce resources for writing papers and as traveling lecturers, SIMULA 67 might have been a paper language today, not a living one with an active user community.

REFERENCES

The letter P, D, or C following each reference should be understood as follows: P—publication, D—document, C—correspondence.

Agreement (1963) June 1. *Agreement made 1st day of June by and between Sperry Rand Corporation (Univac) and the Royal Norwegian Council for scientific and industrial research (by NCC).* (D).

Agreement (1967). *Agreement on implementation of the SIMULA 67 language between Control Data A/S Norway and the Norwegian Computing Center, Oslo May 23, 1967.* NCC Doc. (D).

Agreement (1969) September. *Draft outline for a SIMULA 67 agreement between UNIVAC and the Norwegian Computing Center.* (D).

Birtwistle, G. M., Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1973). *SIMULA BEGIN.* Studentlitteratur, Lund, Sweden and Auerbach Publ., Philadelphia, Pennsylvania. (P).

Blunden, G. P. (1968). Implicit Interaction in Process Models. In Buxton, J. N., ed, *Simulation Programming Languages,* pp. 283–287. Amsterdam: North-Holland Publ. (P).

Buxton, J. N., ed. (1968). *Simulation Programming Languages.* Proceedings of the IFIP Working Conference on Simulation Programming Languages, Oslo, 1967. Amsterdam: North-Holland Publ. (P).

Buxton, J. N., and Laski, J. G. (1962). Control and Simulation Language. *Computer Journal* **5**(5). (P).

Christopher, C. A. (1964) February 28. Letter to Leif K. Olaussen, Director NCC. (C).

Christopher, C. A. (1965) December 15. Letter to Leif Olaussen, Director, NCC. (C).

Clark, I. A. (1966) January 27. Letter to Leif Olaussen, Director, NCC. (C).

Clark, I. A. (1967) January 24. Letter to Kristen Nygaard, NCC. (C).

Dahl, O.-J. (1963) November. *The SIMULA Storage Allocation Scheme.* NCC Doc. 162. (D).

Dahl, O.-J. (1964) February/March. *The SIMULA Data Structures.* NCC Doc. (D).

Dahl, O.-J. (1968a) January 24. Letter to C. A. R. Hoare, Elliott-Automation Computers, Herts., England. (C).

Dahl, O.-J. (1968b). Discrete Event Simulation Languages. In Genuys, F., ed., *Programming Languages,* pp. 349–395. New York: Academic Press. (P).

Dahl, O.-J. (1970). Decomposition and Classification in Programming Languages. In *Linguaggi nella Società e nella Tecnica.* Milan: Edizioni di Comunita. (P).

Dahl, O.-J., and Hoare, C. A. R. (1972). Hierarchical Program Structures. In Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R., *Structured Programming,* pp. 175–220. New York: Academic Press. (P).

Dahl, O.-J., and Myhrhaug, B. (1969) June. *SIMULA 67 Implementation Guide.* NCC Publ. No. S-9. (D).

Dahl, O.-J., and Nygaard, K. (1963) May. *Preliminary presentation of the SIMULA Language (as of May 18th 1963) and some examples of network descriptions.* NCC Doc. (D).

Dahl, O.-J., and Nygaard, K. (1964a) March. *The SIMULA Language. Specifications 17 March 1964.* NCC Doc. (D).

Dahl, O.-J., and Nygaard, K. (1964b) July. *The SIMULA Project.* Tech. Prog. Rep. 1, July 1964. NCC Doc. (D).

Dahl, O.-J., and Nygaard, K. (1965) May. *SIMULA—A language for programming and description of discrete event systems. Introduction and user's manual.* NCC Publ. No. 11. (D).

Dahl, O.-J., and Nygaard, K. (1966) September. SIMULA—an ALGOL-based Simulation Language. *CACM* **9**(9): 671–678. (P).

Dahl, O.-J., and Nygaard, K. (1967a) March. *Class and subclass declarations.* NCC document. (As presented at the IFIP Working Conference on Simulation Programming Languages, Oslo May 1967.) (D).

Dahl, O.-J., and Nygaard, K. (1967b) May. *SIMULA 67 Common Base Proposal.* NCC Doc. (D).

Dahl, O.-J., and Nygaard, K. (1967c) June. *Proposals for consideration by the SIMULA 67 Common Base Conference, June 1967.* (D). [See author's comment on pp. 465–466. Ed.]

Dahl, O.-J., and Nygaard, K. (1967d) June. *SIMULA 67 Common Base Definition.* NCC Doc. (D).

Dahl, O.-J., and Nygaard, K. (1968). Class and subclass declarations. In Buxton, J. N., ed., *Simulation Programming Languages,* pp. 158–171. Amsterdam: North-Holland Publ. (P).

Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1966). *SIMULA. Simula Tracing System.* NCC Doc. (D).

Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1968a) May. *The SIMULA 67 Common Base Language.* NCC Publ. S-2. (D).

Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1968b) October. *Some uses of the External Class Concept in SIMULA 67.* (Presented by P. M. Kjeldaas at the NATO sponsored conference on Software Engineering, Garmisch, Germany, 7th–11th October 1968); referred in the conference report *Software Engineering, 1968,* Naur, P., and Randell, B., eds., p. 157. NCC Doc. (D).

Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1970) October. *Common Base Language.* NCC Publ. No. S-22. (Revised edition of S-2.) (P).

Dijkstra, E. W. (1968) May. The Structure of THE Multiprogramming System. *CACM* **11**(5). (P).

Garwick, J. V. (1968). Do we need all these languages? In Buxton, J. N., ed., *Simulation Programming Languages,* pp. 143–154. Amsterdam: North-Holland Publ. (P).

Gorchow, N. (1966) January 10. Telegram to Leif Olaussen, NCC. (C).

Gordon, G. (1962) September. A General Purpose Systems Simulator. *IBM Systems Journal* 1. (P).

Hegna, H., Lund, O. J., and Nygaard, K. (1968) June. *User's experience with the SIMULA language.* NCC Doc. (D).

Hoare, C. A. R. (1965) November. Record Handling. In *ALGOL Bulletin* No. 21. (P).

Hoare, C. A. R. (1966) May. Further Thoughts on Record Handling. In *ALGOL Bulletin* No. 23. (P).

Hoare, C. A. R. (1968). Record Handling. In Genuys, F., ed., *Programming Languages,* pp. 291–347. New York: Academic Press. (P).

Hoare, C. A. R. (1972). Proof of Correctness of Data Representations. *Acta Informatica* **1**:271–281. (P).

Hoare, C. A. R. (1974) October. Monitors: an operating system structuring concept. *CACM* **17**(10): 548–557. (P).

Holbæk-Hanssen, E., Håndlykken, P., and Nygaard, K. (1975) September. *System Description and the DELTA Language.* DELTA Project Rep. No. 4, NCC Publ. No. 523. Oslo: Norwegian Computing Center. (P).

Ichbiah, J. D., and Morse, S. P. (1969) December. *General Concepts of the SIMULA 67 Programming Language.* DR. SA. 69. 132 ND. Compagnie Internationale pour l'Informatique. (P).

Jonassen, A., and Dahl, O.-J. (1975). Analysis of an Algorithm for Priority Queue Administration. *BIT* **15**(4): 409–422. (P).

Knuth, D. E., and McNeley, J. L. (1964a) August. SOL—A Symbolic Language for General Purpose Simulation. *IEEE Transactions on Electronic Computers* ED-**13**(4): 401–408. (P).

Knuth, D. E., and McNeley, J. L. (1964b) August. A Formal Definition of SOL. *IEEE Transactions on Electronic Computers* ED-**13**(4): 409–414. (P)

Kristensen, B. B., Madsen, O. L., and Nygaard, K. (1977) September. *BETA Language Development.* Survey Rep. 1. November 1976. (Revised version, September 1977.) RECAU-76-77, DAIMI PB-65, NCC Publ. No. 559. (P).

Markowitz, H. M., Hausner, B., and Karr, H. W. (1963). *SIMSCRIPT, A Simulation Programming Language.* Englewood Cliffs, New Jersey: Prentice-Hall. (P).

McNeley, J. L. (1968). Compound Declarations. In Buxton, J. N., ed., *Simulation Program Languages,* pp. 292–303. Amsterdam: North-Holland Publ. (P).

Murphy, J. S. (1968) July 10. Letter to Kristen Nygaard, NCC. (C).

Myhrhaug, B. (1965). *Sequencing Set Efficiency.* Working Paper. (D).

Myhrhaug, B. (1967a) April 25. Letter to I. A. Clark, Systems Programming, Univac, Minnesota. (C).

Myhrhaug, B. (1967b) October. *A note on string handling facilities in SIMULA 67 Common Base.* NCC Publ. No. D24. (D).

Myhrhaug, B. (1968) January. *Proposal for string and input/output definition in SIMULA 67 Common Base. Preliminary presentation.* NCC Publ. No. 212. (D).

Naur, P., *et al.* (1960) May. ALGOL 60 Report. *CACM* **3**(5). (P).

Naur, P., *et al.* (1963) January. Revised ALGOL Report. *CACM* **6**(1): 1–17. (P).

NCC (1967) June. *Recommendations from the SIMULA 67 Common Base Conference, NCC, June 1967.* (D).

NCC (1968a). *1107/1108 SIMULA 67 Project.* Internal Information No. 1, July 22. (D).

NCC (1968b) July 26. *SIMULA Newsletter* No. 1. (D).

Nickitas, J. W. (1965) September 13. Letter to Kristen Nygaard, NCC. (C).

Nygaard, K. (1962a) January 5. Letter to Charles Salzmann, CFRO, Paris. (C).

Nygaard, K. (1962b) September 24. *Diskret-begivenhets nettverk* (Discrete event networks). Note, in Norwegian. (D).

Nygaard, K. (1963a). SIMULA, An Extension of ALGOL to the Description of Discrete Event Networks. In *Proceedings of the IFIP Congress, 62,* pp. 520–522. Amsterdam: North-Holland Publ. (P).

Nygaard, K. (1963b) April 17. Letter to C. A. Christopher, Director of Procurement, Univac (C).
Nygaard, K. (1963c) May 18. Letter to A. M. Paster, Manager Systems Research, Univac, New York. (C).
Nygaard, K. (1963d). A Status Report on SIMULA—A Language for the Description of Discrete-event Networks. In *Proceedings of the Third International Conference on Operational Research,* pp. 825–831. London: English Universities Press. (P).
Nygaard, K. (1963e) September 19. *Opparbeidelse av kompetanse innenfor Real-time Systemer* (Building up competence on real-time systems). Note, in Norwegian. (D).
Nygaard, K. (1965a) August 15. *The Software Contract between UNIVAC and the Norwegian Computing Center.* NCC Doc. (D).
Nygaard, K. (1965b). *Report on the use of SIMULA up to December 1965.* NCC Doc. (D).
Nygaard, K. (1966) February 17. Letter to S. M. Haffter, Univac Sperry Rand Corporation, Lausanne. (C).
Nygaard, K. (1967) November 3. Letter to C. A. R. Hoare, Elliott-Automation Computers, Herts., England. (C).
Nygaard, K. (1968a) January 29. Letter to Niklaus Wirth, Rechenzentrum der Universität, Zürich. (C).
Nygaard, K. (1968b) April 2. *En redegjørelse for samarbeidet mellom Det russiske vitenskapsakademi og Norsk Regnesentral om bruk av programmeringsspråket SIMULA i Sovjet* (An account of the cooperation between the Russian Academy of Science and the Norwegian Computing Center on the use in the Soviet Union of the programming language SIMULA). Note, in Norwegian. (D).
Nygaard, K. (1968c) September. *Oversikt over NR's SIMULA-engasjement* (Survey of NCC's commitment to SIMULA). Note, in Norwegian. (D).
Nygaard, K. (1968d) September. *Markedsføring av SIMULA 67* (Marketing of SIMULA 67). Note, in Norwegian. (D).
Nygaard, K. (1969) September 26. Letter to Peter Weil, Manager, Contracts and Pricing, Univac, England. (C).
Nygaard, K., and Dahl, O.-J. (1965). SIMULA—A Language for Describing Discrete Event Systems. In *Proceedings of the IFIP Congress, 65,* Vol. 2, pp. 554–555. Washington, D.C.: Spartan Books; New York: Macmillan. (P).
Palme, J. (1968). A comparison between SIMULA and FORTRAN. *BIT* **8**:203–209. (P).
Paster, A. M. (1962) October 12. Letter to Kristen Nygaard, NCC. (C).
Reitan, B. (1969) September 4. Letter to Kristen Nygaard, NCC. (C).
Roach, (1963) July 3. Telegram to Kristen Nygaard, NCC. (C).
Ross, D. T., and Rodriguez, J. E. (1963). Theoretical Foundations for the Computer-aided Design System. In *Proceedings of the SJCC,* p. 305. (P).
SIMULA Standards Group (1968).*Report from the meeting of the SIMULA Standards Group, held in Oslo, Norway, February 10, 1968.* (D).
Statutes (1967) May 23. *Statutes for the SIMULA Standards Group.* NCC Doc. (D).
Stevenson, F. (1967) November. *LOGIC, A computer program for simulation of digital logic networks.* NCC Doc. (D).
Tocher, K. D. (1963). *The Art of Simulation.* London: English Universities Press. (P).
Wang, A., and Dahl, O.-J. (1971). Coroutine Sequencing in a Block Structured Environment. *BIT* **11:** 425–449. (P).
Wegner, P. (1976) December. Programming Languages—The first 25 years. *IEEE Transactions on Computers* C-**25**(12): 1207–1225. (P).
Weizenbaum, J. (1962) March. Knotted List Structures. *CACM* **5**(3): 161–165. (P).
Wirth, N. (1968) February 14. Letter to Kristen Nygaard, NCC. (C).

TRANSCRIPT OF PRESENTATION

BARBARA LISKOV: Our speaker on SIMULA will be Kristen Nygaard. At the time that SIMULA was developed, Kristen Nygaard was the Director of Research at the Norwegian Computer Center, and apart from his work on SIMULA he was also responsible for building up the NCC, as it's called, as a research institute. He had been working in computing since 1948, and in Operations Research since 1952. Today, Kristen Nygaard, continues as Director of Research at NCC, but directs only his own projects. He also holds the rank of Professor at the University of Oslo, and his current research interests include programming languages and the social aspects of computing.

KRISTEN NYGAARD: During the writing of the paper for this conference, the authors had an abundance of letters from Jean Sammet and the language coordinators, but little communication between themselves. As it turns out, we have each chosen our own style. Those of you who have read our paper will know that it at least partially is in the form of an "action thriller." We are not certain whether it is fun to read or not, but it was on the point of becoming funnier. Two slides illustrate this.

Frame 1 shows a typical passage from the paper—from the "Greek nightclub episode." Observe the words "watching a beautiful belly dancer."

Quotation from the Greek
night club episode
(final version):

"While they were listening to
bouzouki music, watching a
beautiful belly dancer, Nickitas
presented the following informal
proposal,...."

Frame 1

Now observe the next to final version of the manuscript, (corrected at the last minute) in which we are not content with passive observation [Frame 2].

Also, our very competent typist felt that the sex angle was not sufficiently well handled. Her feeling materialized in terms of a very resolute insistence on writing SINSCRIPT instead of SIMSCRIPT (a prominent language which to our regret is not presented at this conference).

When one looks at one's own work at a distance, one is able to sort out the essentials and talk briefly. Having had the rather large job of writing our paper, we are in trouble because the SIMULA development once more has become very close. To sum up seven years of rather hard work in 25 minutes has become a difficult task; therefore, you must excuse me for using a manuscript. I will *close* my lecture with acknowledgments of some persons and one important activity outside of our own effort.

I will *start* by stating that SIMULA was, and is, a collective effort by many persons. SIMULA exists as a living language with an active user community, and the reason is a series of good implementations by some exceptionally competent teams. I admired the way in which John Backus brought his team members into the picture, and I wish that I

Quotation from the Greek
night club episode
(original version):

"While they were listening to
bouzouki music, washing a
beautiful belly dancer, Nikitas
presented the following informal
proposal,...."

Frame 2

had been able to do the same thing. I want to mention one name, however, that of Bjørn Myhrhaug, who designed SIMULA's string-handling and input–output; Ole-Johan Dahl and I very much would have liked to see him here.

SIMULA did not start as a programming language, and during all of its development stages reasoning outside traditional programming played an important part in its design. The first ideas came from operational research. My own work in that field told me that we needed a set of concepts and an associated language in terms of which one could understand and describe the complexity of the systems one had to deal with. It was also evident that simulation had to be the main tool for analysis. Consequently, from the very start in 1961, SIMULA was labeled both a system description language and a simulation programming language.

I was working within operations research, and had at the time lost contact with programming after leaving that field in 1954. I realized that I was not competent to design such a language alone, and since I knew Ole-Johan from my time at the Norwegian Defense Research Establishment, I tried to get him interested. I succeeded, and from then on, and through all the important development stages of SIMULA we worked together. It is impossible, even just between us, to find out who was responsible for which language concept. This may sound like a beautiful pastoral scene of peaceful cooperation. It was not. And the following story is true:

In the spring of 1967 a new employee at the Norwegian Computing Center came running into the telephone exchange and, very shocked, told the operator: ''Two men are fighting in front of the blackboard on the first floor corridor!'' The operator went out of her cubicle, listened for a few seconds, and said, ''Relax—it's only Ole-Johan and Kristen discussing SIMULA!''

When we started, Ole-Johan knew much about programming and next to nothing about systems thinking. I knew something about systems and next to nothing about programming. Today Ole-Johan knows much about systems thinking.

SIMULA started from a mathematically oriented concept of a network consisting of passive customers flowing through a network of active stations. However, we realized that the processing and decision rules to be described and simulated made it necessary that the language, quoting from a very early document, ''had to include a general algorithmic language such as ALGOL or FORTRAN.'' John Backus said in his speech yesterday that language design was a relatively easy part of the task. I understood him that way. That was not the case with the two SIMULA languages. It was a long and tedious process of working and reworking our concepts before we gradually arrived at the SIMULA of today.

At this point I want to mention another important aspect of our method of working. We always discussed how to implement as we went along, and never accepted anything unless we knew it could be implemented efficiently.

Our network concept dissolved for two reasons: we discovered that we could regard the networks as consisting of active customers and passive stations equally well as the opposite, what we labeled a ''dual view.'' We then realized that an in-between approach in many situations was very useful, and also discovered many situations which couldn't be described well by the network concept.

At this stage, the influence of ALGOL 60 became more and more prominent. It had at earlier stages been both an inspiration and an obstacle. We found the stack to be the obvious way of organizing a system component's action sequence. We believed, and it is specifically stated in our SIMULA contract with Univac, that we should implement SIMULA through a preprocessor to ALGOL 60.

In the spring of 1963 we were almost suffocated by the single-stack structure of ALGOL. Then Ole-Johan developed a new storage management scheme in the summer and autumn of 1963. The preprocessor idea was dropped, and we got a new freedom of choice. In February 1964 the process concept was created, which is SIMULA 67's class and object concept (but integrated at that time in simulation facilities, and without a subclass feature).

Frame 3 sums up some essential facts about SIMULA I. When we moved on to SIMULA 67 we had realized that SIMULA also was a powerful general programming language. Sometimes it is remarked that SIMULA's usefulness as a tool for implementing data types was a welcome lucky coincidence. I do not agree because many of the uses to which SIMULA has been put are the logical and necessary consequences of the system approach married to the ALGOL block concept.

It is true, however, that we did not grasp all of the implications of SIMULA at the time of creating its concepts. The transition from SIMULA I to SIMULA 67 is described in our paper, and I only want to summarize this by the next slide [Frame 4]. When we developed SIMULA we were not concerned with most of the questions which were essential to the ALGOL 60 fathers. We more or less took the algorithmic capabilities of ALGOL 60 for granted. Alan Perlis said to me yesterday that we should have developed SIMULA as the common superstructure for both ALGOL, FORTRAN, and possibly other languages. That was also our initial idea. But as our insight in what we regarded as a proper systems approach increased, this became more and more impossible. The ALGOL block concept which, with its integration of data, patterns for actions (procedure declarations) and actions became the cornerstone of our thinking.

At this stage, I want to illustrate what I mean by ''system thinking.'' Our main reference frame was a set of examples of systems in the world around us: job shops, airports, epidemics, harbors, etc. For this reason, the dynamic systems created by the program execution was first and foremost a model of the system described by the program. This reasoning was carried over to our understanding of more traditional parts of programming as well. Let us examine this approach in a little more detail.

In Frame 5 a sketch of an ALGOL program is showed on the left and a simplified model of the corresponding program execution on the right. What is physically generated is organized as a stack. Systems which may be usefully thought of as such stacks, are described conveniently by ALGOL. Frame 6 states this with other words.

Moving on to SIMULA, each component, now called an ''object'' in our model system (the program execution) is itself a stack. And therefore what is SIMULA? Here we have a

SIMULA I

developed:
June 1961–March 1964

design objectives:
system description
simulation programming

basic feature:
the process concept

Frame 3

SIMULA 67

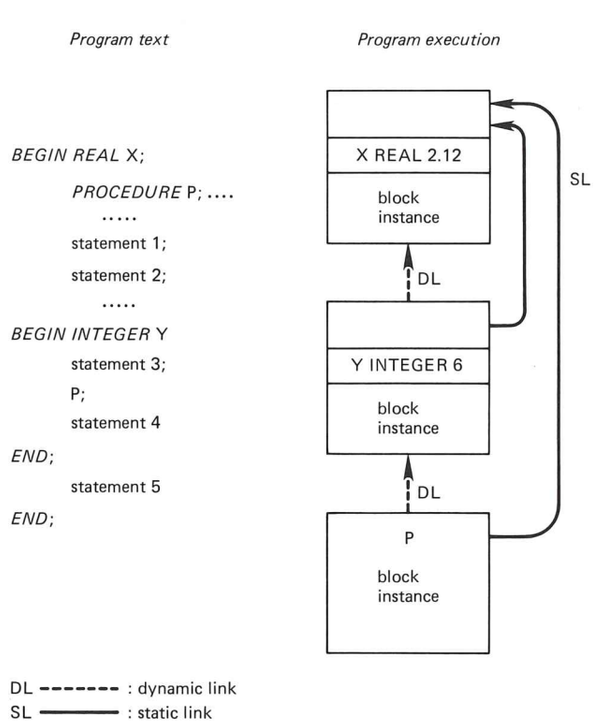
developed:
December 1966–January 1968

design objectives:
system description
high level programming
application languages
(e.g. simulation)

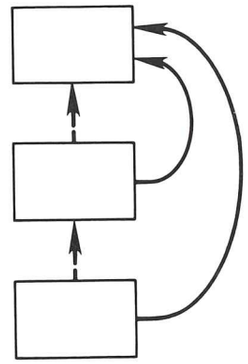
basic features:
the object/class concept
prefixing and subclasses
the virtual concept

Frame 4

number of stacks with the static enclosures, and SIMULA is the world regarded as a nested collection of interacting stacks [Frame 7]. When we examine what was new in SIMULA 67, two main and interrelated features were prefixing and virtual procedures. The prefixing with block concatenation made it possible for us to use Tony Hoare's ideas of reference qualification, and still keep the flexibility we wanted. It also provided the possibility of building up hierarchies of concepts [Frame 8]. As you know, using prefixing by Class *A*, an object of Class *B* contains an *A*, and its prefix and main parts are glued together as one integrated block instance.



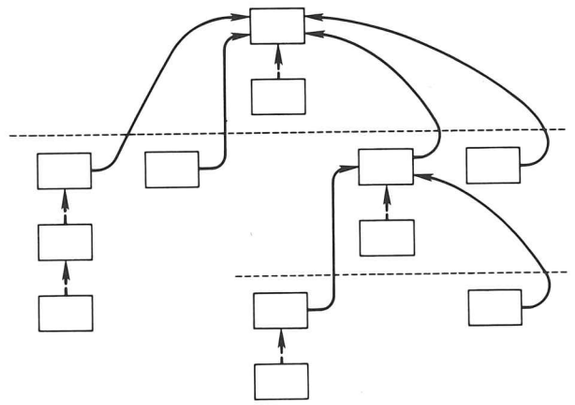
Frame 5



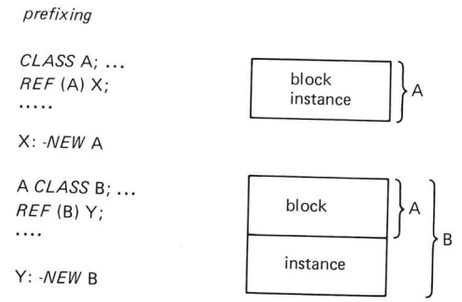
Frame 6. ALGOL 60—the world regarded as a stack of block instances.

One thing which should be mentioned in passing is that we tried by a last minute desperate effort to get a type concept into SIMULA 67 (in terms of in-line types). We did not succeed for reasons which are explained in our paper.

My last visit to the U.S. was in 1970. At that time the class concept only had a certain curiosity and entertainment value. I except people like Don Knuth, John McNeley, Bob Bemer, Al Paster, and some others. Today it's interesting and pleasant to observe that the situation is different. But—and there is a "but"—I still think that many people who refer to SIMULA only grasp parts of it. In fact, those who understand SIMULA best are not the people in programming research, but rather SIMULA's simulation users. The computer scientists like SIMULA as a vehicle for implementing data types. But many of them have never discovered the use and implication of the class/subclass feature. If they have, most have not exploited the virtual concept. And only very few, including Tony Hoare and Per Brinch Hansen, have realized what I feel is the most important property of SIMULA, that it is a multistack language. The computer-based systems we now have to



Frame 7. SIMULA—the world regarded as a nested collection of interacting stacks.



Frame 8

implement are networks of human beings, production equipment and computing equipment. For these systems I'm convinced that SIMULA's multistack framework is useful.

When I planned this presentation, I expected to use much of my time in discussing SIMULA politics. As I wrote along, my mind changed, as you have observed. Politics was, however, an essential part of the SIMULA venture, and it is described in detail in our paper. I will conclude my speech by pointing out what I feel were the most essential political elements of that venture.

When SIMULA 67 was developed, the Norwegian Computing Center employed approximately 120 persons. Of these 120, three persons could be assigned to language development. Backus told us that he got the resources he wanted. Grace Hopper's situation was more like ours. [Refer to the FORTRAN and Keynote presentations. Ed.] In 1962 we were told that (1) there would be no use for SIMULA; (2) there would be use, but it had been done before; (3) we would not succeed; (4) we should not make such efforts in a small and unimportant country like Norway.

In 1967 we were told that SIMULA was wonderful, but the lifetime of a programming language was only three to five years, and we have to earn back our expenses during that time period.

We had very small resources, and we had to fight for them. We wanted SIMULA to be an "existing" language, and our definition of that term is given on Frame 9.

These were our ambitions. To achieve these objectives, we needed the compilers to be of "high standard," a term which is defined on Frame 10.

The period from the spring of 1968 until the summer of 1970 was a crucial phase in SIMULA 67's life. The Control Data implementations were on the way, but SIMULA would not exist unless we got it onto IBM and Univac, and this was something which we

"Existing" language:

— available on most of the major computer systems

— being used over a long period of time by a substantial number of people throughout the world

— having a significant impact upon the development of future programming languages.

Frame 9

Condition for "existence"—
"high standard":

— Compilation and run time execution speeds comparable with best ALGOL 60 compilers.

— Availability of comprehensive and well written documentation and educational material.

— The existence and effective operation of distribution and maintenance organizations for the compilers.

Frame 10

had to do at the Computing Center. And we were in a difficult situation at that time, described in the paper.

In the summer of 1970 the NCC compilers had passed their "point of no return"; it was more costly to drop them than to complete them. Still, our resources were limited, as shown on Frame 11.

For this reason it was essential for SIMULA's success that we already in 1967 had designed an organizational strategy which was carried out during the subseqent five years [Frame 12].

The Norwegian Computing Center was at all stages, except a brief interval, very loyal to the SIMULA effort. And when you are shown loyalty, you also feel loyalty in return. Captain Grace Hopper concluded her opening address by expressing her gratitude and dedication to the organization she had served. I feel the same towards the Norwegian Computing Center and our comrades there. I could stop here, but I have to thank some of those outside the Norwegian Computing Center who were essential to the success of SIMULA. What I'll present to you is my own "short list" of names. Ole-Johan's may be slightly different, so I'll give it to you as my list.

First of all—and underlined—is Jan V. Garwick, the father of computer science in Norway. He was Ole-Johan's and my own first boss, and we are in great professional debt to him. Then follows a series of people at that time associated with Univac. Without Univac's immediate interest at an early stage, SIMULA would at least have been seriously delayed.

Stig Walstam brought us in contact with key Univac people in May 1962, as, e.g., Bob Bemer who listened for twenty minutes and then interrupted me by stating "Why don't you go to Munich [the IFIP World Conference in Munich] and discuss it?" And he was interested in discussing negotiations with Univac. But then Jim Nickitas, the man who got the whole thing started within Univac, the man who had faith in us at the Computing Cen-

Norwegian Computing Center:
120 persons

IBM SIMULA team:
7 persons (peak)

Univac SIMULA team:
2 persons

Frame 11. Resources.

Strategy:

— getting SIMULA 67 implemented on IBM, Univac and Control Data computers

— SIMULA Common Base Conference (June 1967) and SIMULA Common Base Language

— SIMULA Standards Group (SSG)

— Association of SIMULA Users (ASU)

— SIMULA Newsletter

Frame 12

ter, and was material in the Computing Center getting the 1107 computer. Al Paster, our main contact during our work with the concept. Bernie Hausner, staying with us for a while, and telling us about SIMSCRIPT. We learned from it; we copied it to a slight extent, and we learned what we wanted to be different. Joe Speroni behind our ALGOL compiler for our SIMULA I; Don Knuth, John McNeley—who very generously supported us. Eugen I. Yakovlev and Kirill S. Kusmin in the Soviet, at the Zentralniya Ekonomika Mathematicheskiya Institut in Moscow. Then Tony Hoare who is the single person whose ideas have meant most to us. Jean Ichbiah has done a big job in telling other people about SIMULA. Robin Hills, the first chairman of the Association of SIMULA Users. Jacob Palme . . . (I understand from the Johnny Carson show you have a big research project going on here in the states: trying to develop a T-shirt without anything printed on it! Henry Tropp has asked us to give references. I'm loyal to that. But it has not been developed in Sweden, and Jacob Palme is a person who has SIMULA 67 on his T-shirt.) But he has done very very much more. There are many others, and I'll use my leeway by ending up and saying—above all—well, it is at the bottom, but it is in fact, on top—ALGOL 60. Our gratitude first and foremost goes to ALGOL 60, and to all of the elegance and clearness which this language has, and which we have tried to carry over in SIMULA. Thank you.

TRANSCRIPT OF DISCUSSANT'S REMARKS

BARBARA LISKOV: As I'm sure you're all aware, SIMULA was in fact a two-man project. The other member of this team was Ole-Johan Dahl, and he's going to be a discussant for SIMULA today. At the time that the ideas for SIMULA came up, he was working for the Norwegian Defense Research Establishment for Jan Garwick. As a matter of fact, he was working on an implementation of an ALGOL-like high-level language that was going to be implemented—if I have this correctly—on a machine with 1000 words, and a drum of 16K words. Anyway, in 1962, Ole-Johan Dahl moved to the Norwegian Computing Center and started working full-time on SIMULA. Today he's Professor of Informatics at the University of Oslo and his primary research interest is in program specification and verification.

OLE-JOHAN DAHL: I would like to use another few minutes in talking about ALGOL and its surprisingly many kinds of applications, especially of course the block concept of

ALGOL, which turned out to be able to model all we felt that we needed for simulation models—that was the first SIMULA language. It also could be extended to the more general purpose class concept of SIMULA 67.

I have listed five different kinds of uses of blocklike constructs in SIMULA 67, all under the disguise of what we call a *class* or *class body*. I know that SIMULA has been criticized for perhaps having put too many things into that single basket of class. Maybe that is correct; I'm not sure myself. But it was certainly great fun during the development of the language to see how the block concept could be remodeled in all these ways.

So, the first and simplest instance of class is of course the pure data structures that you have in languages such as Pascal, ALGOL 68 and others; recordlike things, which you get out of an ALGOL block simply by deleting the block tail, keeping the declarations of variables and arrays.

The next example can be called "generalized data objects." You get them by including not only variable declarations in your class body block head, but also procedure declarations. The discovery was made in 1964–1965 that these so-called "procedure attributes" could be useful. And people like Tony Hoare have since shown us that what is really lying there is the concept of an abstract data object, with the procedures as abstract operators. The only thing we needed to do to ALGOL in order to make blocks look like data objects, was simply to devise a naming mechanism for block instances. And a mechanism of looking into blocks from the outside.

Next on my list is the process concept. It merely required us to include some very simple mechanisms for coroutinelike sequencing in addition to ALGOL's procedure activation mechanism. Having that, we had all the power of ALGOL programs going in quasi-parallel.

Then, the fourth on the list is the class prefixes. Now this was, as Kristen mentioned, a less trivial addition. The idea of prefixing one class by another one, and thereby composing a composite object consisting of two layers. But this enabled us to use the class mechanism for a kind of abstraction, collecting common properties of different kinds of objects into a single class, and making it available for use at later times as a kind of plug-in unit, where a given class could be extended into more concrete classes at later times by adding more properties.

And finally, the prefix mechanism could be used for ordinary in-line blocks too. This turned out to be a very interesting application, rather like collecting together sets of interrelated concepts into a larger class, and then using that larger class as a prefix to a block. The class would function as a kind of predefined context that would enable the programmer to program in a certain style directed toward a certain application area. Simulation, of course, was one of the ideas that we thought of, and we included a standard class in the language for that particular purpose.

There are two more points on my list of mechanisms that we proposed for inclusion in the language at the Common Base Conference which was held in June or late May of 1967. To use these class objects as generalized variables so that the concept of a class could be used rather like a generalized type concept, like integer and real. And finally we also saw the possibility of unifying the class and procedure concepts.

Now, as I said, it was great fun to see how easily the block concept could be remodeled and used for all these purposes. It is quite possible, however, that it would have been wiser to introduce a few more specialized concepts, for instance, a "context" concept for the contextlike classes. As a matter of fact, one of our disappointments with the usage of

SIMULA during the past years is the relatively little usage of the context-building capability of the language. Maybe this is also due to the fact that it is not at all an easy task to make good contexts. Thank you.

TRANSCRIPT OF QUESTION AND ANSWER SESSION

BARBARA LISKOV: There have been a number of questions, and I'll start with this one from Richard Miller: "Do you think the fact you were in a small group and country working on a project that many other groups were also working on helped push you on to the high quality work that you wanted?"

KRISTEN NYGAARD: Yes. I think that we benefitted from the fact that we had very complete control in a small group of what we were doing. I believe in team development of such a project. I'm skeptical of committee projects in designing languages. ALGOL 60 succeeded. They had a number of very brilliant people. They succeeded also because they had an apparently innocent but in fact very cunning secretary with devious manners, Peter Naur, to help the whole thing succeed. I address this comment also to the television cameras, which (we are told) records for history.

LISKOV: I have a question from Richard Nance at Virginia Tech: "Using the Kiviat–Lackner classification of simulation languages, both SIMULA and GPSS are described as process interaction languages. GPSS is narrowly focused on the interaction of temporary objects with permanent objects. Did SIMULA evolve from the same view, and how well can SIMULA treat interaction among permanent objects?"

OLE-JOHAN DAHL: I think it is fair to say, looking back now, that the conceptual origins are much more similar than we thought they were at the time. Our starting point was also the concept of a network through which things were flowing, and as I have understood, that is also the basic view of GPSS. About the distinction between temporary and permanent objects, I really can't see any great distinction between them. To me, a permanent object is an object that gets created at the start of an execution and lasts the whole time.

LISKOV: Here's another question from Richard Miller: "Could you comment further on the possible use of parallelism in SIMULA programs? When was this idea thought of?"

NYGAARD: I guess that you are talking about real physical parallelism as opposed to quasi-parallelism. Quasi-parallelism, to portray parallelism, was of course essential from the very outset. It occurred to us that this could be carried over to true parallelism, and in fact, there exists a note, in Norwegian, from 1963, about developing SIMULA into a real-time language. When we discovered it, among much dust, I reread it with quite some nervousness. But it turned out that the ideas there were not too bad, but of course we didn't at that time at all really understand all the problems related to physical parallelism. I think that we *could* incorporate such features in SIMULA, but I don't think it *will* be done. There are versions of SIMULA now with extended operating systems for running programs in parallel, but not in the nature of, say, concurrent Pascal.

LISKOV: Thank you very much.

FULL TEXT OF ALL QUESTIONS SUBMITTED

ULF BEYSCHLAG

SIMULA turned out to be a successful, nearly profitable programming language project. This without the strong backing of a manufacturer or a widespread academic community. What problems did you face and how do you see the chances for further projects with these characteristics?

PER BRINCH HANSEN

Many computer scientists now refer to SIMULA as the origin of the idea of abstract data types, that is, as a program module that makes a clear distinction between externally available abstract operations and internal implementation details of these abstract concepts. Was this viewpoint evident to you and Dahl from the beginning or was it a happy (unexpected) consequence of the SIMULA 67 class concept?

RICHARD MILLER

The class concept of SIMULA has become the prototype of the data abstraction techniques now coming into vogue in languages such as CLU and ALPHARD. Was this usage of the class apparent during the design? When did it become apparent?

(For Professor Dahl in particular, but. . . .) Could you describe your attempts (and eventual success) in developing the multiple stack concept for SIMULA. In particular, what options were available to you at the time and how were they used or eliminated?

Do you think the fact that you were in a small group and country working on projects that many groups were also working on helped push you on to the high quality work that you wanted?

RICHARD NANCE

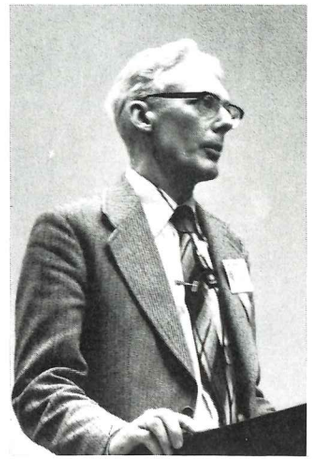
Using Kiviat–Lackner classification of simulation languages, both SIMULA and GPSS are described as process interaction languages. GPSS is narrowly focused on the interaction of temporary objects with permanent objects. Did SIMULA evolve from the same view? How well can SIMULA treat interaction among permanent objects?

BIOGRAPHY OF KRISTEN NYGAARD

Kristen Nygaard was born in 1926 in Oslo, Norway. He worked with the Norwegian Defense Research Establishment from 1948 to 1960 in computing and programming (1948–1954) and operational research (1952–1960). His master's thesis (1956) on abstract probability theory was entitled "Theoretical Aspects of Monte Carlo Methods." From 1957 to 1960 he was head of the operational research groups in the Norwegian defense. He was Cofounder and first chairman of the Norwegian Operational Research Society (1959–1964). Starting in 1960 he was employed by the Norwegian Computing Center, becoming its director of Research in 1962. Among his activities have been: building up the NCC as a research institute (in the 1960s), operational research, SIMULA I, and SIMULA 67.

After working on SIMULA 67 he did research for Norwegian trade unions on planning, control, and data processing, all evaluated in light of the objectives of organized labor (1971–1973). Other research and development work included: the social impact of computer technology, the general system description language DELTA (1973–1975); and the programming language BETA (1976–1980). He was Professor in Aarhus, Denmark (1975–1976) and in Oslo (from 1977 on). His work in Aarhus and Oslo included research and education on system development and the social impact of computer technology. He is a member of the Research Committee of the Norwegian Federation of Trade Unions. He has cooperated with unions in a number of countries.

BIOGRAPHY OF OLE-JOHAN DAHL



Ole-Johan Dahl was born in 1931 in Mandal, Norway. He worked with the Norwegian Defense Research Establishment from 1952 to 1963 with computing and programming under Jan V. Garwick. From 1956 onwards his main activity was software development. His master's thesis ("Numerical mathematics," 1957, University of Oslo) addressed the representation and manipulation of multidimensional arrays on a two-level store computer. His main contribution at the NDRE was a high level programming language, MAC, used locally during the early 1960s (first specification dated 1957, the implemented version modified as the result of ALGOL impact). In 1963 he joined the Norwegian Computing Center for full-time work on SIMULA, and in 1968 he became professor of computer science, then a new discipline at the University of Oslo. His main research during recent years has been in the areas of program architecture, specification tools, and verification techniques.

From 1964 to 1976 he was the Norwegian delegate to IFIP Technical Committee 2 (Programming Languages), and from 1970 to 1977 a member of IFIP Working Group 2.2 (Language Definition). He has been a member of IFIP Working Group. 2.3 (Programming Methodology) since its founding in Oslo in 1969.