Simulation of Large and Complex Systems: Some General Trends and an Example

Berth Eklundh

Lund Institute of Technology, Lund, Sweden

ABSTRACT

Some problems with design and simulation of complex systems are pointed out. The use of special languages for specification and description of systems are mentioned. The effects of a hierarchical system design approach are described and by an example the connection to system simulation is shown. The example clearifies the advantages with a system description that is based on the system being divided into modules by function. Communication between modules by means of signals are looked upon in an abstract manner and the apparition of program structure is reviewed in this case. The program structure hold some advantageous properties which are examined and put in relation to program evaluation.

INTRODUCTION

Due to advances in technology and science, man is capable of constructing larger and more complex systems than ever before. The development has up till now, however, been made with mainly the same technique that is used on smaller systems, but in a larger scale. In the middle we have one or a few chief constructors who, like a spider in his web, control the development of a new system. The subordinate deals with a small part of the system and the chief constructor is aware of most of the details in the subordinate's work. There are no distinct borders between different parts of the system and interaction between constructors is a necessity for a happy outcome. When the size and complexity of the system increases, this approach can be fatal, especially when the project contains software solutions, which is the case in an increasing number of modern systems. It is then no longer possible for one individual to have a clear view of all the details of the system, and this fact calls for new approaches to system construction and new aids for specifikation and description of systems. In recent years some effort has been made to invent special languages for this purpose and the progress on this is slow, but steady.

When it becomes possible to build more complex systems, there will, as a result, be a need for more powerful tools for analysis and prediction. One of the tools that are available for systems, is simulation. But simulation will, as the system that it wants to describe becomes more complex, also tend to a greater complexity. It is therefore desirable that construction of simulation programs keeps equal pace with advances in system construction and that it is possible to make use of concepts that are developed within the system construction area, as, e.g., special languages. By means of an example this paper intends to clearify some of these concepts and point out some trends in the field of realtime computer processing.

SOME QUERIES AND CONCEPTS

Before we describe an example, by which we intend to clearify some general ideas about simulation of large and complex systems, let us point out some problems that arise in this context. If we are faced with the task of simulating a complex system, it is most likely that we can rely on a description made by the system constructors. As mentioned in the introduction, there is no possibility that the total system behaviour could be kept within one man's knowledge and therefore there is a tendency to simplify the understanding of the system by partitioning it into modules, where each module could be seen as a black box, which gives a defined output on a certain input, but where the realization of the behaviour is unimportant. From an efficiency point of view the internal behaviour is, of course, important, but, however, not from a total system behaviour standpoint. A fairly common approach is to divide the system into modules by function. That is, each module contains a certain function, but this function could be achieved with a combination of hardware and software that spacially may be scattered around the system. In such an approach, modules usually communicate with signals and this "signal and module" division makes it possible to get a very pure description of the system. It does, however, bring some new problems. If one wishes to investigate a certain part of the system, like a computer, this part could functionally and descriptively be a part of many different modules and therefore hard to reach with, for example, measuring routines in a simulation program, unless the simulation describes the computer as well, and in such a case there will be a communication problem between the modules and, in this case, the computer. In the exampel that follows we will show a way to establish this communication. The description mentioned above can, however, provided that the communication problem is solved, have some great advantages. The description of the system will not alter if a function is altered from being implemented in software to be implemented in hardware. This means that the simulation program would need only a minor modification to map the new implementation, so this could be a method of testing different implementations without actually building them.

CH1437-3/79/0145-0151$00.75

1979 Winter Simulation Conference

AN EXAMPLE

The afore-mentioned new approach to system construction, with special languages for description and so forth, is especially well developed in the area of telecommunications. We have therefore chosen to illustrate our ideas with an example from this field. The Swedish Telecommunications Administration and Telefonaktiebolaget L M Ericsson have together, through their jointly owned company Ellemtel, developed the by now wellknown AXE telephone exchange system. This system provides, through its structure, a good example of the thoughts earlier described.

The AXE system

The AXE system is a telephone switching system, based on the SPC technique: (Stored Program Control). The development in hardware has made it possible to build more complex and sophisticated systems, and has also meant that the hardware costs have been reduced so that they today only take about 25 percent of the total cost. The rest is accounted for design, program evaluation, testing etc. The prices on hardware are already very pressed, and the main savings could therefore be made in software. It has been found that to accomplish any savings the system should be built with modules in such a way that it provides both functional and growth modularity. From a simulation point of view the functional modularity is the most important and we will concern ourselves with that part.

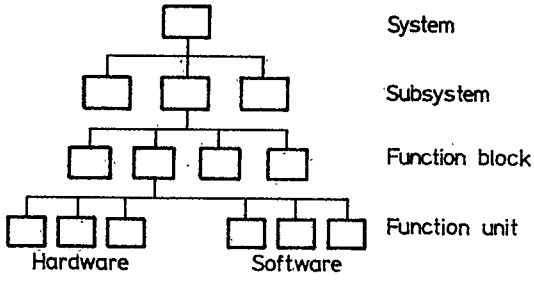

Fig. 1

In accordance with figure 1, the AXE system can be broken down into the following levels: system, subsystem, function block and function unit. The function blocks constitute the handling objects of the system and are implemented in software or a combination of software and hardware. The analysis of the system can be made unconditionally of how the functions are implemented.

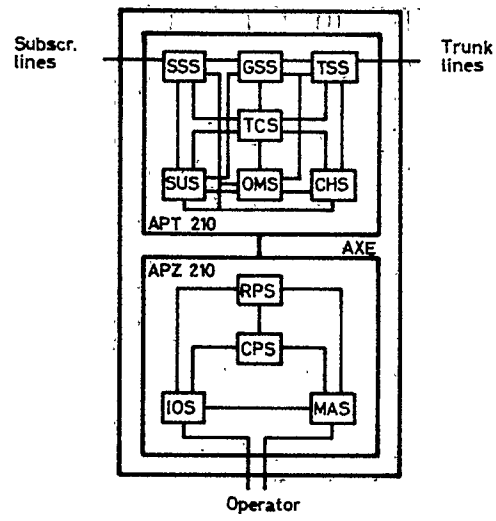

Fig. 2

With a top-down approach the system could be described as follows: At the highest level, AXE consists of the switching system APT 210 and the data processing system APZ 210, as in figure 2. In figures 2 through 4 some abbreviations for the subsystems appear. These have the following interpretation:

SSS	Subscriber subsystem
TSS	Trunk and Signalling subsystem
GSS	Group switching subsystem
TCS	Traffic routing and control subsystem
CHS	Charging subsystem
OMS	Operation and maintenance subsystem
SUS	Subscriber services subsystem
RPS	Regional Processor system
CPS	Central Processor system
IOS	Input/Output system
MAS	Maintenance system

A call to the station implies the execution of a number of tasks and usually the task division is made in such a way that simple timeconsuming parts are done by hardware while more complicated tasks are executed in software. There are for this purpose two software functions, which means that a task could be executed either in a central processor, which takes care of the most complicated software parts, or in one of the numerous (up to 512) small processors, called regional processors, which take care of more simple routines. The distribution between hardware and software of APT 210 is shown in figure 3, and in figure 4 the hierarchical structure of AXE is shown.

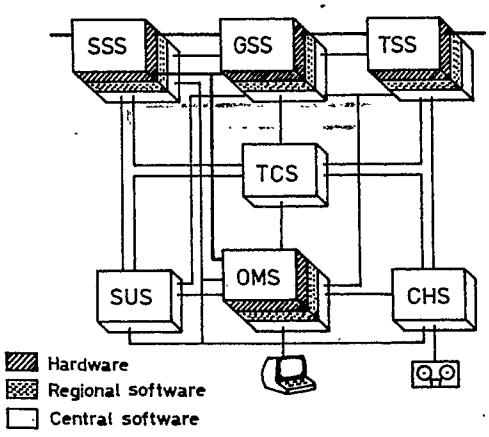

Fig. 3

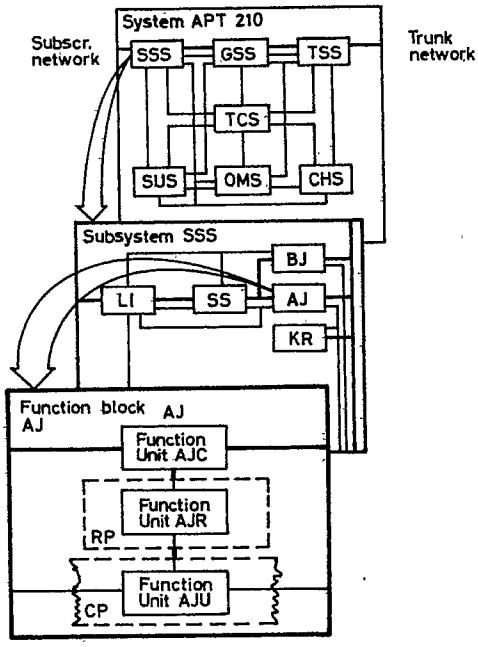

Fig. 4

From the figures it is seen that the system is cut into pieces, not by its physical parts but by function. This means that, for instance, the central processor is a part of many function blocks and this causes some problem in the simulation that will be dealt with later.

There is only one level at which communication can be established: between function blocks. There is no way by which function units could communicate and this restriction gives the system some handling advantages. The function blocks exchange information through signals and each signal has its unique name.

THE SIMULATION PROGRAM

It is seen from the description of the AXE system in the previous paragraph that the system is divided into function blocks and that communication between blocks is performed with signals. Each signal has a unique name and each block reacts on certain signals. An activity in the system is uniquely determined by a sequence of signals and it would therefore be valuable if such a sequence could be a guidance when making decisions in an event simulation. Below we show how this could be achieved but first a few words about the architecture of the simulation. We have chosen to write the simulation program in SIMULA. By using SIMULA we can accomplish some qualities that are desirable in a simulation program, and which make the program easy to read, correct, enlarge etc. We make use of the SIMULA Text concept and can therefore keep the original names of the signals which makes the program readable to those who are familiar with the AXE system but perhaps not with SIMULA. SIMULA has also enabled us to look at the flow in the simulation in terms that are conceptually advantageous, which is not always the case with other programming languages.

The AXE system could conceptually be divided in two parts: an abstract and a concrete. In the abstract part we have the flow of signals between function blocks and the function blocks can be assumed to receive signals, delay them some period of time and eventually send a new signal to some other function block. There are some exceptions to this behaviour, but as a base for the model this description is sufficient. The time that a signal is delayed in the function block depends on how the activities of the function block are implemented. If a signal is treated mostly in hardware it could be assumed to suffer some constant delay, but if it is to be treated by some processing unit, like the central processor or a regional processor, the delay will depend on the loading conditions in the system. The part that contains the buses, the central processor, the regional processors and so on, and which effects how much a signal is delayed, is called the concrete part. Since signals are exchanged only between function blocks and do not actually venture into the concrete part of the system we have chosen not ot let the signals be treated by the processors and buses but to invent new concepts into which a signal changes when it crosses the border between the abstract and concrete parts of the system. In the simulation these new concepts and the signals are all subclasses to a class in SIMULA which is defined through the following declaration:

```
Process Class Message;
```

To this class we have defined five subclasses by which all communication between different parts of the simulation is established:

```
Message Class Signal;

Message Class CPjob;

Message Class CPanswer;

Message Class RPjob;

Message Class RPanswer;
```

All these classes will, in reality, have some formal parameters, but it is convenient to omit them for the moment. The subclass signal is used for inter function block communication and the rest are used for communication between the abstract and concrete parts of the system.

The following figure will help to clearify the picture.

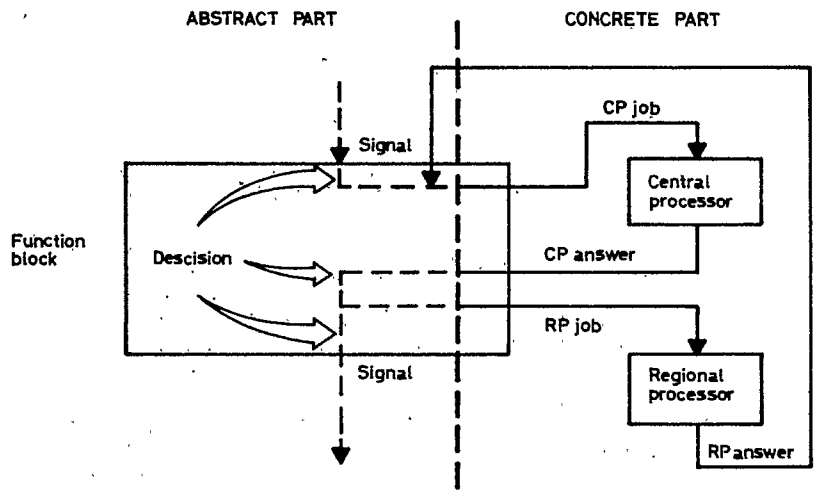

Fig. 5

As mentioned before, each activity in the system is defined through a sequence of signals, and the figure below shows the beginning of the sequence that defines the routines which have to be executed when a connection between two subscribers is to be established. The abbreviations LOAS, LI, SS, SC and AJ do all stand for function blocks, but their meanings are of no importance to us. The arrows that point upwards at the top of the picture indicate that an order is given to a regional processor to perform some function, e.g. test AJC (A-subscriber Junction Circuit). After a while, when the regional processor has fulfilled its duties, it sends a RPanswer back to the function block and the process continues. The messagenames are, of course, mnemonics, and are for readability reasons kept intact throughout the simulation.

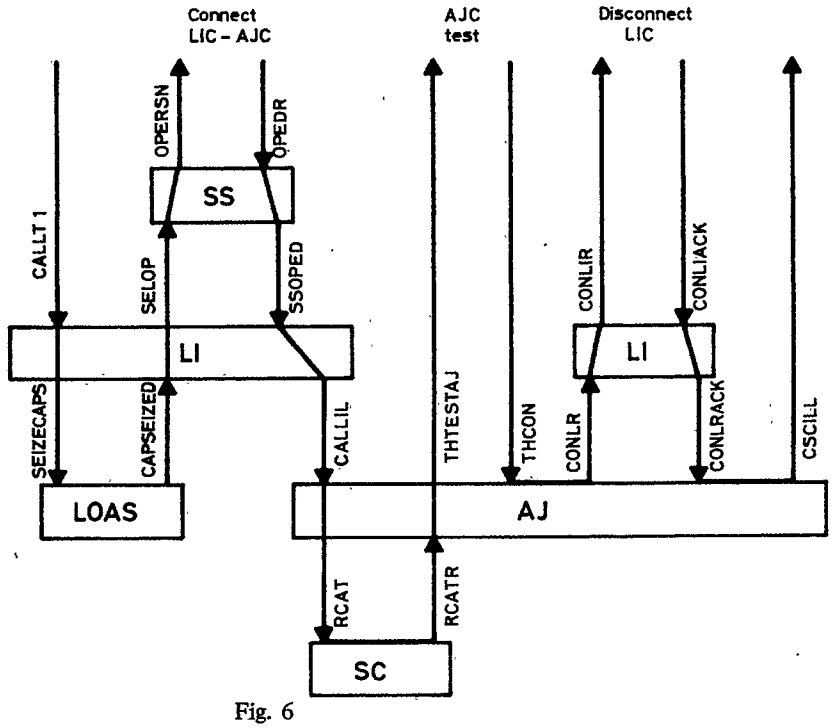

Fig. 6

The signal "CALLT1" is a RPANSWER from a regional processor that notities the function block LI that a call to the station has just been made, i.e. some subscriber has just lifted the receiver.

Now, since every function block has a behaviour like that showed in figure 5, it is possible, by use of the message concept earlier defined, to simulate the performance of the system in detail, and the exactness will depend only on how well the concrete part of the system is modelled. And since all decisions, what Messages are concerned, are taken in the functionblocks, the concrete part is relativily easy to model quite exactly.

Due to system characteristics it is reasonable to model the regional processors to give a constant delay while the central processor must be modelled in some greater detail. It is beyond the scope of this paper to describe the central processor model completely, and it is enough for our purposes to know that the central processor has an input queue JBB (Job Buffer B) and that jobs in the buffer are treated on a first come, first served basis. As can be seen from figure 1 a function block has to be able to recognize three different kinds of messages as input. We have found it convenient to give the formal parameter messagename in the subclasses CPjob, CPanswer, RPjob and RPanswer the same name as the signal it emanates from.

When a call enters the system it will be attached to a signal, and during its propagation through the system it will move between different subclasses to Message, but will always be attached to some, and this is, as will be explained below, a vital advantage with the approach. With a complete list of all the Messages that a function block can receive and some extra information, like the executiontimes for CP- and RPjobs, it is possible to build the simulation program. On the following pages we list the essential parts of a function block and the central processor. The example corresponds to the part of function block AJ that is shown in figure 2. We also give the final description of Message and its subclasses.

```
PROCESS CLASS FUNCTIONBLOCK;

BEGIN

   REF (HEAD) MESSAGEBUFFER;
   REF (MESSAGE) MESSAGEPOINTER;
   TEXT TYPEOFMESSAGE;

   MESSAGEBUFFER:-NEW HEAD;
   TYPEOFMESSAGE :- BLANKS (15);
END*** FUNCTIONBLOCK ***;

PROCESS CLASS MESSAGE
(MESSAGENAME, SUBSCRIBER);

TEXT MESSAGENAME;
VALUE MESSAGENAME;
REF (CALL) SUBSCRIBER;

BEGIN
END*** MESSAGE ***;
```

```
FUNCTIONBLOCK CLASS AJ;

BEGIN

  WHILE TRUE DO
  BEGIN
    PASSIVATE;
    MESSAGEPOINTEER :- MESSAGEBUFFER.FIRST;
    TYPEOFMESSAGE := MESSAGEPOINTER.MESSAGENAME;
    IF MESSAGEPOINTER IS SIGNAL
    THEN
    BEGIN
      IF TYPEOFMESSAGE = "CALLIL              "
      THEN
      NEW CPJOB ("CALLIL              ",MESSAGEPOINTER.SUBSCRIBER,17)
                                      .INTO (CENTRALPROCESSORPOINTER.JBB)
      ELSE
      IF TYPEOFMESSAGE = "RCATR               "
      THEN
      NEW CPJOB ("RCATR               ",MESSAGEPOINTER.SUBSCRIBER,2)
                                      .INTO (CENTRALPROCESSORPOINTER.JBB)
      ELSE
      IF TYPEOFMESSAGE = "THCON               "
      THEN
      NEW CPJOB ("THCON               ",MESSAGEPOINTER.SUBSCRIBER,28)
                                      .INTO (CENTRALPROCESSORPOINTER.JBB)
      ELSE
      IF TYPEOFMESSAGE = "CONLRACK            "
      THEN
      NEW CPJOB ("CONLRACK            ",MESSAGEPOINTER.SUBSCRIBER,17)
                                      .INTO (CENTRALPROCESSORPOINTER.JBB)
    END
    ELSE
    IF MESSAGEPOINTER IS CPANSWER
    THEN
    BEGIN
      IF TYPEOFMESSAGE = "CALLIL              "
      THEN
      NEW SIGNAL ("RCAT               ",MESSAGEPOINTER.SUBSCRIBER).
      INTO (SCPOINTER.MESSAGEBUFFER)
      ELSE
      IF TYPEOFMESSAGE = "RCATR               "
      THEN
      NEW RPJOB ("THTESTAJ            ",MESSAGEPOINTER.SUBSCRIBER,3
      ,"THCON             ").INTO (RP.BUFFER)

      ELSE
      IF TYPEOFMESSAGE = "THCON               "
      THEN
      NEW SIGNAL ("CONLR              ",MESSAGEPOINTER.SUBSCRIBER).
      INTO (LIPOINTER.MESSAGEBUFFER)
      ELSE
      IF TYPEOFMESSAGE = "CONLRACK                "
      THEN
      NEWRPJOB ("CSCILL               ",MESSAGEPOINTER.SUBSCRIBER,7
      ,"ASUBC             ").INTO (RP.BUFFER)
    END
    ELSE
    IF MESSAGEPOINTER IS RPANSWER
    THEN
    IF TYPEOFMESSAGE = "THCON               "
    THEN
    NEW CPJOB ("THCON               ",MESSAGEPOINTER.SUBSCRIBER,29)
                                    .INTO (CENTRALPROCESSORPOINTER.JBB);
  END;
END*** AJ ***;
```

```
PROCESS CLASS CENTRALPROCESSOR;

BEGIN
  REF (HEAD) JBB;
  REF (CPJOB) JOB;

  JBB:-NEW HEAD;

  WHILE NOT JBB.EMPTY DO
  BEGIN
    JOB:-JBB.FIRST;
    HOLD (JOB.NUMBEROFINSTRUCTIONS*INSTRUCTIONTIME);
    NEW CPANSWER (JOB.MESSAGENAME,JOB.SUBSCRIBER).
    INTO (JOB.SUBSCRIBER.FUNCTIONBLOCK.MESSAGEBUFFER);
    JOB.OUT;
  END;

END*** CENTRAL PROCESSOR ***;

MESSAGE CLASS CPJOB (NUMBEROFINSTRUCTIONS);

REAL NUMBEROFINSTRUCTIONS;

BEGIN
END*** CPJOB ***;

MESSAGE CLASS RPJOB (RPTYPE,ANSWERNAME);

INTEGER RPTYPE;
TEXT ANSWERNAME;
VALUE ANSWERNAME;

BEGIN
END*** RPJOB ***;

MESSAGE CLASS CPANSWER;

BEGIN
END*** CPANSWER ***;

MESSAGE CLASS RPANSWER;

BEGIN
END*** RPANSWER ***;

MESSAGE CLASS SIGNAL;

BEGIN
END*** SIGNAL ***;
```

In addition to the small sections of program described above there are in the program of course a large number of procedures, classes and son on, but we hope that this subset is enough for a basic understanding of the program structure and we describe below some advantages with the approach. The program contains at the present level approximately 10000 SIMULA statements and simulationprograms of this size inhabit some extra problems which are not present in smaller programs or which at least not cause any troubles. One of the great advantages with SIMULA is the possibility to make copies of classes, called objects. Each copy occupies a part of the main memory and it is necessary, therefore, to keep the number of copies within the limits of the main memory. An event simulation is based on an event list, into which events are scheduled, and when a new event has been created it has to be sorted to its proper place in the event list. In a large simulation program the executiontime is sometimes a limiting factor and it is consequently desirable to have as few items in the event list as possible, since searching and sorting are most time-consuming activities. Almost every simulation of queueing systems and other service facilities, like a telephone exchange, can be modelled as a flow of customers through a network of some kind. The number of customers tends to be great and they should therefore, by the arguments given above, be kept small and out of the event list. If a simulation is designed in a way similar to the one outlined in this paper these goals could be reached, and we will just briefly give some arguments for this. Primarily, since all decisions are carried out in the function blocks, the numerous part, which in this simulation is the calls, do not need any statements which control their activities. This means that they only have to contain some status variables which define their present state, and this simplifies their treatment. Secondly, since a call's odyssey through the system is divided into a series of events, where each event is delayed due to wait or processing in either a regional processor or the central processor, the call is always attached to some already scheduled part of the system, and it is therefore unnecessary to have the calls in the eventlist. The central processor and the regional processor will, naturally, because of the large number of events, be scheduled a substantial number of times, but since their number is relatively small, compared with the number of calls, the event list will hold much fewer items and there will thus be an important gain.

The program structure does also give a possibility to check the correctness of the program to a certain extent. The activities that a call has to undergo are described by a series of signal names. Therefore, if one lets a single call through the system and on each entry to a functionblock observes the signal name (it could, for example, be put out to a line printer) one should get a sequence that corresponds to the defined path. If a signalname is misspelled somewhere in the chain, the motion ceases and it is easy to tell where the error occurred.

The development of a simulation program usually contains a lot of work with modelling, program evaluation, testing and so forth. The approach described above can in many cases substantially reduce this load. Since the decision chains in the function blocks account for the major part (in the simulation mentioned above, approximately 90 % out of 10 000 statements), and since these chains only have to be copied from the system description, this work could be done by persons that neither have to know much programming, nor the system behaviour.

SOME TRENDS

The efforts to make a more stylistic approach to specification and description of systems have not yet reached very far, but there are some trends that can be mentioned and which probably will be of great importance in the future. Up till now almost all real time programming has been made in assembly languages, mainly because of the lack of proper high level languages for the purpose, or at least good compilers for the few languages that exist. Under supervision of the international teletraffic organization, CCITT, some interestgroups are working on the development of at least two different languages. One, named CHILL, is an ordinary high level language for real time programming, and is very strictly attached to the trends in system construction. The other, named SDL, Specification and Description Language, is somewhat more abstract, and deals with concepts very much like those which have been mentioned in the AXE-simulation context. It works with signals, states and similar concepts, and is very fit for describing systems like, for example, a telephone exchange, and not only that, but all kinds of systems that could be assumed to alter between different welldefined states. It is most likely that languages like these

will be very much used in the design of future systems of all kinds, and that they also will provide a powerful tool for the construction of simulation programs if some correspondence between the description in SDL and the simulation program structure could be established. We have in this paper tried to show a feasible way to do such a connection. As the development of description languages continues, the approach can be more rigorously described, but it is important that some attempts are made already at the present level so that the simulation technique does not end up in some backwater.

CONCLUDING REMARKS

In this paper we have pointed out some trends in system design, and made a stylistic approach to the simulation of systems described in analogy with these trends. The main point was the modularity of the systems and the communication with signals. We mentioned the appearance of special languages for system description and specification and saw that this could mean a simplification of the simulation problem. There was, however, a communication problem between the functional description and the hardware implementation. This problem called for some aids and we defined the message concept, which we used for communication between the functional description and the hardware.

With the simulation of complex systems some new problems came into view. The effectivity of the simulation program has to stand back for other demands, such as correctness, readability etc. We are convinced that to meet these requirements simulation programs have to built in a way similar to the systems they try to describe, and we hope that the experiences we have gained when simulating the AXE system will prove to be generally useful.

References:

Birtwistle, G. M., Dahl, O. J., Myhrhaug, B., Nygaard, K., **SIMULA BEGIN**, Studentlitteratur 1973

Fishman, G. S., **CONCEPTS AND METHODS IN DISCRETE EVENT DIGITAL SIMULATION**, John Wiley and Sons, 1973

Naylor, T. H., Balintfy, J. L., Burdich, D. S., Kong Chu, **COMPUTER SIMULATION TECHNIQUES**, John Wiley and Sons, 1966

References on the languages SDL and Chill can be found in:

CCITT Sixth plenary assembly; orange book vol VI.4, Series 2.100 -, -77

CCITT Study Group XI, HLL Implementary forum, Blue document, Feb. -79

Ericsson Review No. 2, 1976