# ITERATIVE MULTI-LEVEL MODELLING - A METHODOLOGY FOR COMPUTER SYSTEM DESIGN

F. W. Zurcher
B. Randell

Thomas J. Watson Research Center
Yorktown Heights, New York

Abstract: The paper presents a method of modelling a computer system design as it evolves, so that evaluation can be made an integral part of the design process. The paper introduces the concept of concurrent existence, within a single model, of several representations of the system being modelled, at differing levels of abstraction. Thus important design decisions are expressed directly in terms of appropriately abstract quantities, facilitating understanding, validation, and modification of the system design. The paper includes brief details of an experimental implementation of the modelling technique and of the use of the technique to model both hardware and software components of a multi-processing system.

## 1. INTRODUCTION

A major problem in the design of a computer system is that many aspects of system behavior and performance are not discovered until the system has been built and used in its operating environment. At that stage system modifications can be extremely difficult and costly.

Much use has been made of simulation for evaluating particular system components, and for modelling completed designs which require modification, either because of inadequate performance or to meet changing circumstances (see Nielsen (7)). Attempts to use simulation to model a complete complex system as it evolves have however been less successful.

Some of the reasons for this are fairly obvious. A detailed simulation model of a complex system will soon approach the complexity of the system being modelled. Thus modifications to the simulation model will be as difficult and costly as to the actual system, and the simulation will tend to slow down rather than assist the progress of the design. In addition, there is always the difficulty of interpreting the results of the simulator and of convincing the designers that the simulation is an accurate model of their design.

The present paper proposes and presents a modelling approach, which we call 'iterative multi-level modelling', that is intended to overcome these difficulties. Primary attention has been paid to the use of the technique for constructing simulation models, but many of its aspects are of relevance to the design of analytical models, and to the design of computing systems themselves. A designer using the modelling technique will find himself more cognizant of the relationships and interactions between the system components he creates, than of the detailed structure of individual components. In fact the system will be de signed "from the outside inwards", since at each stage the designer (or design team) must express what is to take place, before getting involved in the gory details of how it is to take place.

Thus the approach counteracts the tendency of designers to

i) incorporate design features from previous systems without careful assessment of their suitability to the new system.

ii) to assume that optimization of individual system components will automatically lead to optimization of the whole system.

This goal of providing a technique to aid design "from the outside inwards" is the same

as that expressed very well by Parnas and Darringer (8), in their descriptions of the SODAS system, which is primarily aimed at the design of hardware modules. The present technique on the other hand is designed for the more complex situation typical of an entire computer system consisting of software plus hardware. (The consequences of this are given in Section 2).

## 2. HIERARCHICAL MODELLING

As in several other schemes for using simulation languages to model computer systems(3, 8) an activity based language, reminiscent of SOL (5, 6) and SIMULA (1) is used, rather than an event-based language such as GPSS (4).

Use of an activity-based language facilitates modelling of system components in successively greater detail. Thus. the initial representation is in terms of an algorithms which produces the appropriate output for a given set of input, with the time lapse for this activity specified by an estimate of the time required by an actual component. The algorithm itself thus specifies what the component does and not how it does it. This permits preliminary evaluation of the contribution of the component to a larger activity.

The next step is to replace the algorithm with a sequenced set of calls on the next set of designed components. The new effective algorithm thus represents how the function of the original component is obtained in terms of these sub-components. The model can now be re-evaluated to check that its behavior is still acceptable, and in addition to obtain new information about the activities of sub-components. This evolutionary process is continued as desired, yielding a hierarchical model of the system.

Thus the modeller describes the behavior of the system, or a system component, using a program, constructed of a hierarchy of procedure definitions, incorporating statements to specify sequencing (serial or parallel) and to indicate the passage of time. This program models the behavior of the system by manipulating the values of a set of variables, these values at any moment representing the state of the system.

It is inherent in such a method of model evolution that one is limited to fully nested sets of components. This limitation is of no consequence in modelling a system component in which the problems of providing the resources needed by the sub-components are not considered. This is typically the case in design of hardware components, since the resources needed (circuits, transistors, etc.) are provided when the component is built, and exist for the duration of its lifetime. Thus this is essentially the method of modelling employed in SODAS (8).

Similarly it is possible to design at least simple software components on the assumption that some external agency is going to take care of the provision of resources (storage, CPU, etc.) that will be needed when they are to be used. However once one gets into the realm of the operating system, one has to design components that themselves use system resources, and whose function is to provide system resources for other components. Eventually one has to resolve such apparent paradoxes as "who allocates the allocator? " and "who schedules the scheduler? ".

It is implicit in the goal of designing an entire computer system from the outside inwards that decisions as to whether a system component should be hardware, permanently available software, or software which shares system resources will be made as late in the design process as possible. A model which can cope with this sort of situation during its evolution cannot be fully nested - it cannot be thought of as consisting simply of black boxes within black boxes, etc. It is to solve this problem that we introduce the idea of levels of abstraction.

## 3. LEVELS OF ABSTRACTION

The fundamental concept in the proposed modelling technique is that within a model several representations of the system being modelled can, and can usefully, coexist. These different representations will be at differing

'levels of abstraction'. For example the highest, most abstract level of a model of a computing system might just represent the various jobs currently being processed, indicating what major stage of their progress has been reached. At the level below, the existence of CPU's and a single resource corresponding to the entirety of a storage hierarchy and filing system could be introduced. This level could contain information indicating where the programs and data connected with each job were kept, whether the job was currently using a CPU, etc. Further levels would represent successively more detailed representations of the system, with the lowest level representing all the detailed information that had so far been generated during the design process.

Each level of abstraction consists of a simulation program, constructed of a hierarchy of procedures as described in Section 2. The program specifies the manipulation of the values of the set of variables which represent the system state at this level of abstraction. The program is essentially controlled by the program on the level above, which is making more global decisions based on its own variables. These next higher level variables are an abstraction of those on the current level, and hence are continually updated when the values of variables on the current level change. In turn the basic procedures of the current level, instead of containing algorithms for manipuating the variables of this level directly are in fact just providing requests for the level below to do the work.

Thus each design decision represented in a system model can be given at the appropriate level of abstraction, and expressed directly in terms of the quantities considered fundamental to that decision. This is in contrast to the conventional single-level model in which all decisions, whether high-level or almost trivial, are expressed only in terms of the most basic quantities represented in the model.

The resulting gain in ease of understanding (both of the model and its behavior), and in ease of modification to the model to incorporate design changes is considerable. This can be contrasted with the situation faced when trying to make fundamental although conceptually simple changes to, say, a conventional operating system or its detailed simulation. There the high-level abstractions remain solely in the minds of the original designer, and hopefully, in the documentation. For example, a 'job' may be represented by a linked structure of task control blocks, which are represented by blocks of words scattered about a storage hierarchy.

Furthermore the introduction of levels of abstraction allows one to cope with the problems associated with system components that share system resources, because the nested structure of procedures is not maintained in going from one level to the level below. For example, the highest level might consist of a set of parallel independent processes, one for each job currently in the system, and each executing a procedure which specifies the actions performed by a job. At this level there will be no system component which can be cognizant of the entire set of jobs. However in the level below one can have a finite set of processes, representing CPU's, each capable of executing the procedure which represents the scheduler, and having knowledge of all the jobs requiring scheduling. Hence at this level dependencies can be introduced between jobs which had previously been regarded as completely independent of each other - for example, that in competition for the limited number of CPU's short jobs have priority over long jobs.

## 4. RESOURCES

Also fundamental to iterative multi-level modelling is the fact that resources can be represented as "inactive" and "active" resources at different levels of the model.

The decision as to which of these two modelling techniques should be used depends primarily on what is being investigated at the current stage of evolution of the model. We model a resource in the active form when we wish to account for the fact that something is needed to transform a description of what is to be done into action. The inactive form is used

when we are solely concerned with the finiteness of the resource, and the dependencies this finiteness will introduce into otherwise independent processes. For example at some stage it might be sufficient to model CPU's as inactive resources, perhaps using for each CPU a pair of variables which indicate whether it is executing a program or not, and if so the name of the program. On the other hand, storage would probably be modelled as an active resource only when the design has reached the stage of being concerned with the mechanisms involved in accepting, retaining and disgorging bits of information.

An inactive resource is represented by the values of one or more of the variables which are manipulated by the simulation program (e. g. a counter might represent core storage by indicating how many words of core storage are currently unused). On the other hand, an active resource is modelled by one of the sequential processes which are defined by the simulation program at a given level. Such processes, and hence active resources, can be created and destroyed by executing special program statements.

In some cases a fixed set of active resources is created as part of the initialization of the simulation (e. g. a set of 16 CPU's). Alternatively, processes may be created and destroyed during the running of the simulation (e. g. the 'virtual computers' that users obtain by signing on at a terminal).

## 5. COMMUNICATION BETWEEN LEVELS

Two aspects of communication between levels, namely communication of control and communication of information, must be considered, although space limitations preclude detailed discussion.

The specification of communication of control involves replacing the statements which manipulate variables on a given level by statements which request action on the level below. These requests will be added to the list of requests which are being serviced by the processes which model active resources on this lower level.

The communication of information is needed to keep variables up-to-date with respect to those on lower levels of which they are an abstraction. A flow of information may be needed in the opposite, downwards, direction when only some of the system resources have been modelled in greater detail on the level below.

## 6. EVOLUTION OF A MODEL

At as early a stage as possible in the design of a computing system, a single-level model, consisting of a simulation program using a hierarchical procedure structure, is constructed and used to validate the initial design. The program is then expanded, as was described in section 2, as the design develops, until the designers wish to consider design features which are not amenable to introduction into the current model structure.

Assuming that the existing simulation program is still of value in the way in which it models the fundamental working of the system, it will be retained, and a second, lower, level of abstraction will be introduced. This will involve defining the set of variables which represent the system at this level of abstraction, and defining their relationship to the original set of variables. A program operating upon these variables, and controlled by the upper level program, is then constructed.

The second program will again be developed in a hierarchical fashion until it in turn becomes inadequate. In this way a set of levels will be constructed until the design has been expressed at the final level of detail required. This might be in terms of the instruction set of an already existing hardware system, or in terms of both software and hardware components.

In practice of course there may be many false starts, caused by decisions, both in the design of the system, and in the setting up of the model, which later prove unsatisfactory. A main function of the multi-level structuring of the model and the hierarchical structuring within each level, is to localize the effort involved in changing the model when this occurs. In many cases only one or more of the

lowest levels will have to be modified, and often a, quite significant change can be made at an upper level, without having to modify lower levels. The extent to which this happens depends on the care which has gone into the decisions as to what are, and will remain, valuable levels of abstraction worthy of retention in the model.

When the design and its representation (i. e. the model) have been completed we wish to use it to produce the actual system. One way of doing this, and conceptually the simplest and most elegant, is to replace the basic algorithms in the lowest level of abstraction, and the facilities which are provided by the simulation system for interpreting program text, storing program variables, etc., by the real-life mechanism from which the system is to be built.

This is best demonstrated by an example. A particular system might be designed and , modelled using two levels of abstraction, in the knowledge that it is to be made completely of software, though partly programmed and partly micro-programmed. The system is then placed on some already existing hardware which has facilities for storing and executing microprograms, thus producing an extended machine which can store and execute programs. Thus a two-level model is used to produce a two-level system.

In fact in these circumstances, there is no room for designers to say that they do not believe that the model is a true representation of the system they have designed - it is the system.

However it is not necessary to accept that just because it was found convenient to design a multi-level model, the actual system should have the same number of levels, or even have more than one level. Instead one uses the multi-level model until the design is complete, and then successively removes levels by replacing the sequencing which is defined by a simulation program on one level by sequencing in terms of the more detailed variables on the level below. On the other hand Dijkstra (2) has shown clearly that there are significant advantages to having an actual multi-level system.

## 7. PRESENT STATUS

Our first experimental implementation of the iterative multi-level modelling technique is based on FORTRAN but involves a distinctly ad hoc set of programming conventions. A set of utility routines (some written in assembly language) provide facilities for specifying parallelism and the passage of time, and for communication of control between levels. However the specification of information flow between levels has to be coded explicitly at points where requests are made in one level for action by the program at the level below.

By means of an extensive series of iterations a model has been constructed of a multiprocessing system incorporating a set of CPU's and I/O processors, a storage hierarchy and a set of I/O channels. The model currently consists of three levels. The first level represents the system as a set of independent jobs, without regard for limited system resources. The second level introduces the CPU's and I/O processors, and the storage hierarchy. On this level the operating system strategies which control contention for processors and storage are modelled.

However it is only on the third level that contention for channels, both for information transfer within the storage hierarchy, and to and from I/O devices, is introduced. The current implementation is intended to give us some experience in the use of multilevel modelling. It is hoped later to embark on a more ambitious implementation, in which much more care will have been paid to programmer convenience, to efficiency of model execution, and to provisions for building very large models. In particular we hope to develop a form of generalized dynamic equivalence mechanism for specifying information flow between levels.

## 8 CONCLUSIONS

The present paper has concentrated on describing the motive for, and the basic concepts of, iterative multi-level modelling. Due to limited space, it has been possible to discuss only briefly the various problems associated with translating these concepts into

a useful design tool, and the present use being made of our first experimental implementation. Nevertheless, it is hoped that readers will have obtained an understanding of the possible benefits to be gained from a modelling technique that can be used during the evolution of a system design, whose implementation will utilize both hardware and software.

## 9. ACKNOWLEDGMENTS

The authors have been greatly aided in their work on the iterative multi-level modelling technique by C. J. Kuehner, E Zweig and A. Boutross (the latter two being largely responsible for the detailed design of the experimental implementation). We are also pleased to acknowledge many helpful discussions with L. A. Belady, M. Lehman and H. P. Schlaeppi.

Thanks are also due to Mrs. J. Galto for her part in the preparation of the manuscript.

## REFERENCES

(1) O. J. Dahl, and K. Nygaard, SIMULA - an Algol-Based Simulation Language. Comm. ACM 9, 9 (1966) pp 670-678

(2) E. W. Dijkstra, The Structure of the T. H. E. Multi-programming System. ACM Symp. on Operating System Principles, 1967

(3) D. Fox, and J., Kessler, Experiments in Software Modelling. AFIPS Conference Proceedings Vol 31, 1967 FJCC. Thompson, Washington D. C. (1967) pp 429 -436

(4) H. Herscovitch and T. H. Schneider, GPSS III - An Expanded General Purpose Simulator. IBM Systems J. 4, 3(1965) pp l74-183

(5) D. E Knuth and J. L. McNely, SOLA Symbolic Language for General Purpose Systems Simulation IEEE Trans. EC-13, 4 (1964) pp 401-408

(6) D. E. Knuth and J. L. McNely, A Formal Definition of SOL. IEEE Trans. EC-13, 4 (1964) pp 409 -414

(7) N, R. Nielsen, Computer Simulation of Computer System Performance. Proc. 1967 ACM Nat. Conf. pp 581-590

(8) D. L. Parnas, J. A Darringer, SODAS and a Methodology for System Design. AFIPS Conference Proceedings Vol 31, 1967 FJCC. Thompson, Washington D. C. (1967) pp 449-474