# A Systemic Environment for the Formulation and the Solution of Hierarchical Models

Vaccari Erminia and Scommegna Sabina
Dipartimento di Informatica, Universita' di Bari
Via Orabona 4, 70126 Bari , Italy
Fax +39-080-5443262 Email:vaccari@di.uniba.it

**Abstract**

Assuming that a useful representation of anticipatory systems must be able to take into account a hierarchy of conceptual and physical processes, the paper discusses how the principle of hierarchy and modularity are incorporated in a specific simulation software system.

**Keywords**: anticipatory systems, recursive simulation, structured models, modularity, hierarchy

## 1.Introduction

Computing anticipatory systems whereby a change of a system's state in the present occurs as a function of some predicted future state ( Rosen ,1979 ) entails that the overall system is varying within the relevant support medium (space, time, etc.) The change from one system/model to another requires a replacement procedure that is invariant respect to the support employed, Such a procedure must act at an higher level than the possible system dynamics. Further computing a predicted future state to be used to compute a present system state necessitates the possibility to compute the different system dynamics at different time scales i.e. a faster dynamic for the predicted system/model.

In general, it is reasonable to assume that a useful representation of organised complexity systems must be able to take into account a hierarchy of conceptual and physical processes.

A model can be hierarchical in different senses, e.g.:

a) connections between certain sub-models may represent authority relations in the real system, whereby subordinate sub-models are controlled by higher level sub-models;

b) a sub-system, S, may be represented twice in a model, once by a sub-model representing S as a whole and once by a set of sub-models representing the constituent sub-systems of S.

Typically sub-models at different hierarchical levels will represent phenomena occurring on different time scales.

Hierarchical models are usually formulated and solved by using computational heuristics since the classical mathematical formalisms, on which dynamic modelling is based ,do not foresee hierarchical representations.

In the last two decades, several authors ( Delaney and Vaccari, 1974,1979,1984,1989; Oren and Zeigler,1979 ; Zeigler, 1976a,1976b,1984 ) have stressed the importance of a system theoretic basis for simulation algorithms.

In the following we will describe how the principles of modularity and hierarchy can be incorporated into the system theory based design of simulation software systems .We will use a specific such system, BDSIM.CPP, as an example.

Specifically, discrete event (DEVS) simulation is considered. In such simulations a system is modelled in terms of events occurring at irregularly spaced time instants. First we discuss the formalisation of such simulations in the system theory paradigm. Then the modularity and hierarchy criteria are discussed, and possible incompatibilities between the criteria are pointed out. Finally a way of resolving these incompatibilities is suggested and illustrated with reference to the BDSIM.CPP system.

## 2.DEVS Simulation in the System Theory Paradigm

DEVS simulation involves the simultaneous solution of DEVS models. A DEVS model is defined (Zeigler , 1976b) as a structure:

$$< U , X, Y , f , g , h >$$

where

U= a set of input values

X= a set of model states

Y= a set of output values

f= a function for calculating a model's state in terms of its inputs and preceding state (at the occurrence of an event)

g= a function for calculating a model's output in terms of its state (at the occurrence of an event)

h= a function for calculating the time which must transpire until the next event (at the occurrence of an event)

A set of interconnected DEVS models constitutes a Structured DEVS Model, definable as the structure:

$$<[M(i)],[I(i)],[Z(i)]>$$

where:

-[M(i)]= a set of DEVS submodels

-[I(i)]= the set of submodels which furnish input to submodel M(i)

-[Z(i)]= a set of functions which define the correspondences between the outputs of the submodels in I[i] and the inputs to submodels M[i].

Thus, a *Structured DEVS Model is essentially* a set of generative models and a coupling scheme (defined by functions I and Z)
Simulation of the operation of a system represented by a Structured DEVS Model requires a program which is able to step through simulated time, from event to successive event, and, at each event time, to:

- use the individual DEVS models (functions f, g ) to calculate the models' states and outputs;
- use the individual DEVS models (function h) to determine their "next event times";
- *choose the next event time for the whole structured model as the minimum of those* associated with the individual models.

The present discussion in large part regards how the above steps are realised in a specific system simulator (BDSIM.CPP). Very synthetically, it performs them on the basis of :

1. a declarative *definition of a Structured DEVS Model obtained from an input file.* The functional subsystems constituting the global system of interest and its environment are referred to as *blocks* since they may be represented as blocks in a functional block diagram of the system; the environment block will have no inputs and its outputs represent the total system inputs. A block is labelled as being a physical block (PB) or an abstract block ( MB) in the program input. The distinction is important since the order in which MB's are processed can be important. A PB represents a physical activity, involving time consuming state changes; its output relative to a certain time cannot depend on its input relative to that time. An MB does not consume time and thus its outputs relative to a certain time can depend on its input at that time.
2. user supplied procedures defining the individual DEVS models. The laws of behaviour (of the functional subsystems constituting the structured system) are incorporated in C++ functions called Block Processing Sections (BPS's ) and block interactions are specified by inputting the following information : a list of mnemonic identifiers of block outputs, each such output corresponds either to an input or to a state variable of the total system; a list of mnemonic identifiers of blocks and, for each block, a list of identifiers for its parameters and lists of identifiers of output and input variables associated with the block. Essentially BPS's represent form of relations, thus a single BPS can be used to simulate various blocks performing the same type of activity. A definition of a block includes the specification of its associated BPS which computes new state and output values at the event occurrence. For physical blocks (PB's) also the time to the next event is computed .
3. declarative definition of the simulation experiment to be performed Experimental frame definition in its various aspects is achieved as follows: *Initial state specification, in the program input data,* initial values for state variables can be assigned either as fixed numbers or as possible values with associated probabilities.

The user also sets a flag determining whether each replication initiates from the same state or if it uses the final state of the previous replication as its initial state. *Input specification*, exogenous inputs are generated by the BPS which simulates the environment. *Output specification and termination conditions*, total system outputs may be the outputs of blocks representing subsystems or they may be the outputs of special Observation Blocks (OB's) whose corresponding BPS's have the specific function of calculating such outputs. An OB may be an MB (for sampling outputs at events determined by other blocks) or it may be a PB (for sampling at times determined by the OB itself).

In the following we use the term Total DEVS Simulation Model (TDSM) to designate the simulator (BDSIM.CPP) together with the entities described in points 1) and 2) above (i.e. it is everything except the experiment definition information).

## 3.Modularity and Hierarchy

Modularity is an obvious characteristic of Structured DEVS Models, in that they are formed by combining together individual DEVS models according to well defined rules, whereby outputs from certain DEVS models become the inputs to other DEVS models.
It is important to realise that this output/input relation is the only relation between the DEVS models. Internally a single DEVS model has no knowledge of the existence of the other DEVS models; it only "knows" its own inputs and outputs.
Modularity also extends to Structured DEVS Models as wholes; i.e., two Structured DEVS Models can be coupled together by connecting outputs from one of them to inputs of the other one.
Modularity is very desirable in that it is useful for managing complexity. A very significant example regards the possibility of modularly combining already validated Structured DEVS Models, known to provide valid representations of the systems they represent. The overall structure will still need some testing, but much less than if its parts were not already tested.
Hierarchy is another principle of well known utility in complexity management.
The combination (as discussed above) of two Structured DEVS Models (A and B) into a more complex model (C) involves considerations of a hierarchical nature in that the reason for the combination will often be a perceived hierarchical relationship between real world systems. However it should be noted that the model C is a Structured DEVS Model consisting of the DEVS models which were contained in A and B. The modeller knows that C was constructed from A and B but these latter are not evident in the resulting model. So called "authority hierarchies" can be conveniently represented in this way; instead of representing the management - operations hierarchy present in a real system as a hierarchy in the model, one simply foresees BOSS models at the same level as "WORKER" models knowing that the outputs from the former will, in fact, constitute "commands" in input to the latter, but this is not evident in the model syntactical (I/O) form.

There are however situations in which it is convenient for hierarchy to be (syntactically) explicit in the model. That this is the case is not at all surprising. Even if the real world were not hierarchically organised in itself, it would be very convenient for us to organise our knowledge of the world in hierarchical structures as a complexity reduction technique. In general terms, we might find it desirable to explicate, in an overall model (A), both a submodel (B) and its constituent sub-submodels (C(i)) (we call such a hierarchy a "constitutive hierarchy"; note that the syntactical form of the model would evidence such hierarchical structure). Three possible situations where such formulations could be desirable are:

- studies involving "feedforward" phenomena where it would be useful to represent a subsystem twice, once as an "actor" in the functioning of the system and once as a "thinker" who imagines (mentally simulates) future reality as a means for choosing actions to perform

- studies aimed at discovering examples of "emergent" phenomena; here again it will be useful to represent a subsystem at two hierarchical levels to facilitate testing whether a higher level (valid) model really contains something more than lower level models can explain

- the hierarchical representation yields a significant amount of complexity reduction.

A problem with the introduction of hierarchical structures like those discussed in the preceding is that (depending on the specific modelling context) they may not respect the modularity principle; for example if it is necessary for the subsystem B in the previous paragraph to "know about" its submodels (C(i)), e.g. so as to be able to interrogate them about their "state variables". The best resolution of this problem we have so far been able to identify is to implement an interface utility of BSDIM CPP. A C++ function manages the I/O couplings between levels following the strategy that the higher level becomes the environment of its lower level.

## 4.A Simulation Example

A simulation environment (BDSIM.CPP) incorporating the above ideas has been realised in the C++ language. The aspect of the environment that we want to evidence here is how it incorporates modularity and hierarchy. Very synthetically, it allows Structured DEVS Models to be defined ( in a declarative form ) in correspondence with different model hierarchical levels where the processing of a hierarchical level is accomplished through a recursive invocation of the simulator.

To illustrate how BDSIM.CPP can simplify the simulation of systems exhibiting various forms of hierarchical structure it is useful to discuss the simulation of an hypothetical hospital division (HOSP), which is a good example of a hierarchical system. For our purposes, the essential HOSP subsystems are a laboratory (LAB), a radiological unit (RAD) and a WARD where patients stay, after arriving from the hospital's environment (ENV). Progressively more complex models of the hospital will be discussed.

Two types of model will be considered, descriptive models (with names beginning with "D_") expressed in natural language and formal models (with names beginning with "F_"). Model names also have a suffix (".1", ".2", etc.) to indicate alternative models of the same physical system. In general formal models will be algorithmic formalisations of corresponding descriptive models (e.g. F_WORLD.1 is an algorithmic formalisation of D_WORLD.1). Mention will be made of special kinds of formal models, having particular structural forms and/or special functionality's (DEVS and TDSM's ). The hierarchical aspect of the above total reality can be evidenced by the system decompositions:

WORLD = (HOSP, ENV)

where

HOSP = (WARD, LAB, RAD).

A simple TDSM (F_WORLD.1) of WORLD would incorporate formalisations of the following simple descriptive models, D_WARD.1 of WARD, D_LAB.1 of LAB, D-RAD 1 of RAD and D_ENV.1 of ENV:

D_LAB.1 = laboratory for analysis of blood, urine, etc. The time to complete an analysis is a random variable following a Gaussian probability distribution with known parameters (of course only positive values are considered).

D-RAD 1 = radiological unit performing different types of services like X-rays, TAC, etc. The time to complete a service is given by known probability distributions associated to the specific services.

D_WARD.1 = set of patients. The patients pass through a sequence of phases from entry into the ward to exit from the hospital. Duration of a phase is a random variable with a known probability distribution generally conditioned both by a patient's "disease" and by "disease phase". Waiting for LAB results is considered a phase. Patients do not interact with each other.

D-VISIT 1= the activity of a physician which (for each patient ) determines the type of disease, the disease phase , the eventual phase change and if laboratory analyses and/or radiological services are required.

D-CHAMP1 = the activity of a physician which decides the specific types of laboratory analyses required.

D_ENV.1 = the hospital's environment, which supplies patients at time intervals distributed according to an exponential probability distribution.

In the following we will not consider the radiological unit D-RAD 1 since for our purpose it is analogous to D-LAB 1..

A descriptive model D_WORLD.1 of WORLD will incorporate D_LAB.1, D_WARD.1, D_ENV.1 and descriptions of the interactions between LAB, WARD, and ENV such as:

- the output of ENV is input to WARD
- outputs from WARD (samples to be analysed) are input to LAB
- outputs from LAB (analyses results) are input to WARD

The hierarchical aspect of D_WORLD.1 can be evidenced by the model decompositions analogous to the above system ones:

D_WORLD.1 = (D_HOSP.1, D_ENV.1)

where

D_HOSP.1 = (D_WARD.1, D_LAB.1).

These decompositions are "constitutive" hierarchies.

The TDSM F_WORLD.1 formalising D_WORLD.1 will contain the formal models ( F-WARD.1, F-LAB.1, F-ENV.1 ) respectively based on *(D-WARD.1, D-LAB.1, D-ENV.1)*; i.e.

F_WORLD.1 = (F_HOSP.1, F_ENV.1).

F_HOSP.1 = (F_WARD.1, F_LAB.1).

The descriptive models evidence the characteristic aspect of discrete event (DEVS) models, i.e., events separated by time intervals of random duration. This implies that the simulation model F_WORLD.1 will be a structured DEVS model (plus time management).
In the context of F_WORLD.1, it is easy to envision simple DEVS models (F_LAB.1, F_ENV.1) as adequate formalisations of the descriptive models (D_LAB.1, D_ENV.1), but the formalisation of D_WARD.1 requires special consideration. The basic question is whether F_WARD.1 can also be a DEVS model, which necessitates deciding on the formalisation of a patient (PAT). The simplicity of

D_WARD.1 suggests that an adequate formalisation of a PAT in the context of F_WARD.1 might be

F_PAT.1=(DISEASE,PHASE,TIME_LEFT)

which is a set of (random) variables where TIME_LEFT is the time a patient has to remain in phase PHASE of disease DISEASE. The adequacy of such a formalisation depends on the fact that the PAT's do not interact and their descriptions only evidence the above variables as being significant, and as having known associated probability distributions.
There is nothing atypical of mathematical entities such as F_PAT.1 existing in the context of DEVS models, so one can still entertain the hypothesis that F_WARD.1 could be such a model.

However, to further clarify the nature of F_WARD.1 it is necessary to also evidence the nature of its associated significant events. These events are patient arrivals and phase changes. The time to the "next" such event is generally the shorter of: the time to the arrival of a new patient or the shortest time to a patient phase change. This implies that F_WARD.1 must manage and monitor the TIME_LEFT variables associated with the patients just as is done in a TDSM, which leads to the conclusion that F_WARD.1 is fundamentally such a model.

The nature of F_WARD.1 raises important issues regarding its implementation. Indeed it practically constitutes a simulation model inside of the F_WORLD.1 TDSM, which suggests that it could be implemented in the F_WORLD.1 context by means of a recursive call to the same simulator which manages F_WORLD.1, which, in turn, implies the advisability of using an object oriented simulator (such as BDSIM.CPP), where such operations can be easily accomplished
This approach necessitates formalising the model F-WARD.1 as a structured DEVS model formed by DEVS submodels F-VISIT 1 and F-CHAMP 1 respectively based on D-VISIT 1, D-CHAMP 1 i.e.

F-WARD 1 = ( F-VISIT 1, F-CHAMP 1)
Let us note that the above formalises D-HOSP as a second order structured DEVS model, and global modularity is maintained.

To illustrate another type of hierarchy easily manageable by BDSIM.CPP we consider the following, more complex, descriptive model of LAB.

- D_LAB.2 = a set of analysis stations and a BOSS. Each station (STATION) performs a specific type of analysis (e.g., blood analysis). Some stations require the presence of a operator. The BOSS assigns operators and analysis samples to machines. In general, the stations are not independent of each other; there is an indirect interaction among them (through the BOSS) when the work load is large and the number of operators is too low.

The relation between BOSS and the STATION exemplifies an "authority" relation; it is hierarchical in that, in the real world, the BOSS has more authority than STATION do.

Using D_LAB.2 instead of D_LAB.1 (and keeping D_ENV.1 and D_WARD.1), one obtains a new descriptive world model with constitutive hierarchical structure:

D_WORLD.2 = (D_HOSP.2, D_ENV.1)

D_HOSP.2 = (D_WARD.1, D_LAB.2)

D_LAB.2 = (D_BOSS.2, D_STATION.2).

D_LAB.2 will also contain descriptions of the relations between the constituent subsystems BOSS and STATION.
Also D_HOSP.2 will contain descriptions of the relations between D_LAB.2 and other entities (likewise for D_HOSP.1). These descriptions will be important in determining whether to realise an implementation where D_LAB.2 is explicitly present or not.

## 5.Conclusions

The type of models foreseen by BDSIM CPP are structured dynamic models of first and second order. (Delaney and Vaccari, 1989 ; Klir, 1985; Zeigler, 1976a,1976b)
The peculiarities of structured models, formed by a net of interconnected generative models, are that it is possible to represent different hierarchical levels eventually at different time scales and it is possible to manage descriptive complexity without a drastic increase of uncertainty. These features, as discussed in Vaccari (1998a,1998b,1998c ), are very important when modelling aspects of self-referential, anticipatory systems such as living systems (Maturana, 1981; Rosen, 1979) and social systems ( Luhmann, 1995,1990),.

## References

Delaney William, Vaccari Erminia. (1979). A Block Diagram Oriented Simulation Software System. Proc. of the 10th Annual Pittsburgh Conference, April 1979.

Delaney William et al. (1984). A System Theory Based Simulation Language. Cybernetics and Systems Research, vol.2 pp.111-115, R.Trappl

Delaney William and Vaccari Erminia, (1989). Dynamic models and discrete event simulation. Marcel Dekker. New York .

Klir George(1985). Architecture of Systems Problem Solving. Plenum Press, New York.

Klir George ( 1991). Facets of System Science. Plenum Press, New York and London

Oren *Tuncer* and *Zeigler* Bertrand.*(1979)*. *Conceps* for Advanced *Simulation* Methodologies. Simulation, March 1979

Luhmann Niklas. (1995). Social Systems., Stanford University Press, Stanford. .

Luhmann Niklas. (1990). Essays on Self-Reference. New York: Columbia University Press

Maturana *Humberto.(1981)*. Autopoiesis Milan Zeleny(ed) Autopoiesis: a Theory of Living Organisation, New York

Rosen Robert(1979). Anticipatory Systems in Retrospect and Prospect. Gen. Syst.Yearb, 24

Vaccari Erminia and Delaney *William.*(1974*a*). *Design Criteria for a General Purpose* High Energy Physics Experiment Simulation Program.: Computer Physics Communications, vol.7, pp.135-144.

Vaccari Erminia and Delaney William.(1974b). Toward the general purpose-special purpose simulation program. Proc. 5th Annual Pittsburgh on Modelling and *Simulation*.

Vaccari Erminia (1998a ). Knowing as Modelling. Cybernetics & Human Knowing vol.5, n.2 pp 59-72

Vaccari Erminia (1998b ).Some Considerations on cognitive modelling. In R. Trappl (ed.), Cybernetics and Systems'98, vol.2 pp 663-668

Vaccari *Erminia et al. (1998c ). Hierarchical Modelling: a Systemic framework.* In G. Farre and T. Oksala (eds), Acta Polytechnica Scandinavica, pp 179-184

Zeigler Bertrand (1976 a). The hierarchy of system specifications and the problem of structural inference, PSA 1976 vol.1. F. Suppe and P.D Asquith ( eds.) Philosophy of Science Association, pages 227-239 Michigan

*Zeigler Bertrand (1976 b). Theory of modelling and simulation.* John Wiley and Sons

Zeigler Bertrand (1984). Multifacetted System Modelling and Discrete Event Simulation. Academic Press, N.Y.