Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

HIERARCHICAL MODULAR MODELLING IN DISCRETE SIMULATION

M. Pidd and R. Bayer Castro

Department of Management Science
Lancaster University
Bailrigg
Lancaster LA1 4YX, UNITED KINGDOM

ABSTRACT

The increasing use of discrete simulation in modelling large and complex systems brings new challenges. One such challenge is the need to devise ways of developing modular approaches to reduce the complexity of model building and to make such models easier to maintain. This paper discusses an approach, known as selective external modularity, that builds on the work of Zeigler and others by adding the capacity to cope with emergent behaviour in composite models.

1 INTRODUCTION

Though discrete simulation is widely accepted as a useful way of understanding the behaviour of dynamic systems, its usefulness could be improved by better ways to model large systems. This has been the concern of a number of authors, most notably Zeigler (1976, 1984 and 1990), who calls this 'modeling in the large' and of Cota and Sargent (1992). Both suggest that modularity is the key to coping with the complexity inherent in large systems. That is, they suggest an approach based on *divide and conquer* (Pidd, 1996) in which the large system is fragmented into smaller sub-systems that are easier to understand. These sub-system models are then combined in some way or other in the expectation that the resulting aggregation will be an adequate representation of the system being simulated.

A further complication is that many large systems are inherently hierarchical. For example, command and control systems, whether used by the civil authorities such as the police service, or whether used by the military. That is, the command and control system is, in effect, a layer that sits above the physical layer made up of the main objects of the system that change state. These state changes are sometimes a result of internal operation at the 'physical' level, or may also be a result of intervention from the 'control' level. Many manufacturing systems can be viewed in the same terms: that is, there is a 'physical' system of machines and parts, with an imposed control layer that sets priorities and may intervene in the 'physical' layer. Air traffic control systems would be another example of such systems.

Various approaches have been proposed to the modelling and simulation of these hierarchical systems. Perhaps the most common, is the naive approach in which the control rules are hard-coded into the simulation models that make up the 'physical' layer. The disadvantage with this approach becomes clear when the model needs to be modified in some way or other. The modeller then must find the various fragments of the control code that are scattered around the simulation model. A better approach is a variation on data-driven generic models, in which the control rules are, as far as possible, separated from the event code and are represented by data. Thus, the simulation model reads the control data from a file (say) and then operates accordingly. This second approach at least separates the 'physical' logic from the control logic.

This paper takes the second approach somewhat further by proposing a hierarchical modular approach. In this, different layers may each be represented by simulation models that communicate with one another and which may intervene in one another. The approach is based on the DEVS approach of Zeigler (op cit), though with significant modifications that may upset the DEVS purists.

2 BUILDING MODULAR MODELS

2.1 Modularity

Given that this paper proposes a model building scheme based on inter-connected modules, it seems important to define what is meant by a module and to discuss how these modules may be connected. Cota and Sargent (1992) suggest that modularity is based on the two properties of *locality* and *encapsulation.*

- Locality is the notion that all information relevant to a design decision should be kept in one place, i.e. within a particular module.

- Encapsulation is a protection mechanism which, when combined with locality, allows information one module to be modified independently of other modules.

Zeigler (1984, 1990) argues that modularity should be regarded as an absolute property, in the sense that a model is either modular or not. In his terms, a modular model must satisfy both of the following conditions.

- The model must not directly access the state of any other model or component. That is, the description of its internal state and behaviour must contain no references to the internal states and behaviour of any other model.
- The model must have recognised input and output ports through which all interaction with the exterior is mediated.

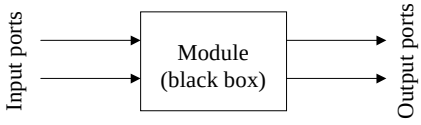

Figure 1: A Modular Model

Thus, as shown in Figure 1, Zeigler insists that a modular model must be regarded as a 'black box' which receives messages (external events) through its input port(s) and which sends messages (information about changes in its internal state) through its output port(s). The description of the 'black box' must be such that it is contextually independent. That is, its internal description must make no assumptions whatsoever about the origin of messages on its input port(s) nor about the destination of messages from its output port(s).

Given the properties of a modular model, it should be clear that modularity ought to make a model easier to understand and safer to modify. This is because changes in any design decisions are local (from the locality property) and are independent of the rest of the model, since they are encapsulated within a single module. A further advantage is that this modularity supports the re-use of model components, since modular models are defined with no direct reference to the state of their potential co-components. This improves the likelihood that modular models may be built, at least in part, from existing components. It also provides an attractive way of introducing hierarchical components into simulation models.

2.2 The coupling of modules

The whole point of modularity schemes in discrete simulation is to provide ways of building large models safely and quickly. Thus there needs to be some way to link the modules together. Zeigler (1984, 1987, 1990) suggests that the process of creating a new composite model from a set of modular components be known as *coupling*. Thus, a *coupling scheme* is a specification of the way in which the input and output ports of the modules are coupled. The result is a *composite* model.

For example, consider Figure 2, which represents a composite model, known as a Buffered Machine. This, in turn, consists of two components, known as Buffer and a Machine.

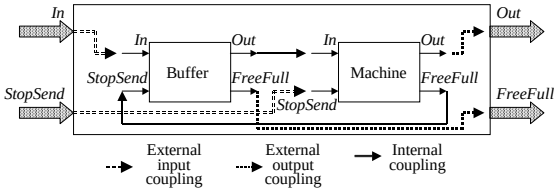

Figure 2: A Coupling Scheme

- Through its first input port *In* a Buffer receives parts to be stored. Through its second input port *StopSend*, it is instructed either to not send parts (for some reason or other that is unknown) or to resume its sending of parts. Parts are sent from the Buffer through its first output port *Out*. Its second output port *FreeFull* sends a control signal to indicate whether it is full, or not.
- The machine processes one part at a time, having two states *Full* and *Free*. Its output port *FreeFull* is used to send external messages about its state. Its input port *In*, receives parts, which are then despatched through its output port *Out.* Like the Buffer, the Machine receives control information through its other input port, *StopSend*.

Figure 2 shows that this composite model of a Buffered Machine has three types of coupling that compose its coupling scheme.

- External input coupling: the coupling of the input ports of the composite model with the input ports of the components. Thus, the input ports *In* and *StopSend* of the Buffered Machine link with the input ports *In* of the Buffer and *StopSend* of the Machine component.
- External output coupling: the coupling of the outport ports of the components to the output ports of the composite model. Thus, the output ports *Out* and *FreeFull* of the Buffered Machine are linked to the output port *Out* of the Machine and *FreeFull* of the Buffer.
- Internal coupling: the coupling of the input ports of the components with the output ports of the other

components. Thus, the output port *Out* of the Buffer is linked to the input port *In* of the Machine. The output port *FreeFull* of the Machine is linked to the input port *StopSend* of the Buffer.

The resulting composite model, the Buffered Machine, is also in a truly modular form and may thus be used as a model component in a yet larger composite model. It is, however, important to realise that the composite model (Buffered Model, in this case) has no behaviour of its own. That is, a composite model displays no emergent behaviour. Instead, its behaviour is entirely explicable from its components and their coupling scheme.

2.3 Delayed binding

Zeigler (1990) suggests that this modularity should be implemented by *delayed binding*, which assumes that the models making up the composite via a coupling scheme are unknown to one another. That is, if a modular model places a value on one of its output ports, the destination of this output is unknown without the coupling scheme. No receiver is defined as a part of the modular model, only as part of the coupling scheme.

Thus, coupling schemes implement a form of delayed binding , which is distinct from dynamic (run-time) binding as implemented in object oriented languages such as C++ and Java. (See Pidd (1995) and Pidd & Cassel (1998) for examples in discrete simulation). In run-time binding, an object may send a message to another without knowing the precise class of the receiver. However, run-time binding still requires the receiver to belong to a class that descends from some earlier (or base) class known to the sender. In delayed binding, which is the essence of Zeigler's approach, there is total contextual independence between the sender and receiver.

Delayed binding means that hierarchical models may be tested at every stage of their construction by coupling them with test models. This permits a form of partial validation. However, the process of developing these composite models can be rather daunting and may seem very complex in simulations of any scale, especially in those that involve much interaction between the modular models.

2.4 Modularity and object orientation

As mentioned above, conventional object orientation provides some support for modularity and this has been exploited in a number of discrete simulation systems (see for example, Joines et al (1992) and Fishwick (1992)) from SIMULA onwards. Pidd (1995) suggests that three aspects of conventional object orientation are useful in discrete simulation.

- *Class mechanisms*: variable types are declared as objects which may inherit part of their definition from previously declared object types.
- *Encapsulation*: by which data and the functions which change that data are kept together in the same place.
- *Polymorphism*: in which the same instruction may be implemented quite differently by members of different classes. This is usually implemented via run-time binding.

Conventional object orientation provides locality if all variables are implemented as classes with their own internal state data and internal methods, thus keeping all state data private.

In addition to meeting the requirements of Cota and Sargent (op cit) for encapsulation and locality, conventional object orientation can be used to implement input and output ports via message passing. Conventional object orientation also supports model building through the aggregation of existing components, making it a sensible starting point for hierarchical modelling.

However, as discussed in the previous section, there is no support for delayed binding in conventional object orientation since the sender must define which object is to receive a message. This definition may be incomplete (some descendent class of a known potential recipient, for example), but something must be known. That is, the level of contextual dependence is higher than would be ideal for hierarchical modular modelling. In conventional object orientation, modular models do need to be acquainted with one another, preventing the development of ab initio coupling schemes of the type used in DEVS.

3 A MORE WHOLISTIC APPROACH: SELECTIVE EXTERNAL MODULARITY

DEVS and other modular schemes have all the virtues and share many of the vices of reductionism. Reductionism is the belief that a complex system may be decomposed into its constituent parts without any loss of information, predictive power or meaning. By contrast, wholism (sometimes, holism) is a view that systems possess emergent properties that cannot be explained solely by recourse to the properties of the components that compose the system. Thus, emergent properties are only meaningful in the context of the whole and not in the parts. Checkland (1984) suggests that the shape of an apple is one such emergent property. It cannot be explained at a molecular level and is not meaningful at that level. This is because systems are properly considered as comprising components and their organisation and interaction.

The hierarchical modelling approach described earlier is essentially reductionist. This is because, as already mentioned, a composite model (resulting from the coupling

of component modular models via a coupling scheme) does not possess behaviour of its own. The resulting new model is no more than a set of components plus a map that specifies how the model components are coupled together, and also which of their input and output ports will serve as ports of the resulting composite model. Uhrmacher (1997), for example, discusses how higher level models in DEVS can be reduced to the behaviour of lower level, atomic models.

3.1 The job-shop example

To illustrate the problem, consider a job-shop made up of *N* identical buffered machines. For simplicity, assume that each buffered machine has a single input port *In* and a single output port *Out*. Different parts arrive at the job-shop requiring different machines and different processing sequences. Thus, following an operation at machine *X*, the part may be ready to leave the system or may need to be passed to another machine for processing. Figure 3 shows a wholly reductionist coupling scheme for *N*=4 machines. The need to wire every machine to each other machine produces the horrendous appearance, which would be completely out of hand if *N* were large. The result of this wiring is that, whenever a part needs to move from the output port, a reference needs to be sent to all *N* machines (allowing for re-work) plus to the composite model. This makes *N*+1 references, of which only one will be acted upon. Were several (say, *m*) machines able to complete the same operation, then some additional mechanism is needed so that of the *m*<*N* machines able to perform an operation, only one actually does so.

A refinement is shown in Figure 4 in which an additional atomic model, a *controller*, is responsible for routing a reference to the appropriate machine. Though this is an improvement, there remains the problem of dimensioning the controller's output ports. The controller needs to know how many machines there are - that is, it needs to be acquainted with some properties of the composite model, which cuts across delayed binding.

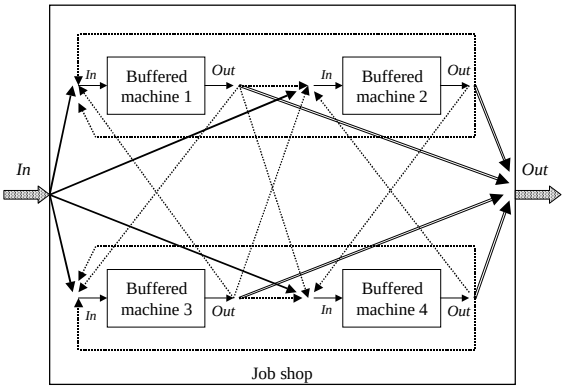

Figure 3: Job-shop Under DEVS

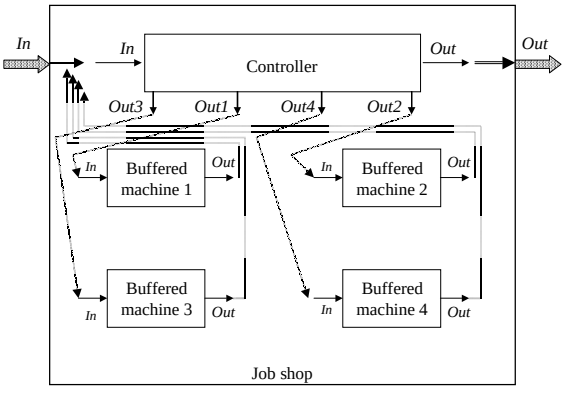

Figure 4: Job-shop with Controller

3.2 Incorporating wholism

To achieve and take advantage of the modelling of emergent properties, some relaxation of the usual constraints associated with modularity is required. The great difference is that an aggregate/composite model must be acquainted with its model components to achieve wholism. The alternative of no acquaintance has no significant advantages over the modular approach used in DEVS-Scheme based implementations.

Under a purely reductionist approach, any effects of a composite system at level X of the hierarchy resulting from the interaction of *N* components at level X-1 must be described by building and connecting at least one additional component *N*+1 at the component's level. On the other hand, under a wholistic approach, the effects described by model *N*+1 at the level X-1 are promoted one level in the hierarchy, and become characteristics of the newly built model. If we do not allow for acquaintance, there is no advantage in this change of level because the emergent effects cannot be described in terms of the interaction between its components.

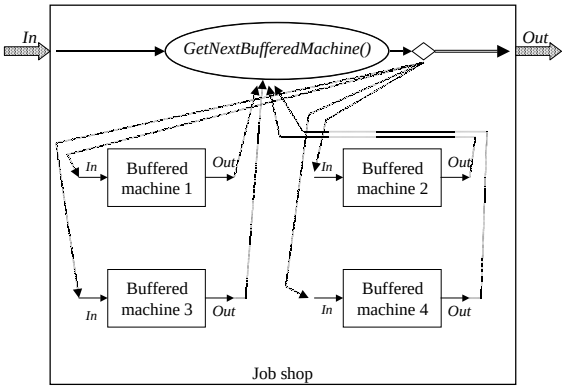

Figure 5: Job-shop, RuiSEM Model

Although this approach reduces the level of modularity, it presents significant advantages in terms of flexibility and ease of expression. To illustrate this, consider the wholistic representation of the job-shop in Figure 5. A pure DEVS coupling scheme would represent the job-shop as a composite with no methods of its own. Figure 5 shows that the job-shop level has a new method, *GetNextBufferedMachine()*, that replaces the controller of Figure 4. This method determines which machine performs the next operation in the sequence of any part. Thus, when a machine has completed work on a part and places a reference on its output port *Out*, the method *GetNextBufferedMachine()* is called, with reference as an argument. The reference is then sent to the input port *In* of the appropriate machine or to the output port *Out* of the job-shop.

In effect, such a wholistic approach is implemented via a form of run-time coupling. The actual connection between the buffered machines being determined at run-time and not before. This sacrifices the pure modularity of a DEVS-type approach in model building, but the resulting job-shop composite model is itself modular, having locality and being encapsulated. The resulting wiring of the internal buffered machine models with a large number of buffered machines is much simpler than would be needed if internal modularity were enforced. External modularity, though, remains in place. This should ensure that the simulation also runs faster, since there is no need to search through all possible buffer machines (plus the composite) whenever an operation is completed on a machine.

4 RUISEM

RuiSEM is an object oriented executive, written in C++, that implements the idea of selective external modularity as discussed in section 3. It assumes that composite models are built by coupling model components (as in DEVS), but it also assumes that the composite model has behaviour of its own and is acquainted with its own components. The components are, however, unacquainted with the composite model. Full details of RuiSEM will be given in a subsequent paper, but its use is as follows.

1. Specify the basic models from which larger ones are to be built.
2. Specify the coupling scheme - how the internal models are to be coupled together as components and with the result composite model.
3. Specify additional behaviour that results from this aggregation.

RuiSEM provides an abstract base class, *Model*, from which all components may be constructed with the expectation that an actual model will have a t least the following.

- A set of state variables, including *Phase* to indicate the current state of the model and *Sigma* to show how long it will remain in this state if no external transition is triggered.
- A set of input methods to describe how the model changes its state when it receives certain input values. Each input method typically places the model in a new *Phase* and *Sigma*, thus (re) scheduling its next internal transition. These input methods implement the concept of input ports of modular models.
- A set of output methods that return relevant information about the model's internal state. These methods implement the concept of output ports of modular models.
- A set of internal methods specifying the next state of the basic model. The scheduled output methods, in conjunction with the scheduled internal methods, characterise the next internal transition.

4.1 The consequences of selective external modularity

Since a composite level now has behaviour of its own, this additional behaviour must be modelled in some way or other. Within RuiSEM, this is managed in either or both of the following ways.

1. By coupling the output methods of the components with methods that belong to the owner, or composite, model.
2. By *trapping* the output from the components.

The coupling of the output methods of the components to the composite is achieved by the introduction of methods that belong to the composite level, such as the *GetNextBufferedMachine()* member function shown in figure 5. This receives the output from the lower level buffered machines output ports and decides which machine (if any) will carry out the next operation. It does not send

the part directly to this next buffered machine, but asks the simulation executive to notify the input port of the buffered machine.

The introduction of behaviour at composite level also means that the external output coupling scheme is no longer a simple mapping between the output ports of the components and the output ports of the composite model. *Trapping* is used to allow a certain level of interference from a higher level (composite model) to a lower level model (component model). The idea behind trapping is to transfer control to a higher level model when something is about to happen (but before it happens) in a lower level model. When this happens, the trapping method of the composite model performs its own state changes and decides if the event of the trapped component is allowed to occur, or if it is to be postponed, cancelled or modified.

The trapping method is responsible for doing several tasks that in the previous cases were carried out by the executive:

- Generates the output of the imminent model by triggering the appropriate output method.
- Determines the destination model of the output produced.
- Asks the executive to trigger the appropriate input model of the destination model, with the output as argument.
- Changes the state of the trapped model by asking the executive to trigger any desired input method with the appropriate information as argument and/or to trigger an internal transition method.

5 DISCUSSION

RuiSEM supports hierarchical modular model building and allows for a certain level of delayed binding (although lower than the one achieved in DEVS-Scheme based implementations (Zeigler 1987,1990)). A model can only be acquainted with its components, so that when a component produces a value as an output the actual destination only is specified when it becomes a component in a larger composite model. It allows for object-oriented features such as inheritance, polymorphism and encapsulation to be properly exploited. This was accomplished because the ports of modular models are implemented as methods.

In addition, RuiSEM allows a certain level of control from a composite model over its components. This is achieved by encapsulating the input methods of components in input methods of the composite model; and by trapping output and internal transition methods of the components, so that control is passed to the composite model. This level of control is achieved by usurping tasks and responsibilities normally assigned to the executive, and by relaxing some of the constraints associated with modular systems.

The approach supports a wholistic modelling view where emergent properties are represented at their natural level in the hierarchy - naturalness of description. However, some of the coupling is more complex, demanding some coding. Nevertheless, situations where the code is more complex would also demand the coding of emergent effects as at least an additional component model. As a benefit, coupling is more effective given that we can use run-time coupling to determine the actual destination of a value at run-time, instead of sending it to all possible destinations. As a cost, the potential for parallelism is reduced.

ACKNOWLEDGEMENTS

The work of Rui Bayer Castro is supported by the Sub-Programa Ciencia e Tecnologia do 2 Quadro Comunitario de Apoio, Portugal.

REFERENCES

Checkland P.B. (1981) *Systems thinking, systems practice*. John Wiley & Sons, Chichester.

Cota B.A., Sargent R.G. (1992) A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation.* Vol. 2, No.2, pp 109-129 April 1992.

P.A. Fishwick (1992) SIMPACK: getting started with simulation programming in C and C++. In *Proceedings of the 1992 Winter Simulation Conference*, Arlington VA, Dec 1992, ed. J. J. Swain, D.Goldsman, R. C. Crain, and J. R. Wilson, 104-114. Institute of Electrical and Electronics Engineers, Piscataway, NJ.

J.A. Joines, K.A. Powell and S.D. Roberts (1992) Object-oriented modelling and simulation in C++. In *Proceedings of the 1992 Winter Simulation Conference*, Arlington VA, Dec 1992, ed. J. J. Swain, D.Goldsman, R. C. Crain, and J. R. Wilson, 104-114. Institute of Electrical and Electronics Engineers, Piscataway, NJ.

Pidd, M. (1995) Object-orientation, discrete simulation and the three-phase approach. *Journal of the Operational Research Society*, 46, 362-374.

Pidd, M. (1996) *Tools for thinking: modelling in management science*. John Wiley & Sons Ltd, Chichester.

Pidd, M and Cassel, R.A. (1998) Three phase simulation in Java. In *Proceedings of the 1998 Winter Simulation Conference*, Washington DC, Dec 1998.Uhrmacher, A. M. (1997). Concepts of object- and agent-oriented

simulation, *Transactions of the Society for Computer Simulation International*. Special issue: multi-agent systems & simulation, Vol 14, Number 2, June 1997, pp. 59-67.

Zeigler, B. P.(1976). T*heory of modelling and simulation*. John Wiley & Sons, New York.

Zeigler, B. P.(1984). *Multifacetted modelling and discrete event simulation*. Academic Press, London and Orlando, Florida.

Zeigler, B.P. (1987). Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment. *Simulation*, vol. 50, pp. 219-230. Zeigler, B. P. (1990). *Object-oriented simulation with hierarchical, modular models: intelligent agents and endomorphic systems*. Academic Press, Boston.

AUTHOR BIOGRAPHIES

MIKE PIDD is Head of the Department of Management Science and Professor of Management Science at Lancaster University in the UK. He is best known for two books, *Computer simulation in management science* (now in its fourth edition) and *Tools for thinking: modelling in management science*; both published by John Wiley. He is active in researching simulation methods related to modularity and object orientation. He acts as consultant to a number of private and public sector organisations in Europe.

RUI BAYER CASTRO is a Ph.D. student in the Department of Management Science at Lancaster University, whose research focuses on hierarchical approaches to discrete event simulation and object orientation. He received the (5 year) Lic. degree in electrical and computer engineering from the Faculdade de Engenharia da Universidade do Porto, Portugal in 1993 and an MSc in operational research from Lancaster University in 1994.