WELL--FORMED MODELS FOR DISCRETE SIMULATION

Douglas Seeley

University of British Columbia

Simulation Correctness

Discrete simulation models of non-trivial complexity are notoriously difficult to debug. In the first place usable systems descriptions of the real world component being modelled are rarely in existence (although these are emerging in the realm of computer systems). (1,2) Secondly individuals often have incomplete knowledge of the system in question (indeed, much of the value of a simulation model, like that of the Critical Path Method, is in bring individuals together in order to agree upon a systems description). Even though structured approaches exist for model description, such as the EVENT routines in SIMSCRIPT and the ACTIVITY routine in SIMULA, are available, many and varied detailed traces are required in order to test the simulation logic. Often, in complex models, many subtle conditions will remain undected for some time, and yet affect model behavior in such a way that outputs remain plausible.

This paper describes a procedure and system for ensuring the logical consistency of a discrete simulation model. The initial system, WORMS I, is intended primarily for models that will be implemented in an event-oriented language such as SIMSCRIPT. In such a language, EVENT routines must be created which specify under what conditions will the model change state, and what will be the resultant conditions and/or sub- events in the model. For somewhat complex situations it is a non-trivial matter to implement these routines correctly, especially with the absence of any convenient language for specifying these events. The WORMS system provides such a descriptive facility, analyses their logical (transition) consequences, and allows the modeller to make alterations to his model interactively.

Graphical Simulation Models

In the past, several graphical models have been developed in order to represent the concurrent activities of discrete simulation and computer multiprocessing. For a survey see the recent paper by Baer. (3) Many of the models have been designed specifically for parallel computation and for the detection of deadlock in resource-sharing systems. Petri's models, commonly known as Petri nets, have been widely used as a reference point in exploring such systems. This paper is not an exception.

Petri nets (see Baer (3) or Holt (2) for formal descriptions) are used to describe the event structure of a system and the conditions that are required for, and result from each event. An event is represented by a vertical bar into which lead arcs from circles which represent all of the conditions which must hold in order that the event take place. Arcs from the vertical bar lead to those system conditions which result. The holding of a condition is indicated by the placing of a token on its location. All paths through the net must encounter events and conditions alternately.

In the WORMS system a modification of Petri nets to a collection of "event graphs" is used (to be more precise, these could be called event transition graphs, since a single event may require several graphs in order to specify the transitions of which it is composed). A Petri net can become a complex and difficult graph structure to interpret for non-trivial models; moreover, the concept of an allowable system state does not appear in the formulation. In WORMS however, the event transition logic is decomposed into a collection of event graphs. In this representation an event (change of system state) appears as either a box (indicating the completion of a process-in-time) or as a vertical bar indicating an instantaneous state transition (sub-event). However, the arcs leading into and out of the event now relate not to loosely defined conditions but exactly to system states.

This is done in the following manner: A discrete system is composed of a collection of components (entities) each of which will be in one of a variety of substates. For instance a service facility may be busy, idle or blocked and a queue may be a particular length. A vector specifying the substates of each system component will therefore represent a system state. In the nodes

on either side of the event graph are the specifications of the substate or substate range of some of the system components. Components that are not specified may be in any substate valid for them, and these substates will remain unchanged by the event. Consequently, the event graph represents a mapping from one range of system states into another. The collection of events graphs for the system therefore, corresponds exactly to the transition function of the finite-state automata analogous to the discrete simulation model being considered. It will be shown in a later example how natural this representation is for event-oriented modelling.

Well-Formed Models and the WORMS Systems

Petri nets were used primarily to study concurrency and conflict in system descriptions. A concept of a "live net" was used to describe a net where all transitions can fire from a given initial state. WORMS considers a more comprehensive notion of "well-formedness". In debugging and testing an event-oriented simulation model, a source of many errors is the precision with which the event routine is coded. The event routine usually corresponds to the group of event graphs which are concerned with the same time process (e.g. the completion of service from a facility, or an inter-arrival process, etc.). In non-trivial models, the specification of this state mapping can be complex and the computer code prone to logical error, error which sometimes may not be easily discovered. WORMS is a tool that organizes this part of model building and helps the modeller anticipate most such logical errors.

Once a model has been defined by specifying component substates, the collection of event graphs, transitions, and some initial states, WORMS will determine: (a) whether there are any trapping states (a state from which the system cannot proceed); this will indicate either the presence of an inadequate event graph, the need for an additional one, or an actual deadlock possibility in the model, (b) whether an event transition is actually used, also indicating an improperly specified transition function, (c) any ambiguous states, i.e. states that may be applied to more than one transition of the same event type, and (d) and what states the system cannot enter.

Whenever conditions (a) and (c) above occur, the user can redefine the model usually by redefining the event graph that led to the problem and reinitiate the analysis. The appearance of condition (b) will require that the modeller backtrack through the transition logic in order to determine why none of the states on the L.H.S. of the event graph have been reached in the model. A similar procedure is required when states that are possible in the real system are discovered in the list of those not reachable from the current model. When all states reachable in the real system are reachable in the model, and when the collection of event graphs are all used and unambiguous, the model may be said to be "well-formed" or logically consistent. At this point the event-oriented simulation model can be programmed with confidence. However, this is not to say that the simulation is a valid one; validation must still be carried out on the model's behavior and on its embedded assumptions.

Extensions

1. WORMS I uses a computer-word-per-state space. In order to keep state-space demands on computer memory manageable, it would be useful to construct tables that encode large ranges of states in a simple manner. Then, whenever a transition results in, for instance, a "+n condition" the reachability of this entire range could be expressed succinctly. A technique related to this has been implemented for Markov Chains by Irani and Wallace (4)

2. It is clear that the analysis that WORMS provides could act as a syntatic-front-end to an actual simulation language. In fact, all that is additionally required, is the specifications of the time-processes in order to complete the model's definition. A program could then automatically implement a discrete event simulation of the model.

3. The most important extension would be to allow the description of model in a process-oriented (e.g. SIMULA or GPSS) manner. A process description normally entails a sequence of events, including their before and after conditions. It appears that it is the way process descriptions factor and organize the states of the system, that makes modelling in this fashion easier than event-oriented modelling. If a process-oriented analysis could be achieved from an elegant graphical description, then model building would be greatly expedited.

References

1. Tsichritzis, D., "Modular System Description", University of Toronto, Department of Computer Science, Technical Report No. 33.

2. Holt, R.C., "Some Deadlock Properties of Computer Systems", Computing Surveys, Vol. 4, No. 3, Sept. 1972.

3. Baer, J.L., "A Survey of Some Theoretical Aspect of Multiprocessing", Computing Surveys, Vol. 5, No. 1, Mar. 1973.

4. Irani, K.B., Wallace, V.L., "A System for the Solution of Simple Stochastic Networks", University of Michigan, CONCOMP Technical Report No. 14.