INFORMATICA, 1998, Vol. 9, No. 3, 297–314

© 1998 *Institute of Mathematics and Informatics, Vilnius*

Conservative Simulation for Discrete Event Systems

Alexandru CICORTAŞ
Faculty of Mathematics, University of West
B-dul V. Pârvan 4, 1900 Timişoara, Romania
e-mail: cico@disco.info.uvt.ro

Received: February 1998

Abstract. More real systems have many components and their simulation requires significant execution times. The practical needs have conducted to distributed simulation rather than sequential method. Asynchronous parallel discrete event simulation (PDES) is studied and its methodology is presented. The paper presents the conservative methodology of PDES and illustrates it with a suggestive particular application: Virtual Assembly Cells.

Key words: parallel algorithm, distributed simulation, synchronisation, manufacturing, virtual assembly.

1. Introduction

The basic technical concepts of distributed simulation for discrete event systems are reviewed and tied to a specific application: assembly system which uses *Virtual Assembly Cells* (VAC). By defining the methodology within the context of proposed application, this paper serves both as a tutorial to conservative Parallel Discrete Event Simulation (PDES) and as an introduction to VAC simulation modeling. VAC is a suggestive application to illustrate some PDES concepts. The particularities of VAC's application simplify its conservative simulation.

For studying a system it must be modeled. Its model can be one of the several types: conceptual, declarative, functional, constraint, spatial, or multimodel (Fishwick and Ziegler, 1992; Fishwick, 1993). The last type (multimodel) permits to take into account the integration of basic model types and to create a model of component models, where each component model represents a level of abstraction for the system. Often, in simulation is used a spatial model. Spatial models can be executed in several ways including time slicing, event scheduling and parallel and distributed.

The paper is organised as follows. In Section 2, for the conservative protocol are defined basic characteristics and are illustrated. In Section 3 are given main aspects of assembly process in the manufacturation. The data bases proposed for structure product representation with their main attributes are presented. The model proposed is presented in object-oriented concepts.

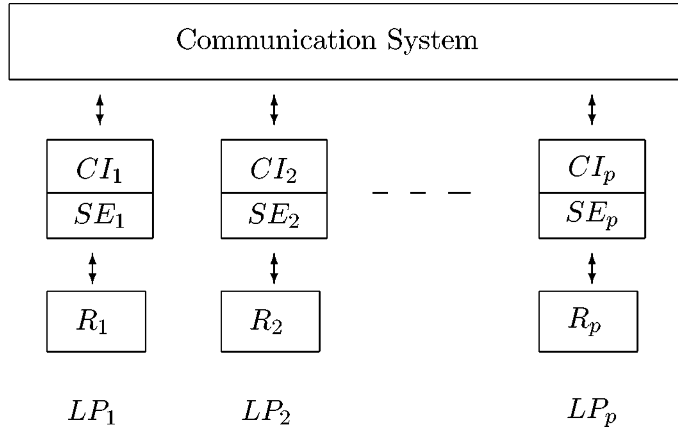

Fig. 1. System simulation by logical processes.

1.1. *Logical Process Simulation*

Some simulation strategies divide the global simulation task into a set of communicating *logical processes* (LPs), trying to exploit the parallelism inherent among the respective model components with the concurrent execution of these processes. We can thus view a system simulation, as the co-operation of interacting LPs, each of them simulating a subspace of space-time which it can be named as region with an event structure. Generally a region is represented by the set of all events in a sub-epoch of the simulation time, or the set of all events in a certain subspace of the simulation space (Bagrodia *et al.*, 1991). The basic architecture of a system simulation by its logical processes (the definition of LP can be found in (Bagrodia *et al.*, 1991)) can be viewed in Fig. 1, see (Ferscha and Tripathi, 1996).

The set of logical processes executes event occurrences synchronously or asynchronously in parallel.

The *Communication System* (CS) provides that the LPs send messages one to other and by these messages allow to synchronize their local activities.

Every LP_i has assigned a *region* R_i (Cicortaş, 1997; Bagrodia *et al.*, 1991), as a component part of the simulation model upon which the *simulation engine (SE_i)* operating in event driven mode, executes *local* and generate remote event occurrences, thus processing a *local clock* (*Local Virtual Time* (LVT)).

For every LP_i exists a fixed subset of the state variables $S_i \subset S$, disjoint to other state variables assigned to other LPs. In each LP_i are processed two event types *internal* events and *external* events; the internal events belong to S_i while the external events belong to $S_j,\ i \neq j$. The *Communication Interface (CI_i)* attached to the SE_i allow sending and receiving external messages.

With respect to the advancement of event executions, we dispose for two classes of protocols, that have a conservative or optimistic in event processing. Both are based on message sending; these messages carrying causality information that has been created by

one LP and affects one or more LPs. For preventing global event causality violations, must define appropriate rules.

In the conservative protocol, by specific constructions, the process evolution is blocked if there is the chance to process an unsafe event (i.e., one event for which causal dependencies are still pending). In such way, we can prevent causality errors occurring.

In the optimistic protocol, if an premature event is detected (i.e., an event with the timestamp lower than the TVL of PL), the SE redo the simulation for those events that local events are inconsistent with causality conditions produced by other LPs.

1.2. *Synchronous and Asynchronous Simulation*

Were implemented two LP simulations: synchronously and asynchronously.

In a particularly synchronous LP simulation, all the LPs' local clocks evolve on a sequence of discrete values $(0, \Delta, 2\Delta, 3\Delta, \ldots)$. In a such way the simulation evolves in according to a global clock. Every LP must process all the events in the time interval $[i\Delta, (i+1)\Delta)$ before any processing of events with occurrence time $(i+1)\Delta$ and after.

Asynchronous LP simulation relies on the existence of events that occur at different simulated times that no affect on to other. By concurrent execution of those events we can reduce the simulation time by comparison with the sequential simulation time. A very difficult problem in asynchronous LP simulation are the local causality errors. If every LP process the events in nondecreasing timestamp order (i.e., *local causality constraints*, (Fujimoto, 1990)), then we have no causality error.

For asynchronous LP simulation the local causality violation is treated in two different ways: conservative methods that strictly avoid local causality violations while the optimistic methods use the chance to process the events even if the local causality will be violated.

2. Conservative LP Simulation

The architecture of a conservative logical process LP_k is given in Fig. 2. From the original works of Chandy Misra and Bryant, the LP conservative simulations are referred as Chandy-Misra-Bryant (CMB) protocols. Causality of events across LPs is preserved by sending timestamped (external) event messages of type $< ee@t >$, where ee denotes the event and t is a copy of LVT of the sending LP at $(@t)$ the instant when the message was created and sent. $(t = ts(ee))$ is called the *timestamp* of the event.

In a conservative simulation, allow to process only *safe*, i.e., the events up to a LVT for which the logical process has been guaranteed do not receive external events (messages) with their timestamp lower than the LVT of logical process. Moreover, all events internal and external, must be processed in chronological order. Thus also the logical process produces messages in chronological order. The communication system preserves the order of messages sent by the sender LP to receiver LP in a FIFO order, as in Fig. 2. On this basis we obtain correctness. The conservative simulation has a static topology, where the FIFO communication channels play an appropriate role.

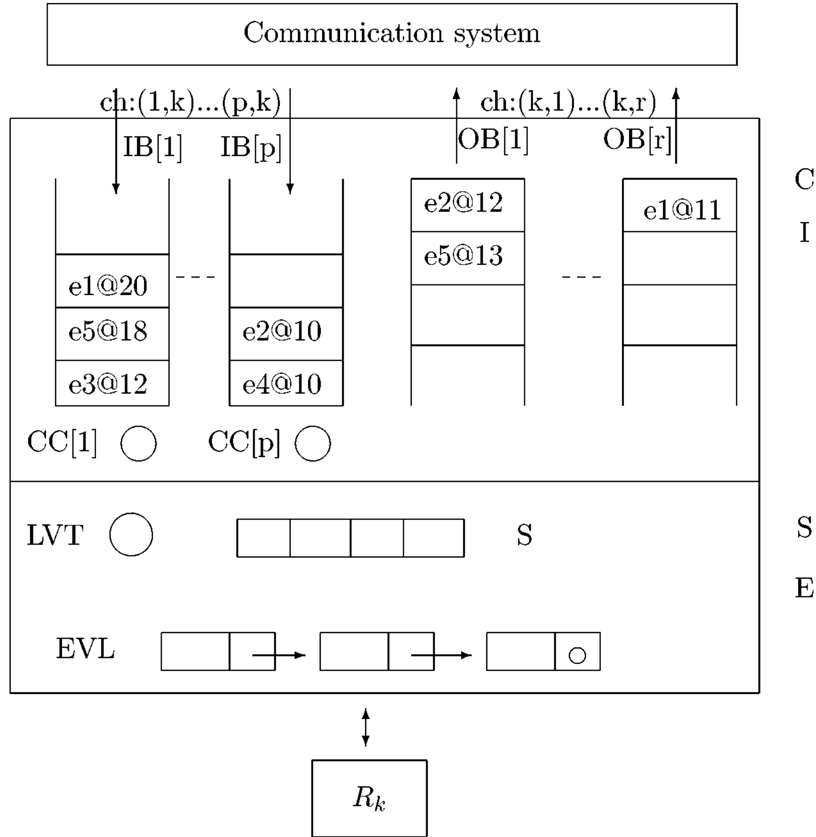

Fig. 2. Architecture of a conservative logical process LP$_k$. CI – Communication Interface, SE – Simulation Engine.

From Fig. 2, the communication interface of the logical process PL_k, has one input buffer $IB[i]$ and a channel clock $CC[i]$ for every channel $ch_{i,k}$ pointing to the PL_k. $IB[i]$ stores in a FIFO order arriving messages from PL_i and $CC[i]$ holds a copy of the timestamp of the message at the head of $IB[i]$; initially $CC[i] = 0$. Define the LVTH as LVTH= $\min_i CC[i]$ the time horizon until which LVT is allowed to progress by simulating internal and external events, since no external events can arrive with timestamp smaller than LVTH.

REMARK 1. In our vision, the Simulation Engine (SE) is the main part that manages the logical process. The Communication Interface (CI), only furnish the communication way between this logical process and other logical processes.

DEFINITION 1. The $CC[i]$ holds a copy of the timestamp of the message at the head of $IB[i]$. Define the LTVH as the time horizon given by the communicating logical pro-

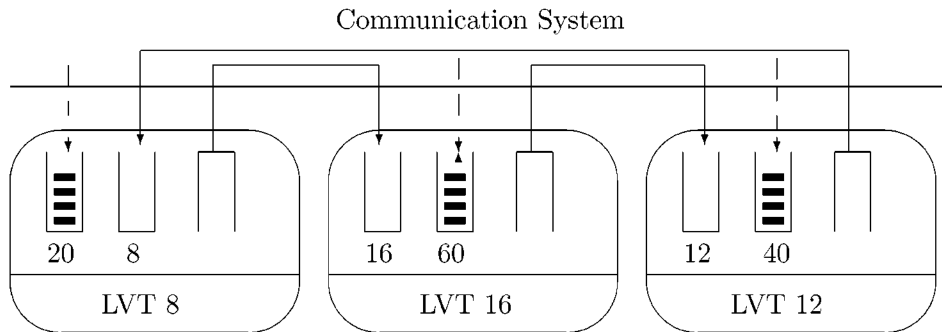

Fig. 3. Deadlock and memory overflow.

cesses from which the current logical process PL_k receives messages, as

$$\text{LVTH} = \min_{i \in \{1,2,\ldots p\}} CC[i].$$

PROPOSITION 1. The Simulation Engine (SE) processes all internal events from EVL that have their timestamp lower than TVLH. Also are processed all messages from $OB[i]$ whose timestamp is lower than TVLH. The TVL is advanced to the value of the higher timestamp of the (last) event (it can be an event from EVL or a message which has the timestamp lower than the TVLH) processed.

More exactly it is important to outline that the TVL has the value of the *last* event processed (from the EVL or message sent from the IBS). The mean of *last* is the higher timestamp of these events that were processed.

The Simulation Engine ($Seek$) dispose now for the value of TVL and now can process some messages received from other LPs, in the Ibs. The condition that must be satisfied for processing of a such message is given in the following proposition.

PROPOSITION 2. For message processing we have two states:

- If the TVLH value is given by a $CC[i]$ whose queue is not empty, then the message in the head of the queue will be processed. The message processing consists eventually from inserting in the EVL of new events, at their appropriate places based on their timestamps. After that is released a new evaluation of the TVLH and the processing is resumed.
- If the TVLH value is given by a $CC[i]$ whose queue is empty, then the processing is blocked, while a new message will arrive at this queue and then is released a new evaluation of the TVLH.

In literature are given many procedures for basic algorithm of conservative simulation of LP, one of them is given in (Ferscha and Tripathi, 1996). Using this blocking policy,

appear two problems that must be solved, these are *deadlock* and *memory overflow*, as in Fig. 3.

Each LP is waiting for a message to arrive, however, awaiting it from a LP that is blocked itself (deadlock). The input buffers can grow unpredictably, causing memory overflow. For overcome the vulnerability of CMB protocol, have been proposed several methods, that can be clustered in two categories: *deadlock avoidance* and *deadlock detection/recovery*.

2.1. *Deadlock Avoidance*

By sending *nullmessages* of the form $< 0@t >$, where 0 denotes a nullevent (event without effect), in the original protocol, the deadlock can be prevented. A nullmessage is not related to the simulated model it only serves for synchronozation purposes. Essentially it is sent on every input channel as a promise not send any other message with smaller timestamp (as t from the nullmessage) in the future. It is launched whenever an LP processed an event that did not generate an event message for some target LP. The receiver LP can use this (implicit) information to extend its LVTH and by that become unblocked. In our example (Fig. 3), after the LP in the middle would have broadcast $< 0@16 >$ to the neighbouring LPs, both of them would have chance to progress their LVTH up until time 16, and in turn issue new event messages expanding the LVTHs of other LPs, etc. The nullmessage protocol can be guaranteed to be deadlock free as long as there are no closed cycles of channels, for which a message traversing this cycle cannot increment its timestamp. This implies, that simulation models whose event structure cannot be decomposed into regions such that for every directed channel cycle there is at least one LP to put a nonzero time increment on traversing messages cannot be simulated using CMB with nullmessages.

The protocol extension with the nullmessage facility is straight-forward to implement, it can put a burden of nullmessage overhead on the performance of the LP simulation. Optimisations of the protocol to reduce the frequency and amount of nullmessages, e.g., sending them only on demand (upon request), or only when the LP becomes blocked have been proposed.

One problem that still remains with conservative LP simulation is the estimation when it is safe to process an event. The degree to which LPs can *look ahead* and predict future events plays a critical role in the safety verification and as consequence for the performance of conservative LP simulations.

DEFINITION 2. An LP has a **lookahead** ε if the outputs of LP in any interval $[t, t+\varepsilon]$ is a function only of its state at instant t and is independent of its inputs in that interval. In other words, the LP can predict that the timestamps of all events that arrive later (after t) and are no smaller than $t+\varepsilon$ for all $t \geqslant 0$.

In the example of Fig. 3, if the LP with the LVT 16 could know that processing the next event will certainly increment LVT to 19, the nullmessage $< 0@19 >$ (so called

lookahead of 3) could have broadcasted as further improvement on the LVTH of the receivers. The value ε of the lookahead, must come from the underlying simulation model and enhances the prediction of future events, which is necessary to determine when it is safe to process an event.

3. Virtual Assembly Cells

3.1. *Introduction*

All the followings are based on the (Cicortaş, 1997).

In the following we are not use the technical details for our problem. All needed elements will be introduced as soon as will be used. The our objective is to define the assembly process in a factory that assemble products and their compound elements from component elements.

The products have a structure that can be assimilated as a tree whose root is the product (or any compound element if it is taken as product). Frequently a lot of compound elements are common to many products. In the manufacturing process can be distinguish two disjoint phases:

- manufacture component elements;
- assemble compound elements (include products) from component elements.

There are many component elements that have no a proper life in manufacturation process, they are manufactured as soon as they are used and incorporated in their compounds elements, without being kept in warehouse. This kind of component element will be neglected. On the other side we have component elements that have their life and are kept in warehouse after their manufacturation and are taken from, as soon as are necessary in the assembly process of their compound elements.

3.2. *Compound Structure Representation*

The product structure can be made by a tree and their component elements can be posted on the levels, on the level one being the product, on the level 2 being their direct components. Also, any compound element has its own structure that can be represented on the levels.

DEFINITION 3. A direct component element of a compound element is that component element that is directly relied on the compound element, without exist any compound elements between the direct component element and the compound element.

AXIOM 1. In product structure representation or a compound element representation the element (product or compound) cannot appear as a component element on a specified level in its own structure.

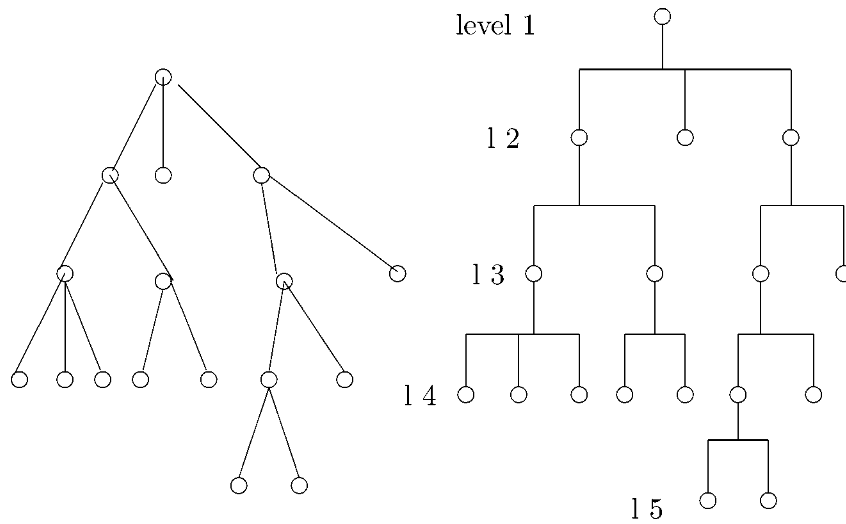

Fig. 4. Trees representation alternatives, with levels specification.

If this axiom is not satisfied, then in the product structure representation appear circuits. For that, it is recommended as the applications that use the product structure representation dispose for functions that allow to avoid the circuits. An distributed algorithm for avoiding the circuits was presented in (Cicortaş, 1997).

In Fig. 4, are given two alternatives for trees representation.

Data Bases

A compound element is represented as an oriented graph where are given all links ascendants (for assembly process) and also descendants (for necessary components determination). For achieving our purposes we propose the following particularly data bases that allow product structure representation:

- items – an entity is an element that can be a product, compound element or component element without any component element (the last is the listen);
- links – an entity defines the relation between the compound element ant one of its component element.

Every entity from a such data base, characterises uniquely the element respectively the relation.

The minimal attributes for the item are:

- the item name i;
- the lot size l_i;
- a flag for stop the decomposition sd_i;
- lot assembly time interval da_i.

The minimal attributes for the link are:

- the compound element (father) $F = i$;
- the component element (son) $S = j$;
- the quantity of component element that enters in the (lot size) compound element q_{ij};
- the gap between the begin of assembly interval and the instant when the component element j is necessary dl_{ij};
- the usage time interval in the assembly for the component element j, denoted as u_{ij}.

Starting from these proposed data bases can be achieve:

- compound element maintenance;
- determine the necessary for component elements in time.

On this basis, can be made the assembly modeling.

3.3. *Petri Nets*

DEFINITION 4. A Petri net is defined (Peterson, 1981) by the 5-tuple

$$RP = (P, T, I^-, I^+, M_0),$$

where

- P is a nonempty set of locations;
- T is a nonemptyset of transitions;
- $P \cup T = \emptyset$;
- $I^-, I^+ : P \times T \to N$, are the incidence functions between the input locations and transitions and transitions and output locations, respectively:
- $M_0 : P \to N$ is the initial marking.

Incidence matrix is given by $C = I^+ - I^-$.

DEFINITION 5. A transition $a_i \in T$ is enabled in the marking M, if $M \geqslant I^-(a_i)$.

DEFINITION 6. A transition $a_i \in T$ fires in the marking M if it is enabled and the firing result is a new marking, given by the relation

$$M' = M - I^-(a_i) + I^+(a_i).$$

Let a transitions sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ and a marking M_0. If a_1 is enabled then by firing of transition a_1, is obtained the marking M_1, if a_2 is enabled, in the marking M_1, by its firing is obtained the marking M_2, and so on, by firing of transition a_k, is obtained the marking M_k. So by firing of transitions sequence s: $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, we obtain the following markings sequence:

$$M_0 \stackrel{a_1}{\to} M_1 \stackrel{a_2}{\to} M_2 \ldots M_{k-1} \stackrel{a_k}{\to} M_k.$$

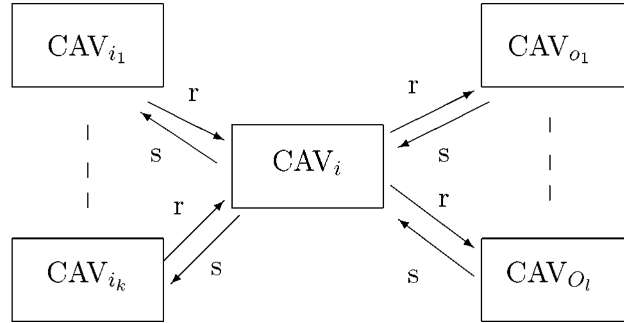

Fig. 5. Communication between the VACs.

By synthesising the result of transition sequence firing $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is given by the following expression

$$M_k = M_0 + C \cdot s.$$

The s is a vector where for every transition is counted its cardinality in the sequence s.

3.4. *Model Definition*

DEFINITION 7. The assembly system is modeled by the co-operation between the virtual assembly cells (VAC). A virtual assembly cell is a logical process from the previous section.

The system is represented in the Fig. 5, where the VAC_i is the current VAC, one of the n system cells. The links between the VACs are static and are defined at simulation begin.

REMARK 2. Instead of input and output buffers in our model will be used the appropriate lists, that will be described as soon as their necessity appear.

Particularities

1. Every VAC_i receives requirements $s_{o_r,i}$ from other VACs. A requirement is a *message* on the form $< cs_{o_r,i}, ms_{o_r,i} >$, where can be identified the message sender o_r, the message receiver i, the amount of component element i required, $cs_{or,i}$ and the instant when it is required $ms_{o_r,i}$. These requirements will be stored in a requirements list LS_i in the decreasing order of requirement instant $ms_{or,i}$.
2. When a VAC is activated (in our case the current VAC), and it has some non executed entries in the LS_i, for every requirement non executed, it execute a necessary estimation that consists from the following:
   - determines the amount for every component element that is necessary as $cn_{i,ij} = cs_{o_r,i} \times q_{ij}$, where $j = i_1, \ldots, i_k$,

   - determines the instant when these component elements are necessary as $m_{i_r} = m_{o_r,i} \times cs_{o_r,i} \times da_i$.

3. When necessary estimation was finished, then the VAC sends for every component element it, to the appropriate VAC. The message has the following form $< cn_{i,ij}, m_{i,ij} >$ (and for not give all details we omitted the exact localisation of requirement).
4. As soon as every cell $i_j \in \{i_1, \dots, i_k\}$ which has received the requirements, accomplish these and sends to (the current) requester cell i as messages which have the following contents $< cn_{i,ij}, md_{i,ij} >$, where $j \in \{i_1, \dots, i_k\}$. For simplify the description, suppose that was accomplished and received the same quantity which was required. In a refined version will allow to accomplish a lower amount from required quantity, as soon as is accomplished the rest from required amount, it follows to be sent. The partially amount received will be used in assembly process and will be assembled a lower amount from the compound element than was required.
5. When, for a requirement were received all needed component elements, then the current cell VAC_i, will pass in the active phase which consists in assembly required amount from its (own) compound element, $cs_{o_r,i}$. The instant when it will obtained *which represents the new LVT*, is given by

$$LVT_i = \max(LVT, md_{i,i_1}, \dots, md_{i,i_k}) + da_i \times cs_{o_r,i}/l_i. \quad (1)$$

   It must to set at the begin of simulation

$$LVT_i = 0, \qquad \forall i \in \{1, \dots, n\}.$$

6. After the previous phase termination , the current cell VAC_i will send the (own) achievement to the requester o_i by a message $< cs_{o_r,i}, LVT_i >$.

3.5. *Assembly Cell Activities*

The previous allow us to define the assembly cell functions, that can be clustered in:

- processing functions:
  - necessary component element estimation;
  - assembly process, active phase, which implies time evolution in the VAC;
- communication functions:
  - receive of requirements;
  - send the own requirements for component elements;
  - receive achievements (of needed component elements);
  - send own achievements.

It must observe that the processing functions are executed by the cell when their conditions are satisfied. These conditions are specific and consist from:

- for necessary component element estimation – the existence of a requirement from another VAC, for the compound element which is assembled by the current VAC_i;
- for assembly process – the existence of all needed component element in appropriate amounts (this condition is equivalent with the enabled condition for the appropriate transition of a Petri net), for assembling its (own) compound element.

DEFINITION 8. The processing functions: necessary component element estimation and assembly are named internal activities. They are (assembly) or not (necessary component element estimation), simulation (processing) time consuming.

REMARK 3. The assembly phase is an atomic activity. In a refined implementation it can be considered as non atomic activity.

In other models the activities have another particularities and then the model differs from our model. In our model the results of internal activities consist form:

- the component element that are necessary for satisfying an (external) requirement;
- the assembled (own) compound element,

which will be the object of the messages send to the other VACs.

Time Evolution in VAC

For time evolution some considerations will be made:

1. In the necessary component element estimation, an instant is defined when a component element it should be used , but this instant has an estimate character;
2. The assembly is the function that is simulation (processing) time consuming and it will modify the cell LVT. The LVT value is given by the relation (1);
3. Every message sent by the VAC_s and received by the VAC_r contains a time value that is:
   - estimate when the message is a requirement;
   - the LVT value when the message is an achievement.

If the internal activity (assembly process) is atomic, then the LVT is given by relation (1).

If the internal activity (assembly process) is not atomic some subactivities from it can be viewed, then we can obtain a refinement, applying a critical path method in which occur:

- the instant $md_{i,ij}$ when the component element is delivered as the message by the VAC_j;
- delay $dl_{i,j}$ from the start of assembly process in the VAC_i;
- the time interval of usage u_{ij} in assembly process of the component element j in VAC_i.

The critical path is built and are specified the elements that are constituents of this path. Another component elements that are not constituents of the critical path, can arrive (their

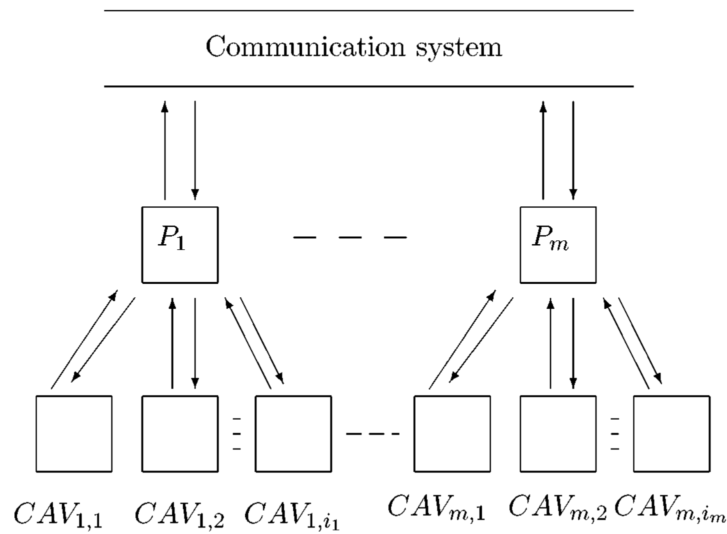

Fig. 6. Virtual assembly cell and processors model.

achievements), between the limits of type the earlier and the latest. If a such component element (which not is constituent of critical path) arrives out of these limits, then it will enter in the critical path and then it must be rebuild.

3.6. *Model Design*

From the previous, we can conclude that we have:

- internal activities;
- communications between the cells.

From these reasons we can design an adequate implementation. For modeling on a distributed architecture, suppose that we dispose for a number of m processors and we have n virtual assembly cells. Evidently, that $m \leqslant n$ or even $m << n$. So, for a processor we will ascribe a subset of virtual assembly cells, on which the processor will manage. The proposed model in represented in the Fig. 6.

A processor from the distributed architecture which manages the subset of ascribed VACs, will assure the achievement of processing functions and communication functions of these VACs. All the necessary elements will be defined in an adequate manner and using the object-oriented concepts.

DEFINITION 9. The system model is composed from the Virtual Assembly Cell and the Processor which manages the ascribed VACs. Logical Process of the model is the VAC.

DEFINITION 10. The Virtual Assembly Cell has the following:

- **attributes:**

- the cell identification (its name) VAC_i;
- the joined Petri net transition;
- the lot size l_i;
- the stop decomposition flag od_i;
- the assembly time interval da_i;

- **services (methods):**

  - create **cre** (VAC_i);
  - destroy **des** (VAC_i);
  - estimate the necessary for a requirement **est** (req_l);
  - assembly for a requirement **as** (req_l);

The VAC_i has the following **lists**:

- the requirements list LS_i, where every entry in the list req_l is a requirement made by another cell VAC_l for its own compound element;
- for every entry in the previous list req_l is created a new list with the component elements that are necessary for that requirement, LCN_{il}.

The entry from the requirement list req_l has the following attributes;

- the sender cell VAC_l;
- the message contents which has the form

  $$< cs_{l,i}, ms_{l,i}, crt(l,i) >,$$

  where

  - $cs_{l,i}$ is the amount of the component element i required;
  - $ms_{l,i}$ is the instant when the component element i is required;
  - $crt(l,i)$ is the number of the requirement made by VAC_l for the VAC_i.

An entry from the LNC_{il} has the following attributes:

- the component element required VAC_j;
- the message contents of the requirement which has the form

  $$< cn_{ij}, ms_{ij} >,$$

  where

  - $cn_{i,j}$ is the amount from the component element j that is necessary;
  - $ms_{i,j}$ is instant when the component element is required;

- the message contents of the receiving amount that was accomplished by the VAC_j, which has the from

  $$< ca_{i,j}, ma_{i,j} >,$$

where

- $ca_{i,j}$ is the accomplished amount and $ca_{i,j} = cn_{i,j}$;
- $ma_{i,j}$ is the LVT_j the instant when the component element is available for the VAC_i.

DEFINITION 11. The Petri net transition tr_i that is associated with the VAC_i has the specific attributes and services.

The attributes are defined for every component element S_j and these are:

- the compound element is implicit VAC_i;
- the component element is S_j;
- the amount of component element j which enters into lot size of i is q_{ij};
- the usage of component element j in assembly process is u_{ij};
- the delay of usage for component element i is dl_{ij}; it is counted relative to start of assembly process for i.

The basic service (method) of transition tr_i is the firing. A transition is enabled if exists a requirement req_l for that was not estimated the necessary. If a such requirement exists, then by calling **ca** (req_l), the transition fires and by firing are estimated:

- the amount of component element that is necessary for achieving the requirement

$$cn_{i,j} = cs_{l,i} \times q_{ij};$$

- the instant when it is required

$$m_{i,j} = ms_{l,i} + cs_{l,i} \times da_i/l_i.$$

The firing results will be loaded into the component elements list $LNC_{i,l}$ and from, will be sent as messages.

REMARK 4. In our concepts, the messages that allow the communication between the VACs, will be sent by the processors which manages these cells involved in the communication.

DEFINITION 12. The processor that manages the associated cells subset, execute mainly the following activities: after a specified policy activate a cell; this cell when is activated take the control, execute its activities (only which are executable at the activated instant) and after, give to the processor the control.

For manage the associated cells subset, the subset is constituted as a closed list, which the processor examines it. When the processor examines a cell it can be in the one of the following alternatives:

- For a requirement for which was estimated the necessary of component elements, the cell dispose for all these achievements (made by the respective component element cells). In a such case the processor gives to the cell the command which pass in the active phase, assembles the (own) compound element, evolves its own LVT and in the end gives to the processor the command. At the phase end, the appropriate lists will be updated.
- The cell has a requirement or more for which not was yet made the estimation for necessary component elements. In a such case the processor gives the command to the cell, which starts the necessary estimation for the respective requirement(s). At the end of it, the appropriated lists will be modified appropriately. After that, the command pass to the processor.
- While exist cells which is not in one of the previous alternatives and in the associated lists a cell has some messages to send, then the processor execute its appropriate task as follows:
  - in the LNC lists exists the result of necessary estimation for a requirement and these were not sent, then the processor send the messages to the appropriate VACs;
  - if a cell just finished its active phase, the assembly phase, and its result was not sent, then the processor send to the appropriate cell that made the requirement, the message with the accomplished requirement (and the LVT).
- The cell is not in one of the previous alternatives. Then the processor (give not it the command) and pass to the next cell in the list. The simulation ends when there are no any cell in a processing phase and all requirements for all cells were satisfied.

The VACs receive external messages from other VACs. The message contents was defined in the previous considerations. The message receiving is solved as follows:

- A cell receives a message which is a requirement (for assembling its own compound element). Then the processor (if there is not creates the requirement list and) inserts in the requirements list the arrived message, without giving the control to the cell.
- A cell receives a message which is an achievement of a component element made by the appropriate cell as a consequence of a requirement (made by the current cell). Then the processor operates the message insertion in the LNC list.

Starting from these, we can develop the attributes and services for the processor in the object-oriented concepts. So, the processor will have the following:

- **attributes**:
  - the processor id (name) pr_i;
  - the subset of associated VACs, as a closed list set_i;
  - the current VAC, VAC_c;
  - the queue of non processed messages received for the associated VACs. An item of the queue, has the following attributes:

  - $*$ the source VAC;
  - $*$ the destination VAC;
  - $*$ the message;

- **services (methods)**:
- create processor **crepr** (pr_i);
- destroy processor **destrpr** (pr_i);
- create VACs subset **creva** ();
- select next cell from list **senex** ()which becomes the current cell *crtVAC*;
- analyse the current cell. The current cell, *crtVAC* can be in the one of the following alternatives:
  - crtVAC not requires any processing phase (necessary estimation, or assembly) and also not requires message sending. In this case the processor selects the next cell from the (closed) list;
  - crtVAC reclaims a necessary estimation, when it receives the command;
  - crtVAC reclaims an assembly, when it receives the command;
  - crtVAC has executed an assembly and must send the results to the appropriate cell. Then, the processor sends the assembly result by a service **trmas**$(crtVAC, mes_r)$.
  - crtVAC has executed a necessary estimation and it must send the appropriate messages, which are taken by the processor and sent to the cells **trmes** $(crtVAC, mes_j)$.
- the queue messages processing consists from:
  - inserting of a (just arrived) message in the queue **insm** (source, destination, message contents);
  - processing the current message (from top of queue), by inserting them in the appropriate list requirements or achievement of a requirement made, and deleting it form the queue **prme** (list, message contents); where list represents LS_i or LCN_i, j respectively.

4. Conclusions

This paper describes the asynchronous parallel discrete event simulation (PDES) and its mechanisms by examples of virtual assembly cell system simulation. Were described only conservative mechanisms and some specific methods tailored to the VAC system. No performance studies have been conducted. Investigating the performance of these methods will be one of the future research directions.

The methods described in the paper are applied in many possible applications but they must be tailored to the specificity of these applications. The VAC system can be seen belonging of a larger class simulation where the space is discretized into a set of VACs and then the element component (of product structure) are moved from one cell to another.

Like all real applications it was developed for it, an user-friendly environment, but it can be developed together to the specific methods tailored to other applications.

References

Bagrodia, R., K.M. Chandy and Liao Wen-Toh (1991). A unifying framework for distributed simulation. *ACM Transactions on Modeling and Computer Simulation*, **1**(4), 348–387.

Cicortaş, A. (1995). Designing data bases for product structure representation using object-oriented concepts. *Annals of the West University of Timişoara*, Series Economic Sciences, **1**(7), 97–102.

Cicortaş, A. (1997). Distributed modeling of discrete events. *PhD. Thesis*. West University of Timişoara, Mathematics Faculty.

Coad, P., and E. Yourdon (1990). *Object-Oriented Analysis*. Prentice-Hall, N. Y.

Coad, P., and E. Yourdon (1991). *Object-Oriented Design*. Prentice-Hall, N. Y.

Coleman, D., *et al.* (1994). *Object-Oriented Development, The Fusion Method*. Prentice-Hall, N. Y.

Ferscha, A., and S.K. Tripathi (1996). *Parallel and Distributed Simulation of Discrete Event Systems*. Computer Science Department University of Maryland.

Fishwick, P.A., and B.P. Ziegler (1992). A multimodel methodology for qualitative model engineering. *ACM Transactios on Modeling and Computer Simulation*, **2**(1), 52–81.

Fishwick, P.A. (1993). A simulation enviroment for multimodeling. *Discrete Event Dynamic Systems, Theory and Applications*, **3**, 151–171.

Fujimoto, R.M. (1990). Parallel discrete event simulation. *Communications of the ACM*, **33**(10), 31–53.

Henderson-Sellers, B., and J.M. Edwards (1990). The object-oriented systems life cycle. *Comm. of the ACM*, **33**(9), 142–159.

Peterson, J.L. (1981). *Petri Net Theory and the Modeling of the Systems.* Prentice-Hall, N.Y.

Wirfs-Brock, R., B. Wilkerson and L. Wiener (1990). *Designing Object-Oriented Software*. Prentice-Hall, N. Y.

Yi-Bing, Lin, and P.A. Fishwick (1996). Asynchronous parallel discrete event simulation. *IEEE Transactions on Systems, Man and Cybernetics*, **XX**(Y), Part A.

A. Cicortaş obtained a MS in mathematics in 1966 from the Mathematics Faculty of “Babeş Bolyai” University of Cluj and a PhD in mathematics in 1997 from the Mathematics Faculty of West University of Timişoara. He is interested in: modeling and simulation (especially discrete event modeling), design and analysis in object-oriented concepts, modeling with Petri nets and their derivates (evaluation nets, stochastic Petri nets and coloured Petri nets used in modeling of manufacturing processes).

Konservatyvus diskretinių įvykių sistemos modeliavimas

Alexandru CICORTAŞ

Straipsnyje nagrinėjamos asinchroninių lygiagrečių diskretinių įvykių modeliavimo problemos. Pasiūlyta tokia modeliavimo metodologija. Jos mechanizmas iliustruojamas pramonės gaminių surinkimo iš smulkesnių elementų proceso aprašymo ir modeliavimo pavyzdžiu.