Asynchronous Parallel Discrete Event Simulation

Yi-Bing Lin and Paul A. Fishwick, senior member

Abstract— **Complex models may have model components distributed over a network and generally require significant execution times. The field of parallel and distributed simulation has grown over the past fifteen years to accommodate the need of simulating the complex models using a distributed versus sequential method. In particular, asynchronous parallel discrete event simulation (PDES) has been widely studied, and yet we envision greater acceptance of this methodology as more readers are exposed to PDES introductions that carefully integrate real-world applications. With this in mind, we present two key methodologies (*conservative* and *optimistic*) which have been adopted as solutions to PDES systems. We discuss PDES terminology and methodology under the umbrella of the personal communications services application.**

Keywords—**parallel algorithm, distributed simulation, synchronization, virtual time, network communications**

I. Introduction

Our purpose is to introduce the basic technical concepts of distributed simulation of event-based models (so called *discrete event models*), and to tie these generic concepts to a specific application: personal communications services (PCS). Several introductory articles have been presented in the literature such as Fujimoto [1], Nicol et al. [2] and Richter et al. [3]. These papers have helped to disseminate the *asynchronous parallel discrete event simulation* (PDES) methodology for a wide readership. Our approach is similar but stresses a single real world application for discussing the methodology of PDES. By defining the methodology and all PDES terminology within the context of the PCS application, this paper serves both as a tutorial to PDES and as an introduction to PCS simulation modeling. PCS is a rich enough application to illustrate most basic PDES concepts.

The processing elements in PDES can either be of a *parallel* or *distributed* nature. An MIMD machine with multiple asynchronous elements performing message passing is an example of a parallel machine. Distributed elements normally refers to local or wide area networks composed of inter-connected set of heterogeneous workstations and computers. PDES is used for one of two reasons: 1) one wants to execute a model faster than is possible in a sequential machine, or 2) one must model in a distributed fashion because of a constraint that a process (i.e., computation) must be distributed rather than localized to a single processor. One author (Lin) has demonstrated various speedups possible on a distributed memory architecture for the PCS application [4], [5]. There is no question that PDES speeds up otherwise serial computations during a simulation. The second reason for PDES (distributed model constraint) is based on a situation where models for system components are stored in physically different locations. The other author (Fishwick) is building a prototype distributed simulation of a process plant where each plant component is ultimately co-located with the manufacturer of that component.

The paper proceeds as follows. First, in Section II, we define our terms within the PDES area and demonstrate the generic approach to distributed simulation. In Section III, we introduce the PCS application and demonstrate the need for synchronization of incoming messages to a given process. There are two key approaches to synchronization. Method 1, defined in Section IV, is termed the *conservative method* since it ensures that the causal relation among time consecutive events will be maintained at all times during the simulation. Method 2 is defined in Section V, and identifies the *optimistic method.* In this approach, the causal relation can be broken with subsequent fixing of state variables. We close in Section VI with directions for the future of PDES.

Throughout this paper, we use three font styles to represent different concepts. The `typewriter` type style represents attributes or methods (e.g., `SendMessage()`) of objects. The *italic* type style represents variables such as *LP* or *p*. The serif type style represents event types such as CallArrival.

II. Parallel Discrete Event Simulation

A. Basic Terminology

We begin by defining terms which are commonly found in the simulation and PDES fields. These terms will be revisited in Section 3 when we assign the terms to the PCS application. The study of any physical system to be simulated begins with the creation of a *model.* Such a model can be in one of several types [6]: 1) conceptual, 2) declarative, 3) functional, 4) constraint, 5) spatial or 6) multimodel. One begins with a conceptual model which describes qualitative terms and class hierarchies for the system. In many ways, the conceptual model "organizes" the definition of attributes, methods and general characteristics of each system component without going so far as to ascribe dynamics to components. The next four model types reflect an orientation to system construction; a system may be constructed as a Petri net [7], queuing model [8] or as a cellular automaton [9] for instance. The last model type (multimodel) permits the integration of basic model types to create a model composed of component models [10], [11] where each component model represents a level of abstraction for the system.

Yi-Bing Lin is Professor with the Dept. of Computer Science and Information Engineering, National Chiao Tung University. E-mail: liny@csie.nctu.edu.tw.

Paul A. Fishwick is Associate Professor with the Dept. of Computer and Information Sciences, University of Florida. E-mail: fishwick@cis.ufl.edu .

The PCS area, to be discussed in Section 3, uses a *spatial model* in that the system is viewed as a hexagonal discretization of a large two-dimensional space representing an area where cellular communications are to be implemented. Spatial models can be executed in several ways including *time slicing*, *event scheduling* and *parallel and distributed*. Our approach will be to use a parallel and distributed approach to *model execution*, while using the concept of event scheduling within each process. Speaking of *process*, we must define this term appropriately. Model components for a PCS implementation will be a collection of hexagonal cells. Other model types, such as a queuing model, are composed of other components (facilities). A *logical process* (LP) is defined as a set containing basic model components, so a PCS logical process will be a set of hexagons, or just one hexagon. A *physical process* or *processor* is a set of *logical processes* mapped in a way that conforms to the architecture of the parallel/distributed system.

An LP contains several objects:

- *Local Virtual Time* (LVT): time associated with the LP. The LP does not know another LPs time unless communicated via a message.
- *Future Event List* (FEL): event list used when there are internal events posted within the LP itself.
- *Event*: an item within the FEL.
- *Message*: an item sent from one LP to another.

The FEL is composed of events, where an event combines the following objects: 1) time stamp, 2) token, 3) event type. The time stamp reflects when the event is to occur. An event's occurrence correlates with the execution of an *event routine* for that LP. The token is associated with whatever is flowing through the network of LPs. For the PCS application, *portables* (i.e., mobile phones) flow through the system. An event type specifies what will happen to the token (arrival, boundary crossing, departure, incoming call). An LP has input channels and output channels where each channel has a first-in/first-out (FIFO) buffer associated with it. A message is equivalent to an event that must be moved from one LP to another. Messages which simply enter an FEL and are processed are generally called events. When an event must be issued to another LP, it becomes a message. The relationship among the above terms are shown in Figure 1.

[Figure 1 about here.]

Messages arrive in one of several input channel buffers and are routed directly to the LPs FEL. Note that simple LPs may involve a calculation such as 1) taking the timestamp from an incoming message, 2) adding a value to this timestamp, and 3) sending the new message to the output buffers. Such an LP would not have any need of an FEL and would be a "pure" distributed simulation. This kind of technique, however, is wasteful of the computing elements since there will be a large price to pay in communications overhead among inter-LP communication. A simple addition is not sufficient to warrant a distributed approach. On the other hand, if the processing element can be *made to do work* then the communications overhead becomes less critical. The kind of work ideally suited in simulation is a sequential simulation within the LP, composed of the usual FEL and event routines. Thus, the distributed simulation is hybrid in form with sequential simulation coinciding — and synchronized with— distributed simulation. The LVT of this more substantial LP is updated by removing the highest priority event (lowest timestamp) from the FEL and executing the associated event routine. Some (or all) of these event routines will contain scheduling commands to place events with new times back into the FEL. Some event routines will involve messages to be issued through the output buffer(s) to a target LP.

B. Object Oriented Implementation

A PDES consists of several PDES objects or LPs. These LPs execute asynchronously with coordination to complete a simulation run. To implement the objects in an LP (as described in Section II-A), the attributes and methods of the LP are classified into four categories (see Table I):

[Table 1 about here.]

- A *clock* mechanism indicates the progress of the LP. An attribute `LVT` represents the timestamp of the event that just occurred in the LP. The `LVTUpdate()` method updates `LVT` to advance the "clock" of the LP.
- A *FEL* mechanism processes the events occurring in the LP. The FEL is basically a *priority queue* with one attribute and three methods. An attribute `eventList` maintains the events to occur in the future. The `Enqueue()` method inserts a time-tagged event into `eventList` so that `eventList` maintains its ordered sequence. The `Dequeue()` method deletes the event with the minimum timestamp in `eventList`. The `Cancel()` method deletes the event with a specified timestamp in `eventList`.
- A *synchronization* mechanism interacts with other LPs to coordinate the execution of PDES. The `ReceiveMessage()` method receives messages from other LPs (these messages will be inserted into the FEL for processing). The method `ExecuteMessage()` executes events in the FEL. The `SendMessage()` method sends output message (generated by the execution of events) to their destination LPs.
  It is probably more appropriate to consider `ExecuteMessage()` as a method of the FEL. However, this method is affected by the PDES synchronization mechanisms to be described later. Thus the method is classified as part of the synchronization mechanism.
- An *application* mechanism represents a sub-model for a specific simulation application to be simulated by the LP (to be elaborated).

C. PDES Implementation Platforms

PDES systems have been implemented in different parallel architectures such as BBN Butterfly [12], [13], [14], Sequent [15], [16], [17], JPL Mark III [18], Simulated Stanford Dash Multiprocessor [19], Transputers [20], [21], CM-

1/CM-5 [22], KSR [23], and iPSC/860 [24]. PDES has also been implemented in workstations connected by a local area network [4] which is widely available in both the industrial and the academic environments.

III. Personal Communication Services

We use *personal communication service* (PCS) network simulation to illustrate PDES functionality. A PCS network [25], [26] provides low-power and high-quality wireless access for PCS subscribers or *portables*. The service area of a PCS network is populated with a number of radio ports. Every radio port covers a sub-area or *cell*. The port is allocated a number of channels (time slots, frequencies, spreading codes or a combination of these). A portable occupies a channel for an incoming/outgoing call. If all channels are busy in the radio port, the call is blocked. In PCS network planning, PCS network modeling (usually conducted by simulation experiments) is required to investigate the usage of radio resources. Since PCS network simulation is time-consuming, PDES effectively speeds up the process of PCS network simulation. Specifically,

- The size of the PCS network under study is usually large (e.g., thousands of cells). A typical sequential PCS simulation run takes over 20 hours, while the corresponding PCS PDES takes less than 3 hours using 8 processors [4].
- Another popular parallel approach, the *parallel independent replicated simulation* [27], [28], [29] (running multiple simulation replications concurrently) does not work for PCS simulation. In most cases, the PCS designer only is interested in the behavior of the PCS network at the engineered workload (e.g., the workload at which the blocking probability is 1%). To calibrate the simulation at the engineered workload, the setup of input parameters for the next simulation run is dependent on the previous run.

Now we describe the PCS model and its mapping to the corresponding PDES. For demonstration purposes, we describe a simplified PCS model without considering the details of the radio signal propagation issues (such as Rayleigh fading, co-channel interference, and so on). We assume that there are S cells in the PCS network, and on the average, there are n portables in a cell. Every port is allocated some number of channels. A portable resides at a cell for a period of time which is a random variable with some distribution (e.g., exponential [30], [31], [32]). Then the portable moves to a neighbor cell based on some routing function (e.g., equal routing probabilities for all neighbors). The call arrivals to a portable is a random process (e.g., Poisson), and is independent of the portable's movement. A call is connected if a channel is available. Otherwise, the call is *blocked*. When a portable moves from one cell to another while a call is in progress, the call requires a new channel (in the new cell) to continue. This procedure of changing channels is called *handoff* or *automatic link transfer* (ALT). Several handoff schemes have been proposed in the literature [33], [34], [35]. In this paper, we consider the simplest scheme called *non-prioritized scheme*. In this scheme, if no channel is available in the new cell, then the call will be dropped or forced terminated immediately.

The PCS example is probably more realistic to the reader if we add some geometry to these moving vehicles (portables). Unfortunately, whether a vehicle moves from one cell to another cannot be simply determined by the physical movement of the vehicle. We also need to consider the radio propagation. It is possible that the connection to a vehicle changes from one port to another even if the vehicle is stationary – the change of radio signal strength may result in re-connecting the vehicle to a different port. According to the PCS network measurement methods, we determine that the movement (in the sense of port connection) of a vehicle is best characterized by the residence time[1] distribution and the destination cell routing probability. The reader may image that this movement model is equivalent to a simple path approach where a vehicle moves straight with an angle. The angle determines the destination cell and the residence time is the product of a constant speed and the diameter of the cell[2].

To map the PCS model into PDES, the cells in the PCS network are represented by cell objects derived from the PDES objects (i.e., LPs). These LPs are then mapped to processors for execution (see Figure 2). A cell LP has the following attributes and methods (i.e., the application mechanism of a general LP): A constant attribute `channelNo` represents the total number of channels in a radio port. An attribute `idleChannelNo` represents the number of idle radio channels. A `portableList` collects the information of all portables reside in the cell. There are five methods in the cell object: `CallArrival()`, `CallCompletion()`, `PortableMoveIn()`, `PortableMoveOut()`, and `Handoff()`. These methods will be elaborated later.

[Figure 2 about here.]

The portables in the PCS network are represented by the *portable objects*. A portable object consists of four attributes:

- The `busy` attribute indicates the status of the portable. If `busy=YES` then the portable is in a conversation.
- The `callArrivalTime` attribute represents the next call arrival time.
- The `callCompletionTime` attribute represents the completion time of the current phone call when `busy=YES`. If `busy=NO`, the `callCompletionTime` attribute is meaningless.
- The `portableMoveOutTime` attribute represents the time when the portable moves out of the current cell.

There are two categories of events in a PDES. An *internal* event is scheduled and executed at the same LP (the event represents the interaction between a cell and a portable within the cell in our PCS example), and an *external* event is scheduled by one LP and is executed by another LP.

[1] Residence time refers to the time that a portable *resides* within a cell.

[2] But note that our movement model is practical – it is used to approximate real radio systems, while the simple path approach cannot.

Thus, after its creation, an internal event is inserted in the FEL by using the `Enqueue()` method, and an external event is considered as a message, and is sent to the destination LP by using the `SendMessage()` method. In the PCS PDES, there are three internal event types and one external event type. The internal event types are described below.

- CallArrival: Either the port (the cell) or the portable initiates a call setup. A radio link is required to connect the port and the portable. If no radio link is available or the portable is already busy with another conversation, the call is dropped.
- CallCompletion: A phone call completes, and the radio link between the port and the portable is disconnected.
- PortableMoveOut: The portable moves out of a cell. If the portable is in a conversation, the radio link between the portable and the port is disconnected.

We treat the CallArrival event type as an internally generated event based on a probability distribution. This is just an abstraction of the actual situation where arrivals are sent from outside the LP to one of the LPs input channels. Therefore, a more detailed simulation would involve "electromagnetic messages" reflecting the true nature of incoming calls. The use of a probability function is an abstraction for this underlying process.

The external event type is described below.

- PortableMoveIn: A portable moves in a new cell. If the portable is in a conversation, then a new radio link between the cell (port) and the portable is required. If no radio channel is available, the call is forced terminated.

In PDES, the execution of a PortableMoveOut event at a logical process LP_A always results in the scheduling of a PortableMoveIn event for the destination logical process LP_B. This event type is *external* (to LP_B), and the scheduling of the event requires communication between LP_A and LP_B.

An event/message m is of the format

$$m = (timeStamp, \bar{p}, eventType)$$

where *eventType* represents the type of the event, *timeStamp* represents the (simulated) time when the event occurs, and $\bar{p}$ is the pointer which points to the corresponding portable p. The execution of the event message m at a cell object LP is described as follows. The LP.`ExecuteMessage()` method invokes different methods according to the event type of m (the Pascal-like "case" statement is used in the following definition).

```
ExecuteMessage(m) {
    LVTUpdate(m.timeStamp)
    /* I.e., LVT← m.timeStamp. */
    case (m.eventType) of
        CallArrival: invoke CallArrival(m.p);
        CallCompletion: invoke CallCompletion(m.p);
        PortableMoveIn: invoke PortableMoveIn(m.p);
        PortableMoveOut: invoke PortableMoveOut(m.p);
    end case
}
```

The methods invoked in `ExecuteMessage()` are described below. When the event type of m is CallArrival, the following action is taken.

```
CallArrival(p) {
    if p.busy=YES then
        /* A call is already in progress when the new */
        /* call arrives at LVT. In other words, a busy line */
        /* occurs and the new call arrival is ignored. */
        update the busy line statistic;
    else /* I.e., p.busy=NO. */
        if idleChannelNo = 0 then
            /* The call arrival is blocked. */
            update the blocking statistic;
        else /* I.e., idleChannelNo> 0. */
            idleChannelNo ← idleChannelNo -1;
            p.busy ← YES;
            generate the call holding time t, and
                p.callCompletionTime ← LVT + t;
        end if
    end if
    generate the next inter-call arrival time t' and compute
    the next call arrival time as
        p.callArrivalTime ← LVT + t'.
    invoke ScheduleNewEvent(p);
    /* Schedule a new event (to be described). */
}
```

Note that the busy line and call blocking statistics are output measures of the PCS simulation (not shown in our PDES example) [3].

When the event type of m is CallCompletion, the following action is taken.

```
CallCompletion(p) {
    /* Release occupied channel at call completion. */
    idleChannelNo←idleChannelNo+1;
    p.busy=NO;
    invoke ScheduleNewEvent(p);
    /* Schedule a new event (to be described). */
}
```

When the event type of m is PortableMoveIn, the following action is taken.

```
PortableMoveIn(p) {
    if p.busy=YES then /* A handoff occurs. */
        invoke Handoff(p); /* To be described. */
    end if
    generate the portable residence time t and compute the
    next move time p.portableMoveOutTime← LVT+t;
    invoke ScheduleNewEvent(p)
```

[3]Call blocking is a major performance measure of a PCS network. A PCS Network is usually engineered at 1% or 2% blocking probabilities.

```
    /* Schedule a new event (to be described). */
}
```

The method `Handoff()` is described below.

```
Handoff(p) {
    if idleChannelNo=0 then
        /* No channel is available to connect the */
        /* handoff call i.e., the handoff fails. */
        update the forced termination statistic
        (not shown in our PDES example);
        p.busy=NO;
    else /* The handoff succeeds. */
        idleChannelNo← idleChannelNo-1;
    end if
}
```

If the event type of m is PortableMoveOut, the following action is taken.

```
PortableMoveOut(p) {
    if p.busy=YES then
        idleChannelNo←idleChannelNo+1;
        /* When a communicating portable moves */
        /* to a new cell, it releases the occupied */
        /* channel of the old cell. */
    end if
    determine the destination cell (LP') to which the
    portable moves ;
    generate an output message
    m' = (LVT, p̄, PortableMoveIn);
    invoke SendMessage(m', LP');
    /* A PortableMoveIn event is scheduled for LP'. */
}
```

Note that the timestamp of m' is the same as that of m.

In our implementation, the execution of the event message m results in the scheduling of exactly one future event m'. When m is executed, one or more attributes of the corresponding portable are updated. Then the next event for the portable is determined based on the updated values of the attributes. If the event type of m is PortableMoveOut, then a PortableMoveIn event message with the same timestamp is scheduled for the destination LP as described in the definition of `PortableMoveOut()`. For the other three event types, the message $m' = (ts, \bar{p}, eventType')$ is determined by invoking `ScheduleNewEvent()`:

```
ScheduleNewEvent(p) {
    if p.busy=NO then
        /* The portable is idle at LVT. The next */
        /* event occuring to p is either a call arrival */
        /* or a cell crossing movement. */
        ts ← min(p.callArrivalTime,
            p.portableMoveOutTime);
        if ts = p.callArrivalTime then
            m'.eventType' ←CallArrival;
        else m'.eventType' ←PortableMoveOut;
        end if
    else /* I.e., p.busy=YES. The portable is busy at */
        /* LVT. The next event occurring to p is either a */
        /* call arrival, a call completion or a cell crossing */
        /* movement. */
        ts ← min(p.callArrivalTime,
            p.callCompletionTime,
            p.portableMoveOutTime);
        if ts = p.callArrivalTime then
            m'.eventType' ←CallArrival;
        else if ts = p.callCompletionTime then
            m'.eventType' ←CallCompletion;
        else m'.eventType' ←PortableMoveOut;
        end if
    end if
}
```

Consider the example illustrated in Figure 3. In this figure, a portable is represented by a car (although in many PCS systems, portables are carried by pedestrians). A call arrival is represented by a phone connected to the car. A call completion is represented by a cross (disconnection) on the phone line.

At time 0, portable p_1 is at cell A. At time 10, a phone call for p_1 occurs. The call completes at time 13, and the portable moves to cell B at time 16. At time 20, another phone call for p_1 arrives. At time 24, p_1 moves to cell C (and a handoff occurs).

[Figure 3 about here.]

In PDES, cells A, B, and C are simulated by logical process LP_A, LP_B, and LP_C respectively. At the beginning of the simulation, the attributes of p_1 are

$$\begin{aligned}&\texttt{busy} = \texttt{NO},\\&\texttt{callArrivalTime} = 10,\\&\texttt{callCompletionTime} = ?,\\&\texttt{portableMoveOutTime} = 16\end{aligned}$$

and an event

$$m_1 = (10, \bar{p}_1, \mathsf{CallArrival})$$

is scheduled and inserted in the FEL of LP_A. When the LVT of LP_A advances to 10, m_1 is executed by invoking LP_A.`CallArrival`(p_1). Suppose that an idle channel exists. The call is connected and the call holding time for the conversation is generated (which is 3, or the call completion time is 10+3=13). The next call arrival time is also generated (which is 20 in Figure 3). Thus the attributes of p_1 are modified as

$$\begin{aligned}&\texttt{busy} = \texttt{YES},\\&\texttt{callArrivalTime} = 20,\\&\texttt{callCompletionTime} = 13,\\&\texttt{portableMoveOutTime} = 16\end{aligned}$$

and a new event

$$m_2 = (13, \bar{p}_1, \mathsf{CallCompletion})$$

is scheduled and inserted in LP_A's FEL. When the LVT of LP_A advances to 13, m_2 is executed. The method LP_A.`CallCompletion(`p_1`)` is invoked and the attributes of p_1 are modified as

$$\begin{aligned}&\texttt{busy} = \texttt{NO},\\ &\texttt{callArrivalTime} = 20,\\ &\texttt{callCompletionTime} = ?,\\ &\texttt{portableMoveOutTime} = 16\end{aligned}$$

and a new event

$$m_3 = (16, \bar{p}_1, \textsf{PortableMoveOut})$$

is scheduled. At LVT 16, m_3 is executed. The method LP_A.`PortableMoveOut(`p_1`)` is invoked to determine the destination cell (which is B in Figure 3), and a message

$$m_4 = (16, \bar{p}_1, \textsf{PortableMoveIn})$$

is sent from LP_A to LP_B by invoking LP_A.`SendMessage`(m_4, LP_B). Note that the portable p_1 migrates to LP_B when m_4 is sent. (In GIT/Bellcore's PCS implementation [4], a message is part of a portable object, and sending a message automatically migrates the corresponding portable object.) When LP_B's LVT advances to 16, it executes m_4. The next portable move time is generated (which is 24). The attributes of p_1 are modified as

$$\begin{aligned}&\texttt{busy} = \texttt{NO},\\ &\texttt{callArrivalTime} = 20,\\ &\texttt{callCompletionTime} = ?,\\ &\texttt{portableMoveOutTime} = 24\end{aligned}$$

and a new event $m_5 = (20, \bar{p}_1, \textsf{CallArrival})$ is scheduled.

A PDES is correct if the following rule is satisfied.

Local Causality Constraint: Every LP processes events in nondecreasing timestamp order.

The major problem of PDES is that the logical processes are executed at different speeds. Consider the scenario in Figure 4 that portable p_1 moves from cell A to cell B at time 20 with an ongoing phone call (i.e., a handoff call), and portable p_2 moves from cell C to cell B at time 13 with an ongoing phone call (see Figure 4(a)).

[Figure 4 about here.]

Consider the PDES scenario in Figure 4(b). LP_A sends a PortableMoveIn event (message) m_1 (for p_1) with timestamp 20 to LP_B. Later LP_C sends m_2 (for p_2) with timestamp 13 to LP_B. If LP_B executes m_1 before m_2 arrives, then the modifications to LP_B.`idleChannelNo` is out of the timestamp order, and the local causality rule is violated. Thus the simulation result is not correct.

To solve this problem, the executions of the logical processes must be synchronized. The remainder of this paper describes two popular asynchronous synchronization mechanisms, the *conservative* and the *optimistic* methods.

IV. Conservative Method

The conservative simulation [36] is *conservative* in the sense that it does not execute an event before it ensures that the local causality rule is satisfied. The conservative simulation follows two rules: the *input waiting rule* and the *output waiting rule*. It also assumes that

- the messages are received in the order they are sent (the *FIFO communication property*), and
- the communication channels among LPs are fixed and never change during the simulation. In Figure 4(b), LP_A (LP_C) has one *output channel* directed to LP_B, and LP_B has two input channels (one from LP_A and one from LP_C).

A. Basic Synchronization Mechanism

In a conservative simulation, every logical process LP repeats the following two steps.

Step 1. LP waits to select an input message m from its input channels (extra data structures are required to implement input channels in a logical process) by invoking LP.`ReceiveMessage()`. This method is implemented based on the input waiting rule to be described. The method inserts m into LP's FEL.

Step 2. Let ts be the timestamp of m. LP.`ExecuteMessage()` is invoked to process all events in the FEL with timestamps no larger than ts in non-decreasing timestamp order. The execution may invoke LP.`SendMessage()` to send output messages. This method is implemented based on the output waiting rule to be described. If the termination condition is satisfied (e.g., LP.`LVT`>5000), then exit the loop. Otherwise go to Step 1.

The waiting rules are described as follows.

The input waiting rule. An LP does not process any input message until it has received at least one message from each of its input channels. The input message with the smallest timestamp is selected for processing. Figure 5 shows how the input message is selected for the PCS simulation.

[Figure 5 about here.]

Figure 5(a) illustrates a PCS system where 6 portables p_1, p_2, p_3, p_4, p_5, and p_6 move from cells B, C, D, E, F, and G to cell A at times 30, 10, 26, 4, 12, and 14, respectively. In the PDES model (see Figure 5(b)), the PortableMoveIn events of $p_1, ..., p_6$ are represented by the messages $m_1, ..., m_6$ sent to LP_A. By the input waiting rule, m_4 is the next message to be executed in LP_A.

Assume that all messages sent from one LP to another are in non-decreasing timestamp order (this property will be guaranteed by the output waiting rule to be described next), then the input waiting rule ensures that the timestamp of the selected message is no larger than any input messages to be processed in the future.

The output waiting rule. An LP does not send an output message to another LP until it ensures that no output messages with smaller timestamps will be scheduled (at LP) in the future. Assume that all input messages are handled in non-decreasing timestamp order (the property is guaranteed by the input waiting rule). The output waiting rule is satisfied if an LP only sends output messages with timestamps no larger than its current LVT value.

Consider the following PCS example. Portables p_1,p_2 and p_3 move into cell A at times 10, 20, and 30, and move out of the cell at times 29, 24, and 36, respectively (see Figure6(a)). This situation occurs since a portable, once inside cell A, may take a dramatically different from other portables. Some portables may stay in the same physical location for a period while other portables continue moving towards an adjacent cell to A.

[Figure 6 about here.]

In PDES, m_1,m_2, and m_3 are input messages representing the arrivals of p_1,p_2 and p_3, respectively (see Step 1 in Figure 6(b)). When m_1 is processed, a move event m_1' for p_1 is scheduled with timestamp 29 (see Step 2 in Figure 6(b)). In the conservative simulation, m_1' cannot be sent to the destination LP immediately, or the output waiting rule may be violated. In our PCS PDES implementation, the portable move is simulated by two types of events: a PortableMoveOut event and a PortableMoveIn event. In Figure 6(b), m_i' and m_i'' represent the PortableMoveOut event and PortableMoveIn event of portable p_i, respectively. When the event m_i' is scheduled, it is inserted in LP_A's FEL. When the LVT of LP_A advances to the portable "move time" (i.e., the timestamp of m_i'), m_i' is processed, which results in sending the PortableMoveIn event m_i'' (with the timestamp of m_i') to the destination. In Figure 6 (b), m_2'' and m_1'' are sent after Step (3) and before Step (4); i.e., when LP_A is sure that next input message to be handled has timestamp larger than m_1' and m_2'. Note that m_2'' is sent before m_1'' is.

Since the output waiting rule is guaranteed by using the two "move" event types, the conservative `SendMessage()` method simply sends the output message to the destination. Note that for other applications, a different conservative `SendMessage()` method may be required to implement the output waiting rule.

The correctness of the conservative simulation can be proved by induction on the interaction of the two waiting rules.

B. Deadlock and Deadlock Avoidance

The input waiting rule may result in deadlock (LPs are waiting for input messages from each other and cannot progress) even if the simulated system is deadlock free.

Consider a three-cell PCS network (see Figure 7(a)).

[Figure 7 about here.]

There is one portable in the network, and the portable moves in the path $A \to B \to C \to A$. At time 0, the portable is in cell A. The portable moves form cell A to cell B at time 8. In the conservative simulation, a PortableMoveOut event m_1 is scheduled in LP_A initially (see Figure 7(b)). By the input waiting rule, LP_A waits for an input message from LP_C before it can process m_1. Similarly, LP_C does not send out any output message before it receives an input message from LP_B, and LP_B does not send out any output message before it receives an input message from LP_A (i.e., before m_1 is processed). Thus the PDES is in the deadlock situation. Two deadlock resolutions have been proposed: *deadlock avoidance* [36] and *deadlock detection/recovery* [37], [38]. It has been shown [39] that the cost of deadlock detection/recovery is much higher than deadlock avoidance. This article will focus on the deadlock avoidance mechanism.

In a PCS network, a portable is expected to reside in a cell for a period of time before it moves. Assume that every portable resides in a cell for at least six time units before it moves to a new cell. The information that "a portable resides in a cell for at least 6 time units" is used in the deadlock avoidance mechanism to predict when an LP will receive an input message, and "6 time units" is referred as the *lookahead* value. The lookahead information is carried by the control messages called *null messages.* A null message does not represent any event in the simulated system. Instead, it is used to break deadlock as well as to improve the progress of a conservative simulation.

In Figure 7(b) , at the beginning of PDES, the LVTs of the three LPs are 0, and a PortableMoveOut event m_1 with timestamp 8 is in LP_A's FEL. At time 0, LP_A sends a null message with timestamp 0+6=6 (the LVT value plus the lookahead value) to LP_B (see Figure 7(c)). The null message implies that no portable will move in cell B earlier than time 6. Thus, the LVT of LP_B advances to 6 when the null message arrives (Figure 7 (d)). Since no portable arrives at cell B before time 6, it implies that no portable will move out of cell B before time 12 and LP_B sends a null message with timestamp 12 to LP_C. After the sending of several null messages, LP_A will eventually receive a null message with timestamp larger than 8 (see Figure 7 (e)), and by the input and output waiting rules, m_1 is sent from LP_A to LP_B and the deadlock is avoided (see Figure 7 (f)).

C. Exploiting Lookahead

It is important to exploit the lookahead to improve the progress of a conservative simulation. Experimental studies have indicated that the larger the lookahead values, the better the performance of the conservative simulation [39]. Based on the techniques proposed in [40], [41], [42], we give three PCS examples for lookahead exploration. The first two examples assume single cell entrance and exit. The single entrance/exit PCS model has been used in modeling highway cellular phone systems [43]. The results can be easily generalized for multiple entrances and exits. The techniques introduced can be combined to exploit greater lookahead.

1. *Lookahead Method 1 (FIFO):* In a large scale PCS network, a cell may only cover a street, and the portables leave the cell in the order they move in (the FIFO property; see Figure 8(a)).

   [Figure 8 about here.]

   Consider the corresponding FIFO LP for cell A in PDES. The lookahead for the LP can be derived by a presampling technique proposed by Nicol [41]. The idea is to presample the residence times of the arrival portables.

If the FEL is not empty, then the next departure time can be easily computed. In the PCS PDES, the move-out timestamp of a portable is computed and stored in `portableMoveOutTime` of the portable object at the time when the PortableMoveIn event is processed. The FIFO property guarantees that the next departure time is the minimum of the `portableMoveOutTime` values of portable objects in the FEL. Thus, the pre-computed next departure times can be used as the lookahead.

If the FEL of the LP is empty at timestamp LP.`LVT`, then the lookahead can be generated by the same pre-sampling technique. Since the portable will arrive at the cell later than LP.`LVT`, it will leave the cell later than LP.`LVT` $+ t$ (where t is the presampled portable residence time). The FIFO property guarantees that after time LP.`LVT`, no portables will depart earlier than LP.`LVT`$+t$, and the LP may send null messages with this timestamp to the downstream LPs.

2. *Lookahead Method 2 (Minimum Inter-Boundary Crossing Time):* Consider the example in Figure 8(b) where the FIFO portable movement property in the previous example does not hold. In practice, the inter arrival times to the cell (for the portables from the same entrance) cannot be arbitrary small. Instead, a minimum cell crossing time τ is assumed. Let p_i ($i = 1, 2, 3, ...$) be the ith portable arrival after time LP.`LVT`. The portable residence time for p_i is t_i. Then the departure time of p_i is later than LP.`LVT` $+ (i-1)\tau + t_i$, and the next departure time at the cell after LP.`LVT` is later than LP.`LVT` $+ \Delta$ where

$$\Delta = \min_{1 \le i < \infty} \left[(i-1)\tau + t_i \right] \qquad (1)$$

Since $\tau > 0$, there exists j such that

$$j\tau \ge \min_{1 \le i \le j} \left[(i-1)\tau + t_i \right] = \Delta$$

In other words, to compute Δ it suffices to consider the first j presampled residence timestamps in (1). Figure 9 displays a situation where we employ formula (1).

[Figure 9 about here.]

Four portables arrive using times 10, 14, 19 and 22. Let $\tau = 3$ so that we know that no two consecutive portable arrivals will be less than 3. The residence times for the portables are placed in parentheses in Figure 9. The variable j is increased by 1 until the above inequality is satisfied. Suppose that LP_A needs to send a null message to its downstream before it receives the PortableMoveIn event for p_1. The residence times of the subsequent arriving portables are pre-samples as $t_1 = 9, t_2 = 4, t_3 = 1, t_4 = 5...$ Our algorithm proceeds as follows:

(a) For $j = 1$, $1 \times 3 \ge \min[0 + 9] = 9$? No.
(b) For $j = 2$, $2 \times 3 \ge \min[0 + 9, 3 + 4] = 7$? No.
(c) For $j = 3$, $3 \times 3 \ge \min[0 + 9, 3 + 4, 6 + 1] = 7$? Yes.

From this procedure, we derive $\Delta = 7$ by using the first three pre-sampled residence times.

3. *Lookahead Method 3 (Minimum Residence Time):* If the FIFO portable movement property does not preserve, and τ does not exist (or is too small to be useful), then the technique proposed in the previous example may not work. In a PCS simulation, the total number $N = S \times n$ of portables is an input parameter. To compute the next lookahead value for an LP, it suffices to sample the next N portable residence times, and (1) is re-written as [42]

$$\Delta = \min_{1 \le i \le N} t_i$$

The last two examples may require a large number of operations to generate a lookahead value. In [40], $O(1)$ algorithms have been proposed to generate the lookahead values.

When the `ExecuteMessage()` method processes a null message in an LP, it invokes a method `ComputeLookahead()` to compute the timestamp of the output (null) messages. The `ComputeLookahead()` method may implement the lookahead exploiting techniques described above. Then the new null message is sent to some or all output channels by invoking the `SendMessage()` method.

V. Optimistic Method

The optimistic simulation [44] is *optimistic* in the sense that it handles the arrival events aggressively. When a message m arrives at an LP, LP.`ReceiveMessage()` simply inserts m in the *input queue* (the optimistic simulation terminology for the FEL). The logical process assumes that the events already in its input queue are the "true" next events. The `ExecuteMessage()` method proceeds to execute these events in timestamp order, and `SendMessage()` is invoked whenever an output message is scheduled . When a message arrives at the LP, the timestamp of the message may be less than some of the events already executed. (This arrived message is referred to as a *straggler*.) The optimism was unjustified, and therefore a method `Rollback()` is invoked by `ExecuteMessage()` to cancel the erroneous computation. To support rollback, data structures such as the *state queue* and the *output queue* are required (to be elaborated).

Several strategies for cancelling incorrect computation were surveyed by Fujimoto [45]. Two popular cancellation strategies called *aggressive cancellation* [44] and *lazy cancellation* [46] are described in this section.

A. Cancellation Strategies

Consider the example in Figure 10.

[Figure 10 about here.]

For simplicity, assume that cell C has one radio channel (i.e., LP_C.`channelNo`=1 in PDES). In this example, portable p_2 moves from cell B to cell C at time 10 (event 1), and make a phone call at time 13. The call is completed at time 21. Portable 1 moves from cell A to cell C at time 16 (event 2), and attempts to make a phone call at time 20. Since the only radio channel is used by portable 2, the call attempt from portable 1 is blocked. Portable 1 moves from cell C to cell D at time 24. Figures 11, 12, and 13

[Figure 11 about here.]

illustrate the data structures of LP_C (the logical process corresponding to cell C) assuming that message m_1 (the message that represents event 2) arrives at LP_C earlier than message m_5 (the message that represents event 1) does. In LP_C, a state queue and an output queue are maintained to supported rollback. In our example, the state variable (attribute) for LP_C is the number of idle channels LP_C.`idleChannelNo`. The state variable is checkpointed and saved in the state queue from time to time. The snapshots in the state queue are used to recover the state of LP_C when rollbacks occur. The output queue records the *anti-messages* of the output messages that have been sent from LP_C. The anti-messages are used to annihilated *false* messages sent in the incorrect computation.

In Figure 11(a), LP_C receives m_1 that is inserted in LP_C's input queue. Initially, the output queue of LP_C is empty, and the value of LP_C.`idleChannelNo` at timestamp 0 is saved. After m_1 is executed, the system state at timestamp 16 is checkpointed, and a call arrival event (message m_2) is scheduled for LP_C itself (see Figure 11(b)). Note that after its execution, m_1 is kept in the input queue (this message may be re-executed if a rollback occurs). A pointer in the input queue indicates the next event to be executed. The anti-message m_2^- of m_2 is saved in LP_C's output queue. The message m_2^- is identical to m_2 except that it includes a destination field (in the original optimistic or Time Warp algorithm [44], the sender and the destination are recorded in both the output message and the corresponding anti-message for flow control). To summarize, the `ExecuteMessage()` method for the optimistic simulation saves the system state after an event execution (note that the state may be saved after several event executions), and the executed event is not deleted from the input queue. The `SendMessage()` method saves the anti-messages in the output queue when it sends an output message.

After m_2 is executed, the number of idle channel is decremented by 1, and

$$LP_C.\texttt{idleChannelNo} = 0$$

is saved in the state queue. A PortableMoveOut event m_3 is scheduled at timestamp 24, and its anti-message m_3^- is stored in the output queue (see Figure 12(a)).

[Figure 12 about here.]

When m_3 is executed, a PortableMoveIn message m_4 is sent to LP_D (see Figure 12 (b)). After m_4 is sent, the straggler m_5 (the event that p_2 moves in LP_C at timestamp 10) arrives. Since LP_C.`LVT`=24, the out-of-order execution is detected (see Figure 13 (a)) by LP_C.`ReceiveMessage()`, and LP_C.`Rollback()` is invoked. Two strategies for cancelling incorrect computation are described below.

Aggressive Cancellation. When a straggler arrives, aggressive cancellation assumes that the out-of-order computation, as well as all other computations that may have been affected by this computation are not correct. Thus, the out-of-order computation is recomputed, and LP_C.`Rollback()` cancels the affected computations immediately by sending anti-messages. In our example, a rollback of LP_C at timestamp 10 occurs. In Figure 13 (b), the anti-messages m_2^-, m_3^-, and m_4^- are

[Figure 13 about here.]

deleted from the output queue, and are sent to their destinations to annihilate false messages m_2, m_3, and m_4, respectively. After the rollback (see Figure 13 (c)), messages m_2 and m_3 (and m_4 in LP_D) are removed from the input queue. The state of LP_C at timestamp 0 is re-stored. Then LP_C.`ExecuteMessage()` resumes the simulation by executing m_5.

Lazy Cancellation. It is possible that the erroneous computation still generated correct output messages. In that case, it is not necessary to cancel the original message that was sent. In lazy cancellation, logical processes do not immediately send the anti-messages for any rolled back computation. Instead, they wait to see if the reexecution of the computation causes any of the *same* messages to be regenerated. If the same message is recreated, there is no need to cancel the original. Otherwise, an anti-message is sent. In our example, lazy cancellation applies to three situations.

1. If portable p_2 arrives at cell C (LP_C) at time 10 and leaves cell C at time 28 without making any phone call (see Figure 14(a)) then the arrival of m_5 in Figure 13 (a) will not affect the executions of m_1, m_2, and m_3. (Note that in PDES, whether a call for p_2 occurs in the interval [10,28] can be detected in the portable object.) Thus messages m_1, m_2, and m_3 do not need to be reexecuted after m_5 is executed. This is called *jump forward* or *lazy reevaluation* [1].

   [Figure 14 about here.]

   In this case, LP_C.`ReceiveMessage()` simply inserts m_5 in the input queue, and the pointer of the input queue points to m_5. LP_C.`ExecuteMessage()` executes m_5 and the pointer jumps directly after m_3 without re-executing m_1, m_2, and m_3.
2. The call for p_2 does not block the call for p_1 if p_2's call completes before p_1's call arrives (see Figure 14 (b)) or p_2's call overlaps p_1's call but LP_C has two or more radio channels (i.e., LP.`channelNo` ≥ 2; see Figure 14(c)). In these cases, the channel utilization (not shown as a state variable in our example) changes, but the subsequent messages (i.e., m_2, m_3, and m_4) scheduled due to the execution of m_1 are not affected. Thus, messages m_1, m_2, and m_3 are re-executed to reflected the correct channel utilization. No anti-messages need to be sent (i.e., m_2^-, m_3^-, and m_4^- are not sent out). Like the previ-

ous case, LP_C.`ReceiveMessage()` simply inserts m_5 in the input queue. After m_5 has been executed, LP_C.`ExecuteMessage()` will re-execute m_1, m_2, and m_3 without re-generating any output messages.

If lazy cancellation does succeed most of the time, then the performance of the optimistic simulation is improved by eliminating the cost of cancelling the computation which would have to be reexecuted. If lazy cancellation fails, then the performance degrades, because erroneous computations are not cancelled as early as possible. In our PCS simulation, we may exploit situations that lazy cancellation does not fail (as described above), and a logical process can be switched between aggressive cancellation and lazy cancellation to reduce the rollback cost.

B. Memory Management

To support rollback, it is necessary to save the "history" (the already executed elements in the input, the output, and the state queues) of a logical process. However, it may not be practical to save the whole history of a logical process because memory is likely to be exhausted before the simulation completes. Thus, it is important that we only save "recent history" of logical processes to reduce the memory usage.

Memory management for the optimistic simulation is based on the concept of *global virtual time* (GVT). The GVT at (execution) time t is the minimum of the timestamps of the not-yet executed messages (these messages are either in the input queue or are in transit) in the optimistic simulation at time t. (Several other operational definition of GVT are given in [47], [48].) It has been pointed out [44] that at any given time t, a logical process cannot be rolled back to a timestamp earlier than the GVT at t. Therefore the storage for all messages with timestamps smaller than the GVT value can be reclaimed for other usage. The process of reclaiming the storage for the obsolete elements is called *fossil collection*.

The GVT computation is not trivial in a distributed system because it may be difficult to capture the messages in transit. Several GVT algorithms have been developed in the systems with the FIFO communication property [49] or without the FIFO communication property [50], [51].

In GIT/Bellcore PCS PDES (where eight workstations are connected by a local area network), all logical processes are frozen during GVT computation. By utilizing the low level communication mechanism, all transient messages are guaranteed to arrive at their destinations before the GVT computation starts. The fossil collection procedure works as follows. A coordinator initiates the procedure by freezing the execution of every logical process. After all transient messages arrive at their destinations, every logical process reports its *local minimum value* (the minimum of the timestamps of all unprocessed messages in the input queue) to the coordinator. The coordinator then compute the GVT value as the minimum of the received local minimums. The GVT value is broadcast to all logical processes for fossil collection.

To illustrate the storage reclaimed in fossil collection, consider the example in Figure 15. In this example, we ignore the phone call events and assume that all PortableMoveIn/PortableMoveOut events must be executed in their timestamp order in the optimistic simulation. We further assume that the state variable of a logical process is the number of portables move in the corresponding cell after time 0.

[Figure 15 about here.]

Portable 1 moves from cell C to cell A at time 4 and moves from cell A to cell B at time 60. Portable 2 moves from cell C to cell B at time 10. Portable 3 moves from cell B to cell A at time 20. Portable 4 moves from cell A to cell C at time 8. Portable 5 moves from cell A to cell C at time 7. Portable 6 moves from cell A to cell B at time 1 and moves from cell B to cell C at time 15.

Figure 16 illustrates the elements in the input/output/state queues of LP_A, LP_B, and LP_C after all transient messages arrive at their destinations, and the GVT value (which is $8 = \min(60, 20, 8)$) is found.

[Figure 16 about here.]

Figure 17 illustrates the elements in the input/output/state queues of LP_A, LP_B, and LP_C after the fossil collection procedure is completed.

[Figure 17 about here.]

All messages with timestamps smaller than 8 were fossil collected. Note that fossil collection for the state queue is not exact the same as that for the input/output queues. In the state queue, the element with the largest timestamp smaller than the GVT value (i.e., 8) must not be removed (see Figure 17). The other elements with timestamps smaller than 8 are removed.

C. Performance Evaluation

The performance of an optimistic PCS PDES implementation has been investigated in [4]. In this study, a version of Time Warp has been developed that executes on 8 DEC 5000 workstations connected by an Ethernet.

In the experiments, *speedp* was used as the output measure where the sequential simulator used the same priority queue mechanism as that of PDES for managing the pending set of events, but did not have the state saving, rollback and fossil collection overheads associated with the PDES implementation. 1024 cells are simulated for 2.5×10^5 simulated seconds. Figure 18 shows the performance of the optimistic PDES.

[Figure 18 about here.]

The figure indicates good performance of PDES for the PCS application. PDES is particularly efficient when the number of portables is large, the cell residence time is long, and the call interarrival time is short.

VI. Future Directions for PDES

This tutorial describes the asynchronous parallel discrete event simulation (PDES) mechanisms and optimization techniques by examples of personal communications services (PCS) network simulation. We described the conservative and the optimistic PDES mechanisms and several optimizations tailored for the PCS simulation. The performance of the optimistic method was briefly discussed. Since the conservative optimizations (tailored for PCS) introduced in this paper are new and were not previously reported, no performance studies have been conducted. Investigating the performance of these optimizations will be one of our future research directions.

The optimization techniques described in the paper are general and apply to other simulation applications such as battlefield simulation, VLSI simulation, queueing network simulation and computer architecture simulation. However, these optimization techniques may need to be tailored for specific applications. Many studies have devoted to this issue (see [1], [2], [52], [53], [54] and references therein). The PCS example can be seen as being a member of a larger class of simulation model where one first discretizes the spatial domain into a grid, and then simulates moving entities from one grid cell to another. In this sense, the PCS problem is isomorphic to the problems of particle/n-body simulation.

An important research direction that has not been fully exploited is the building of user-friendly PDES environments. Such an environment should provides convenient tools to develop simulation application. Methods should also be provided to tailor general optimization techniques to fit a specific simulation application. We anticipate that these user-friendly environments can be constructed by the object-oriented models described in [6].

Acknowledgments

C. Carothers and Y.C. Wong provided useful comments to improve the quality of this paper.

References

[1] Fujimoto, R.M., "Parallel Discrete Event Simulation", *Communications of the ACM*, vol. 33, no. 10, pp. 31–53, October 1990.

[2] Nicol, D. M. and Fujimoto, R. M., "Parallel simulation today", *Annals of Operations Research*, vol. 53, pp. 249–286, December 1994.

[3] Richter, R. and Walrand, J.C., "Distributed Simulation of Discrete Event Systems", *Proceedings of the IEEE*, vol. 77, no. 1, pp. 99–113, January 1989.

[4] Carothers, C., Fujimoto, R.M., Lin, Y.-B. and England, P., "Distributed Simulation of PCS Networks Using Time Warp", *Proc. International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 2–7, 1994.

[5] Carothers, C., Lin, Y.-B. and Fujimoto, R.M., "A Re-dial Model for Personal Communications Services Network", To appear in *45th Vehicular Technology Conference*, 1995.

[6] Fishwick, P.A., *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall, 1995.

[7] Peterson, J.L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[8] Law, Averill M. and Kelton, David W., *Simulation Modeling & Analysis*, McGraw Hill, 1991, Second edition.

[9] Toffoli, T. and Margolus, N., *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, 2nd edition, 1987.

[10] Fishwick, P.A. and Zeigler, B. P., "A Multimodel Methodology for Qualitative Model Engineering", *ACM Transactions on Modeling and Computer Simulation*, vol. 2, no. 1, pp. 52–81, 1992.

[11] Fishwick, P.A., "A simulation environment for multimodeling", *Discrete Event Dynamic Systems: Theory and Applications*, vol. 3, pp. 151–171, 1993.

[12] Ebling, M., Di Loreto, M., Presley, M., Wieland, F. and Jefferson, D., "An Ant Foraging Model Implemented on the Time Warp Operating System", *Proc. 1991 SCS Multiconference on Distributed Simulation*, pp. 21–26, March 1991.

[13] Hontalas, P., Beckman, B., Diloreto, M., Blume, L., Reiher, P., Sturdevant, K., Warren, L., Wedel, J., Wieland, F. and Jefferson, D., "Performance of the Colliding Pucks Simulation on the Time Warp Operating Systems (Part 1: Asynchronous Behavior & Sectoring)", *Proc. 1989 SCS Multiconference on Distributed Simulation*, pp. 3–7, March 1989.

[14] Fujimoto, R.M., "Time Warp on a Shared Memory Multiprocessor", *Proc. 1989 International Conference on Parallel Processing*, vol. Volume III, pp. 242–249, August 1989.

[15] Ayani, R. and Rajaei, H., "Parallel simulation of a generalized cube multistage interconnection network", *Proc. 1990 SCS Multiconference on Distributed Simulation*, pp. 60–63, January 1990.

[16] Thomas, G.S. and Zahorjan, J., "Parallel simulation of performance Petri Net: Extending the domain of parallel simulation", *Proc. 1991 Winter Simulation Conference*, pp. 564–573, 1991.

[17] Reed, D.A. and Malony, A., "Parallel Discrete Event Simulation: The Chandy-Misra Approach", *Proc. 1988 SCS Multiconference on Distributed Simulation*, pp. 8–13, February 1988.

[18] Wieland, F., Hawley, L., Feinberg, A., Di Loreto, M., Blume, L., Reiher, P., Beckman, B., Hontalas, P., Bellenot, S. and Jefferson, D., "Distributed Combat Simulation and Time Warp: The Model and Its Performance", *Proc. 1989 SCS Multiconference on Distributed Simulation*, pp. 14–20, March 1989.

[19] Soule, L. and Gupta, A., "An Evaluation of the Chandy-Misra-Bryant Algorithm for Digital Logic Simulation", *ACM Transactions on Modeling and Computer Simulation*, vol. 1, no. 4, pp. 308–347, 1991.

[20] Beazner, D., Lomow, G. and Unger, B., "A parallel simulation environment based on Time Warp", To appear in *International Journal in Computer Simulation*, 1995.

[21] Turner, S. and Xu, M., "Performance evaluation of the bounded Time Warp algorithm", *The 6th Workshop on Parallel and Distributed Simulation*, 1992.

[22] Lubachevsky, B., "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks", *Communications of the ACM*, vol. 21, no. 2, March 1989.

[23] Ghosh, K., Panesar, K., Fujimoto, R.M. and Schwan, K., "PORTS: A parallel, optimistic, real-time simulator", *Proc. 8th Workshop on Parallel and Distributed Simulation*, 1994.

[24] Gaujal, G., Greenberg, A.G. and Nicol, D.M., "A sweep algorithm for massively parallel simulation of circuit-switched networks", *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 484–500, 1993.

[25] Cox, D.C., "Personal communications – A viewpoint", *IEEE Commun. Mag.*, vol. 128, no. 11, pp. 8–20, 1990.

[26] Cox, D.C., "A radio system proposal for widespread low-power tetherless communications", *IEEE Trans. Commun.*, vol. 39, no. 2, pp. 324–335, February 1991.

[27] Glynn, P.W. and Heidelberger, P., "Analysis of Initial Transient Deletion for Parallel Steady-State Simulation", *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 4, pp. 904–922, 1992.

[28] Heidelberger, P., "Discrete Event Simulations and Parallel Processing: Statistical Properties", *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 6, pp. 1114–1132, November 1988.

[29] Lin, Y.-B., "Parallel Independent Replicated Simulation on A Network of Workstations", To appear in *SIMULATION*, 1995.

[30] Wong, W.C., "Packet reservation multiple access in a metropolitan microcellular radio environment", *IEEE J. Select. Areas Commun.*, vol. 11, no. 6, pp. 918–925, 1993.

[31] Wong, W.C., "Dynamic allocation of packet reservation multiple access carriers", *IEEE Trans. Veh. Technol.*, vol. 42, no. 4, 1993.

[32] Lin, Y.-B., "Determining the user locations for personal communications networks", *IEEE Trans. Veh. Technol.*, vol. 43, no. 3, pp. 466–473, 1994.

[33] Lin, Y.-B., Mohan, S. and Noerpel, A., "Channel Assignment Strategies for Hand-off and Initial Access fora PCS Network", *IEEE Personal Communications Magazine*, vol. 1, no. 3, pp. 47–56, 1994.

[34] Lin, Y.-B., Mohan, S. and Noerpel, A., "Queueing Priority Channel Assignment Strategies for Handoff and Initial Access for a PCS Network", *IEEE Trans. Veh. Technol.*, vol. 43, no. 3, pp. 704–712, 1994.

[35] Lin, Y.-B., Noerpel, A. and Harasty, D., "Sub-rating Channel Assignment Strategy for Hand-offs", *To appear in IEEE Trans. Veh. Technol.*, 1995.

[36] Chandy, K.M. and Misra, J., "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", *IEEE Trans. on Software Engineering*, vol. SE-5, no. 5, pp. 440–452, September 1979.

[37] Chandy, K.M. and Misra, J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", *Communications of the ACM*, vol. 24, no. 11, pp. 198–206, April 1981.

[38] Misra, J., "Distributed Discrete-Event Simulation", *Computing Surveys*, vol. 18, no. 1, pp. 39–65, March 1986.

[39] Fujimoto, R.M., "Performance Measurements of Distributed Simulation Strategies", *Proc. 1988 SCS Multiconference on Distributed Simulation*, pp. 14–20, February 1988.

[40] Lin, Y.-B. and Lazowska, E.D., "Exploiting Lookahead in Parallel Simulation", *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 4, pp. 457–469, October 1990.

[41] Nicol, D.M., "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks", *Proc. ACM SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages and Systems*, pp. 124–137, 1988.

[42] Wagner, D.B. and Lazowska, E.D., "Parallel Simulation of Queueing Networks: Limitations and Potentials", *Proc. 1989 ACM SIGMETRICS and Performance '89 Conference*, pp. 146–155, 1989.

[43] Kuek, S.S. and Wong, W.C., "Ordered Dynamic Channel Assignment Scheme with Reassignment in Highway Microcells", *IEEE Trans. Veh. Technol.*, vol. 41, no. 3, pp. 271–277, 1992.

[44] Jefferson, D., "Virtual Time", *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.

[45] Fujimoto, R.M., "Optimistic Approaches to Parallel Discrete Event Simulation", *Transactions of the Society for Computer Simulation*, vol. 7, no. 2, pp. 153–191, June 1990.

[46] Gafni, A., "Rollback Mechanisms for Optimistic Distributed Simulation", *Proc. 1988 SCS Multiconference on Distributed Simulation*, pp. 61–67, February 1988.

[47] Jefferson, D., "Virtual Time II: The Cancelback Protocol for Storage Management in Time Warp", *Proc. 9th Annual ACM Symposium on Principles of Distributed Computing*, pp. 75–90, August 1990.

[48] Lin, Y.-B., "Memory Management Algorithms for Parallel Simulation", *Information Sciences*, vol. 77, no. 1, pp. 119–140, 1994.

[49] Lin, Y.-B., "Determining the Global Progress of Parallel Simulation", *Information Processing Letters*, vol. 50, 1994.

[50] Samadi, B., *Distributed Simulation, Algorithms and Performance Analysis*, PhD thesis, Computer Science Department, University of California, Los Angeles, 1985.

[51] Mattern, F., "Efficient Distributed Snapshots and Global Virtual Time Algorithms for Non-FIFO Systems", *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423 – 434, 1993.

[52] Fujimoto, R.M., "Parallel Discrete Event Simulation: Will the Field Survive?", *ORSA Journal on Computing*, vol. 5, no. 3, 1993.

[53] Arvind, D., Bagrodia, R. and Lin, Y.-B., Ed., *Proc. 8th Workshop on Parallel and Distributed Simulation.* ACM, 1994.

[54] Bailey, M. and Lin, Y.-B., Ed., *Proc. 9th Workshop on Parallel and Distributed Simulation.* ACM, 1995.

Yi-Bing Lin received his BSEE degree from National Cheng Kung University in 1983, and his Ph.D. degree in Computer Science from the University of Washington in 1990. Between 1990 and 1995, he was with the Applied Research Area at Bell Communications Research (Bellcore), Morristown, NJ. In 1995, he was appointed full professor of Department and Institute of Computer Science and Information Engineering, National Chiao Tung University. His current research interests include design and analysis of personal communications services network, distributed simulation, and performance modeling. He is a subject area editor of the *Journal of Parallel and Distributed Computing*, an associate editor of the *International Journal in Computer Simulation*, an associate editor of *SIMULATION*, a member of the editorial board of *International Journal of Communications*, a member of the editorial board of *Computer Simulation Modeling and Analysis*, Program Co-Chair for the *8th Workshop on Distributed and Parallel Simulation*, and General Chair for the *9th Workshop on Distributed and Parallel Simulation.*

Paul A. Fishwick is an associate professor in the Department of Computer and Information Sciences at the University of Florida. He received the BS in Mathematics from the Pennsylvania State University, MS in Applied Science from the College of William and Mary, and PhD in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a senior member of the IEEE and the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick founded the `comp.simulation` Internet news group (Simulation Digest) in 1987, which now serves over 15,000 subscribers. He was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, *IEEE Transactions on Systems, Man and Cybernetics*, *The Transactions of the Society for Computer Simulation*, *International Journal of Computer Simulation*, and the *Journal of Systems Engineering.*

List of Figures

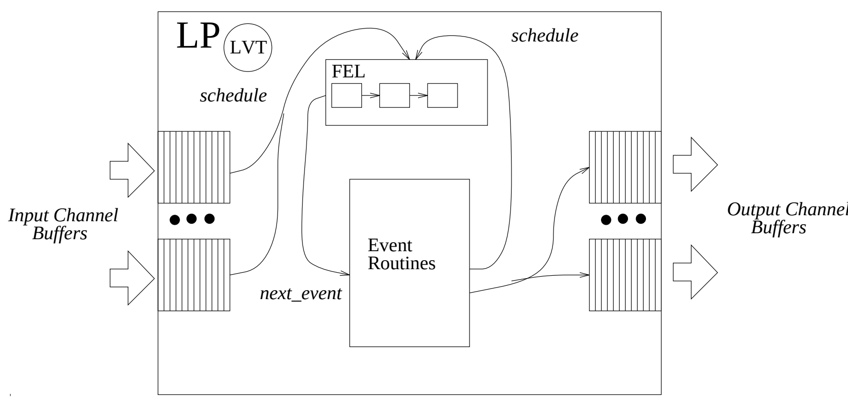


Fig. 1. Anatomy of a logical process (LP).

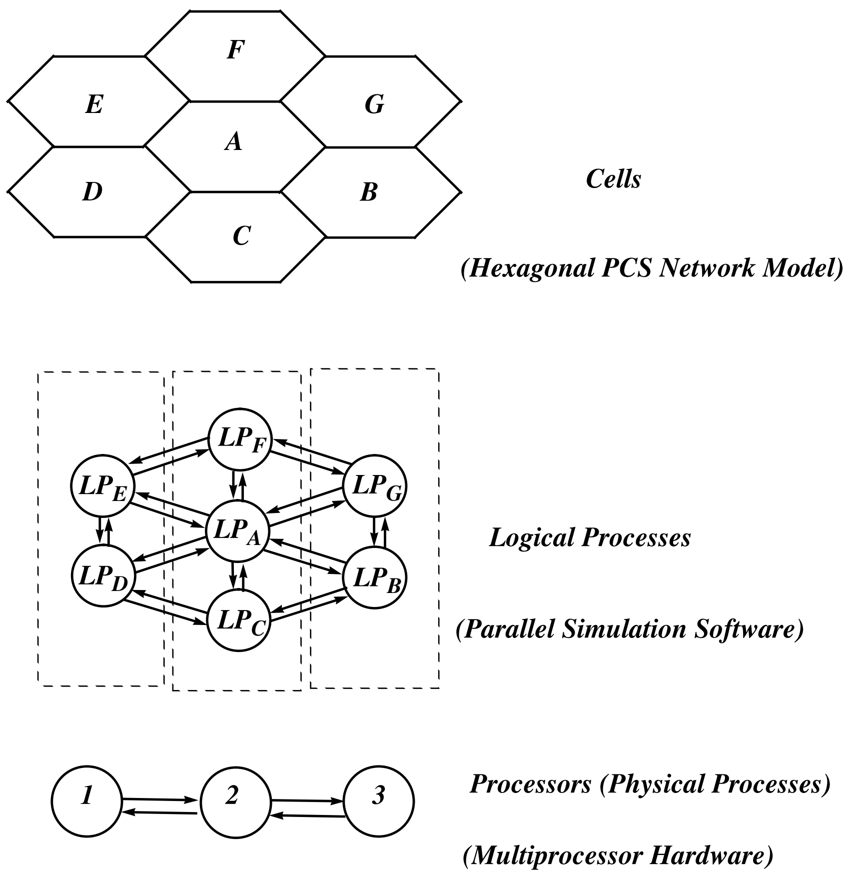


Fig. 2. Cells, logical processes, and processors. A PCS cell is represented by a logical process (LP) in PDES. More than one LP may be mapped to a processor for execution.

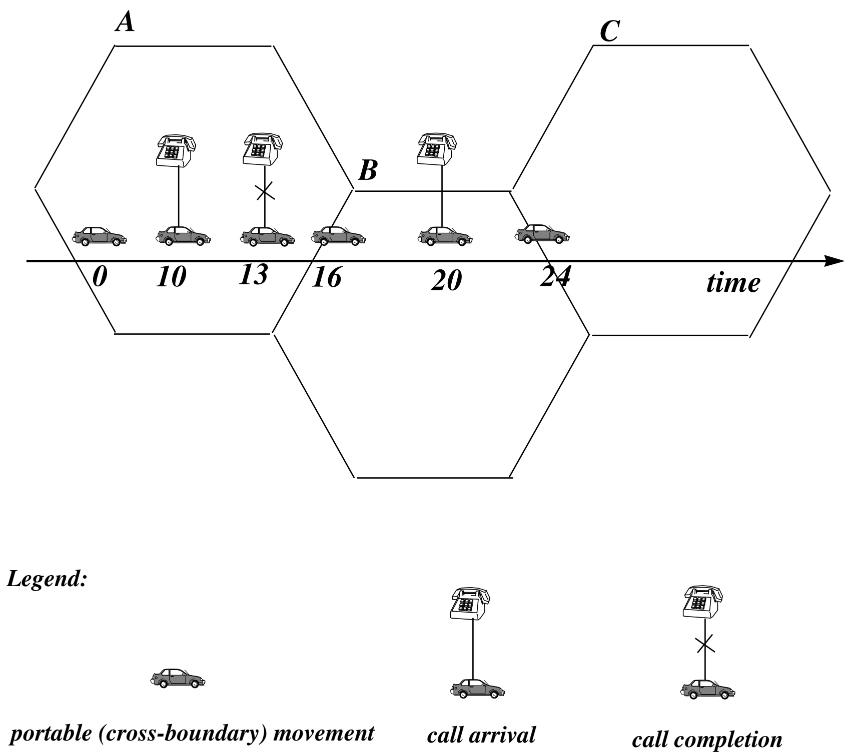
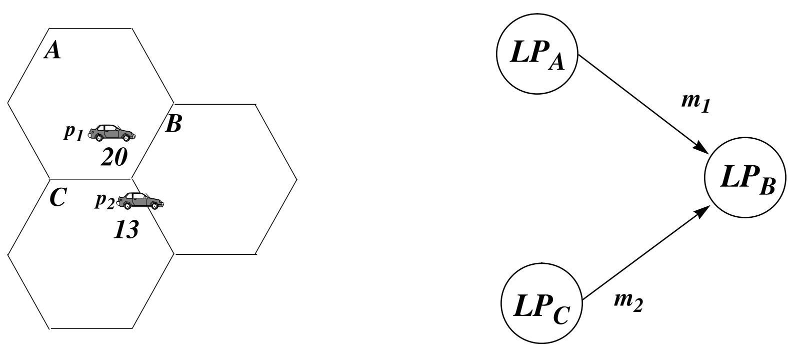


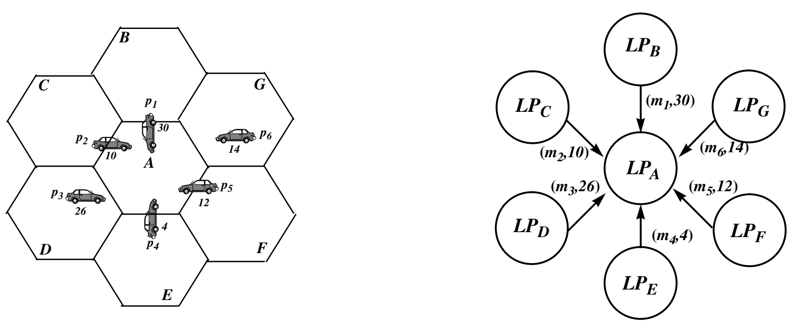
Fig. 3. A simple PCS example.



(a) The PCS model.

(b) The PCS PDES.

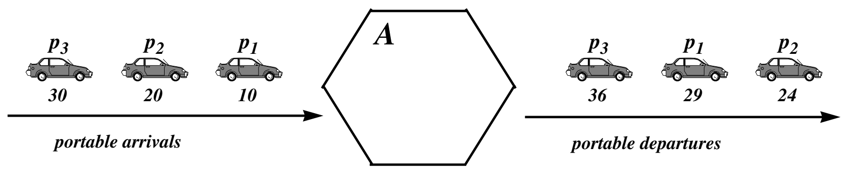
Fig. 4. PDES synchronization problem.



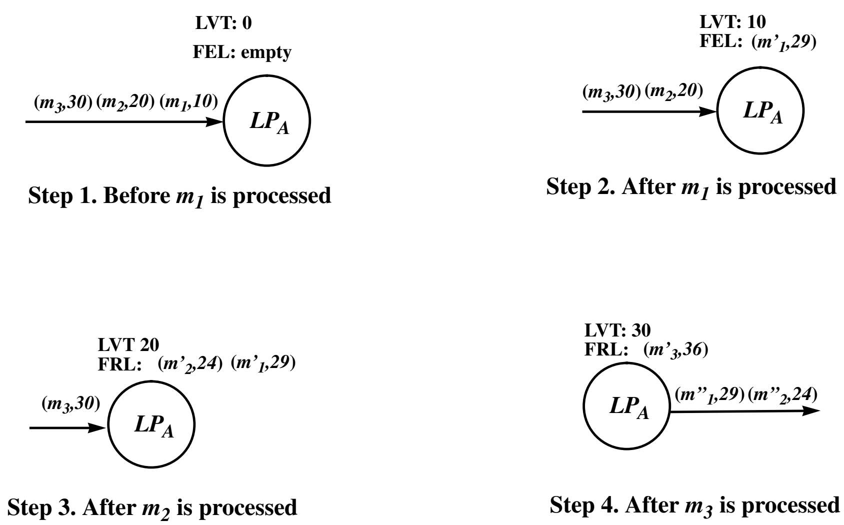
(a) The PCS model.

(b) The PCS PDES.

Fig. 5. The input waiting rule. In (a), the number below a car represents the time when the portable crosses the cell boundary.

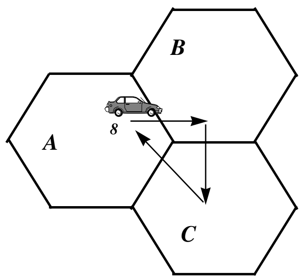


(a) The portables do not depart in the order they arrive.

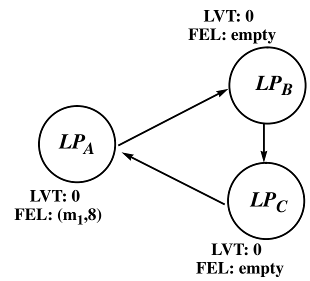


(b) The PCS PDES

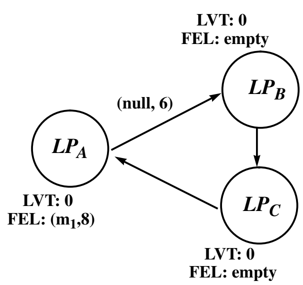
Fig. 6. The output waiting rule.



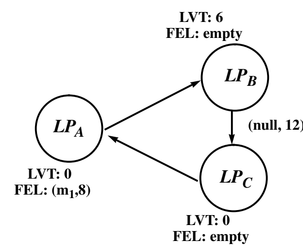
(a) The three-cell PCS model.



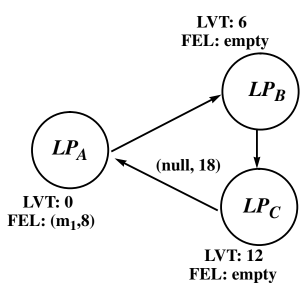
(b) Deadlock in PDES.



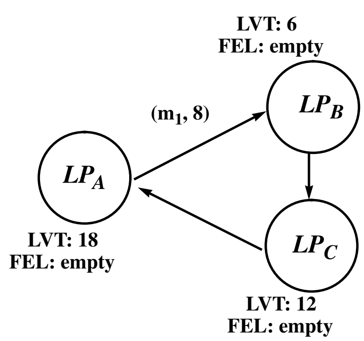
(c) LP_A sends a null message to LP_B.



(d) LP_B sends a null message to LP_C.

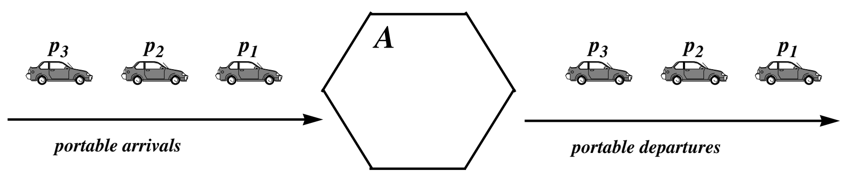


(e) LP_C sends a null message to LP_A.

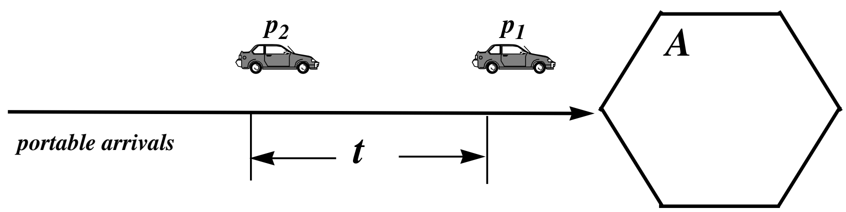


(f) m_1 is processed and sent to LP_B.

Fig. 7. Deadlock and deadlock resolution.



(a) The portables depart in the order they arrive.



(b) Non-zero inter-boundary crossing time ($t > \tau$).

Fig. 8. Examples for lookahead exploiting.

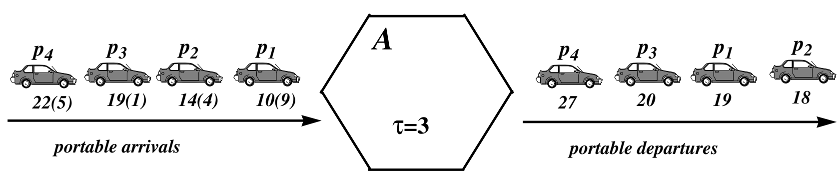


Fig. 9. Portables entering and leaving cell A.

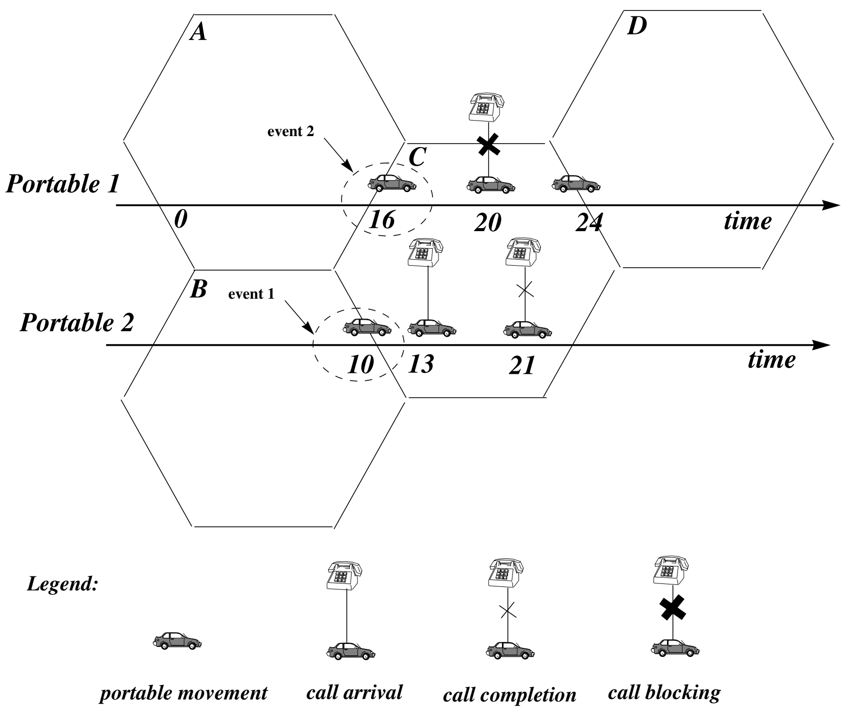
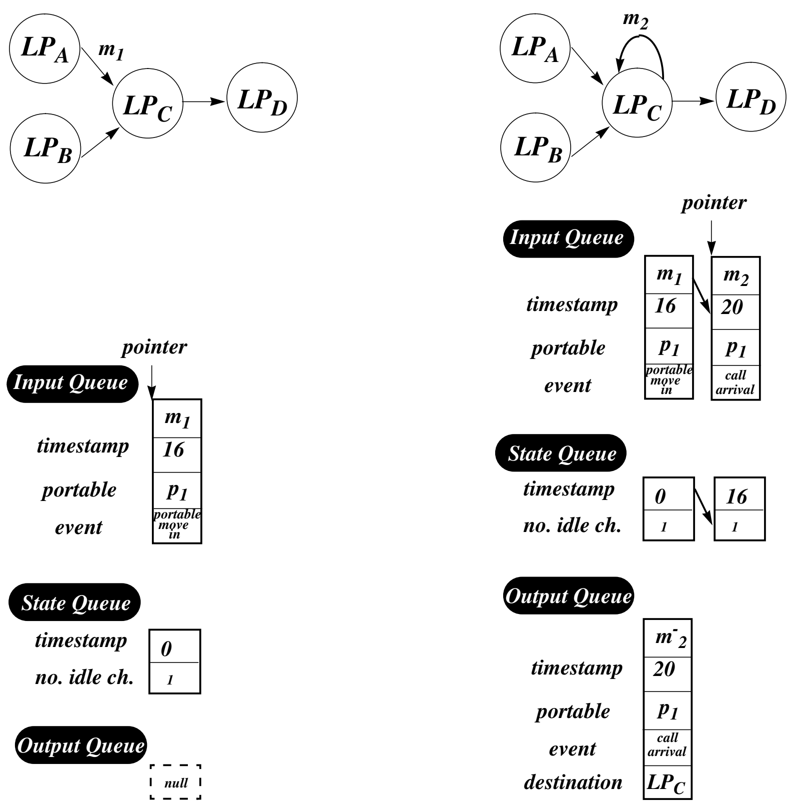


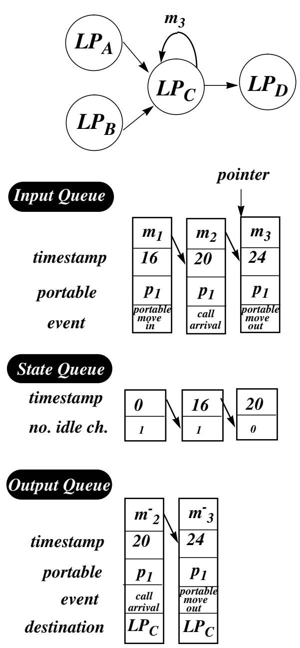
Fig. 10. A PCS example for optimistic PDES. Events 1 and 2 will be represented by messages m_5 and m_1 respectively in the optimistic PDES (see the next figures).



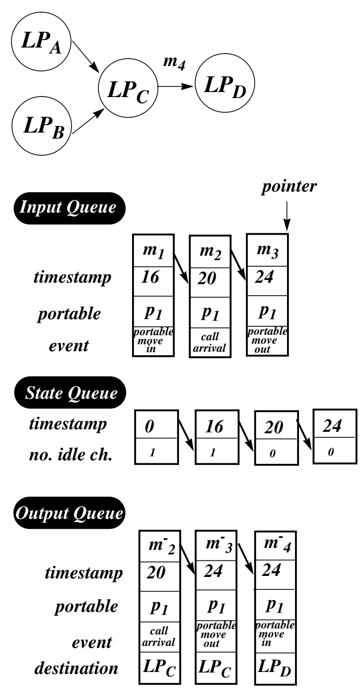
(a) Before m_1 is executed.

(b) After m_1 is executed.

Fig. 11. The data structures of LP_C before/after rollback.

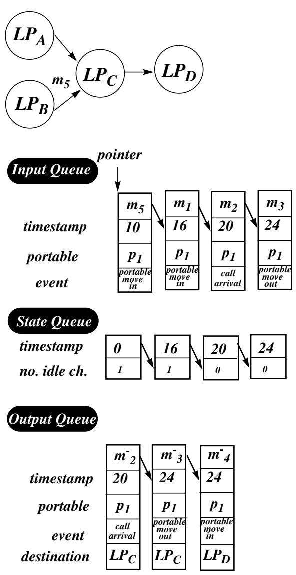


(a) After m_2 is executed.

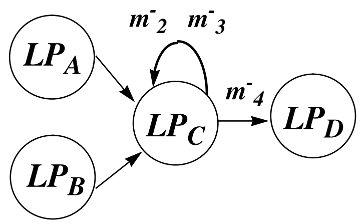


(b) After m_3 is executed.

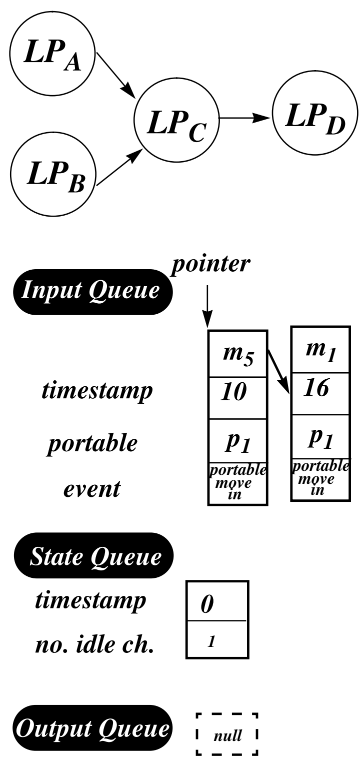
Fig. 12. The data structures of LP_C before/after rollback (cont.).



(a) Before the rollback.

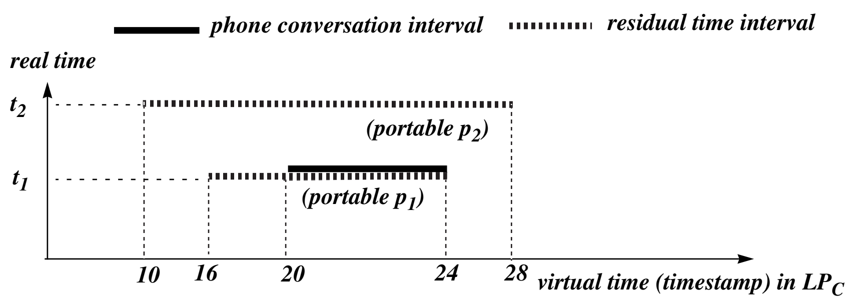


(b) The rollback in progress.

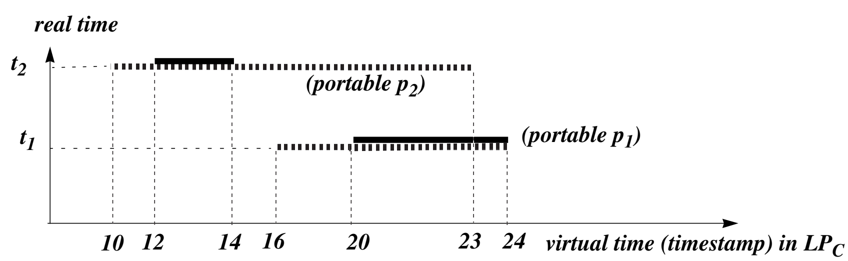


(c) After the rollback.

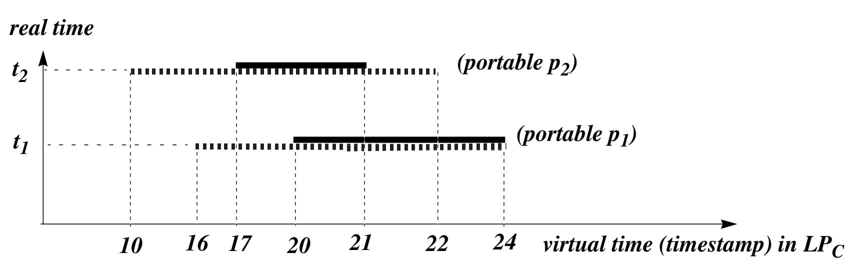
Fig. 13. The data structures of LP_C before/after rollback (cont.).



(a) Situation 1. The number of radio channels in LP_C is 1.



(b) Situation 2. The number of radio channels in LP_C is 1.



(c) Situation 3. The number of radio channels in LP_C is 2.

Fig. 14. Situations when lazy cancellation applies (in these situations, $t_2 > t_1$).

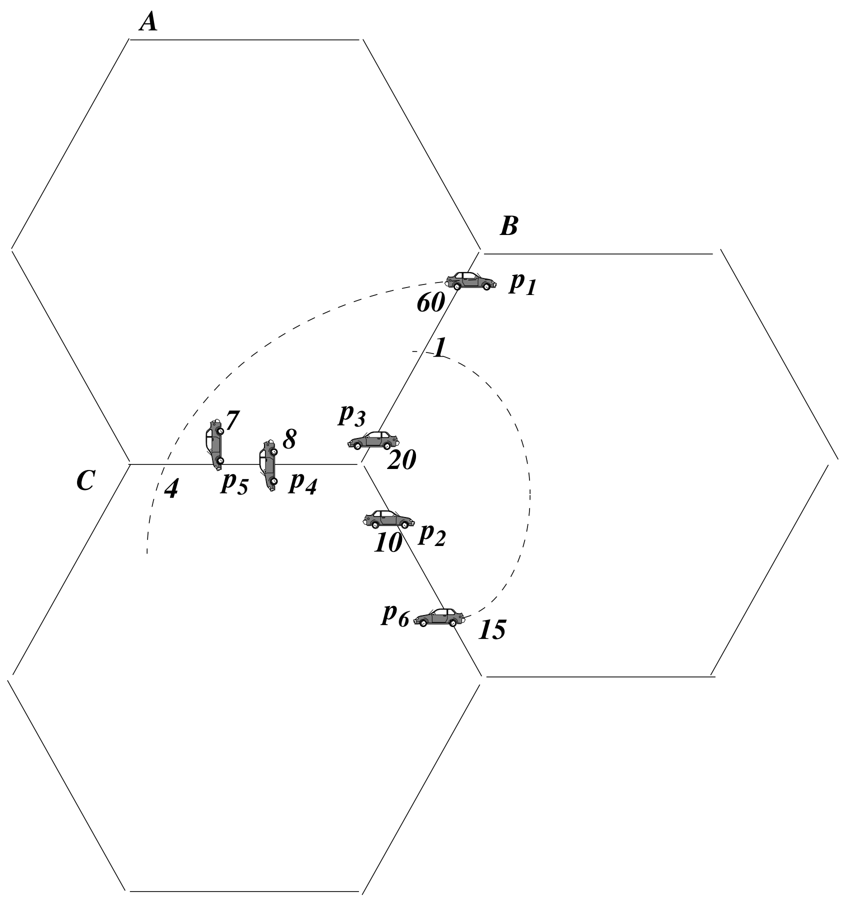


Fig. 15. An PCS example for fossil collection in optimistic PDES.

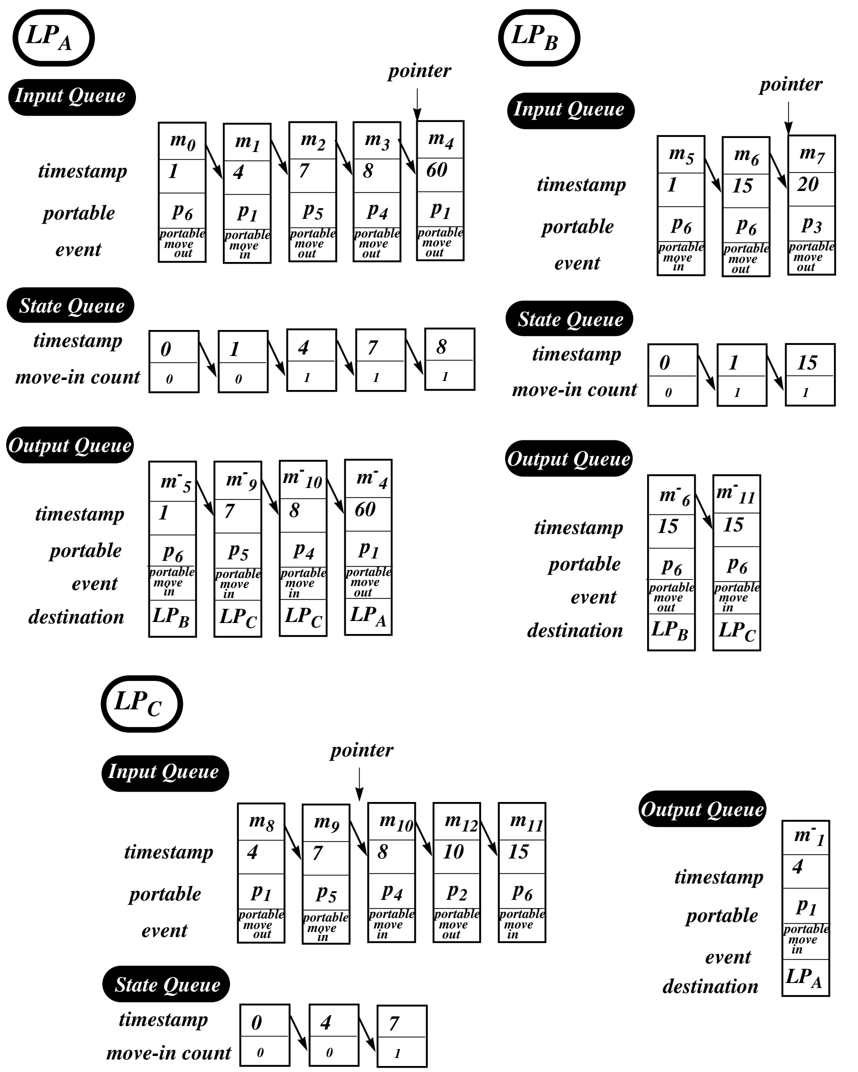


Fig. 16. The optimistic PDES before fossil collection.

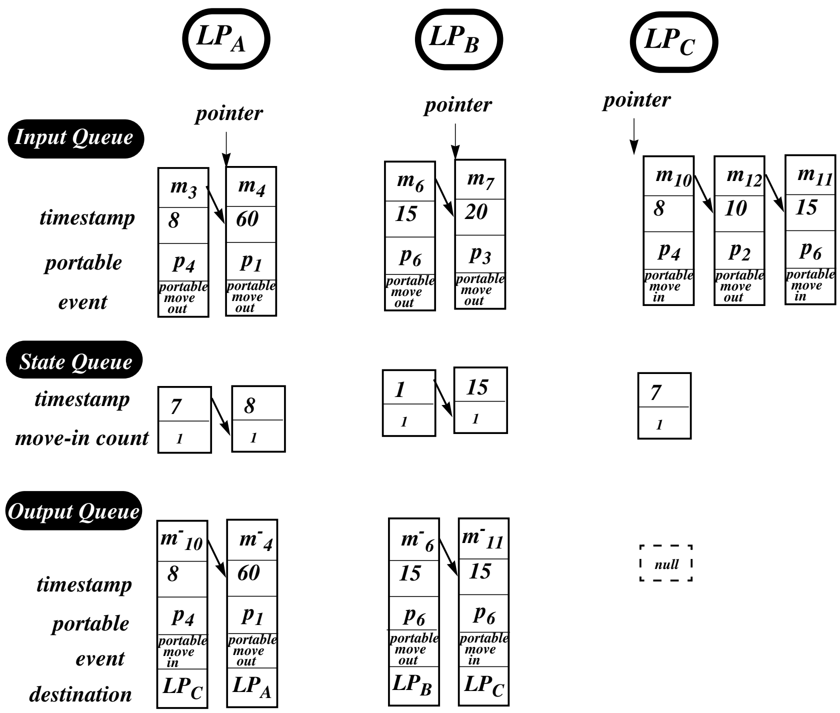
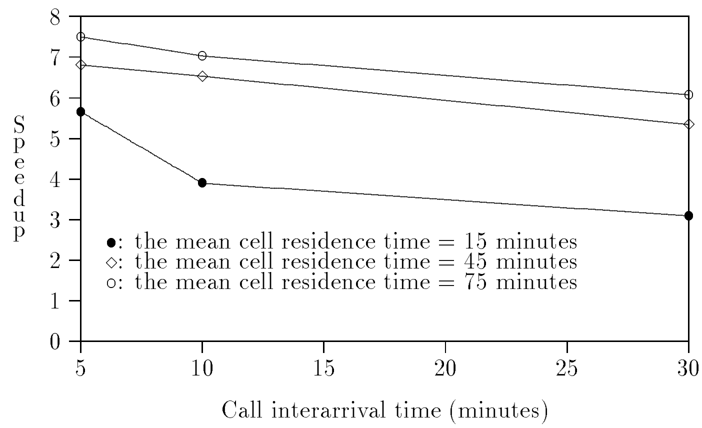
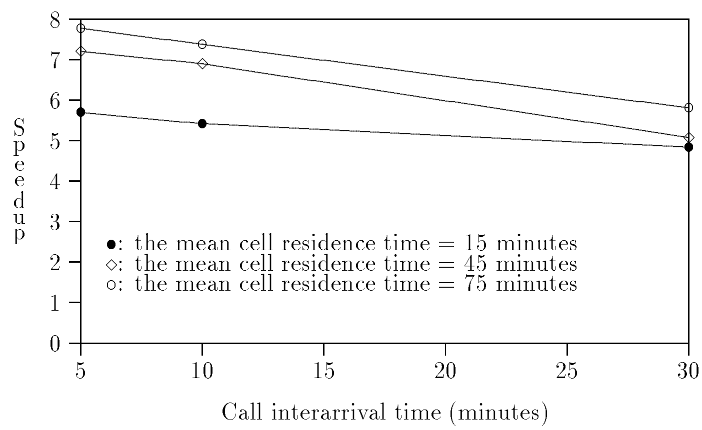


Fig. 17. The optimistic PDES after fossil collection.



(a) The expected number of portables per cell is 50.



(b) The expected number of portables per cell is 75.

Fig. 18. Speedup of the Optimistic PDES (The call holding time is exponentially distributed with mean 3 minutes. Eight processors are used in the parallel simulation.)

List of Tables

Mechanism	Attributes	Methods
Clock	`LVT`	`LVTUpdate()`

FEL	`eventList`	`Enqueue()`
		`Dequeue()`
		`Cancel()`

Synchronization		`ReceiveMessage()`
		`SendMessage()`
		`ExecuteMessage()`

Application	to be elaborated	to be elaborated

TABLE I
Attributes and methods of an LP.