General Principles
Chapter 3

(nib = Not In Book)

Overview

- Concepts and Jargon
- Event scheduling algorithm
- World Views
  - Event scheduling
  - Activity scanning
  - Process interaction
- List Processing

Concepts and Jargon

- See p64 (60-61 in 2nd ed.) for detailed definitions
- Most are obvious, but
- Some are counterintuitive:
  - Entity
  - Activity
  - Delay

Concepts and Jargon

- Static
  - System consists of Entities which have Attributes and has a System State
- Temporal
  - Event is System State change
  - Event notice schedules Event
  - FEL Future Event List
  - Activity is a known period of time
  - Delay is an unknown period of time

Concepts and Jargon

- Generic
  - List is ordered set
  - Clock is the simulated time counter
  - Model is set of relations between everything

Activities explained (nib)

- Activity is time interval between causally related events
- Duration known at start time
- Activity starts with an event E1
  - E1 = Arrival
  - E1 = Service completion
- Ends with *primary* event E2 caused by E1
  - E2 = Next Arrival
  - E2 = Next service completion

Delays explained (nib)

- Delay is time interval between an event E1 and another event E2 which are *not directly* related by cause and effect
- Duration unknown at start time
- Delay starts with an event E1
  - E1 = Arrival of customer C
- Delay ends with event E2
  - E2 = Customer leaves
  - Many events in between if C has to queue up

Event Scheduling Algorithm

- Event scheduling
- Activity scanning
- Process interaction

Discrete Event Scheduling is based on one simple idea (nib)

- One thing leads to another
- Causality
- Events cause other events

Mathematical Viewpoint of Event Scheduling Alg. (nib)

State: $\bar{x}(t) = (x_1(t), \ldots, x_n(t))$.
Event set: $\mathcal{E} = \{E_1, \ldots, E_k\}$.
Global CLOCK: t.
Model: $\mathcal{R} = \{R_1, \ldots, R_k\}$ (1 rule per event).
R_i, $i = 1, \ldots, k$, creates a STATE CHANGE caused by event E_i at time t AND a set of m future EVENT NOTICES $\{E_{k_1}(t_1), \ldots, E_{k_m}(t_m)\}$.
Causality demands $t_j > t$, $j = 1, \ldots, m$.
The m time intervals $[t\ t_i]$ are the m activities created by event E_i.

Event scheduling/time-advance algorithm (3.1.1)

- Assume we have an FEL (future event list) consisting of time ordered event notices: FEL = $\{E_1(t_1), \ldots, E_L(t_L)\}$
- $t_k < t_n$ for $k<n$
- E_k is a data structure (Object) with at least a time member variable (usually more)

Event scheduling/time-advance algorithm: core loop

- Remove first event E from FEL
- Advance clock to t=E.t
- Apply model rule R to E to create:
  - State change
  - Set of events $\{E_1(t_1),..,E_m(t_m)\}$ caused by E
- Insert $\{E_1(t_1),..,E_m(t_m)\}$ into FEL
- Collect data (whatever you're interested in)
- GOTO↑

Event scheduling/time-advance algorithm: bootstrapping

- Define initial state at t=0
- Clear all counters and measurement vars
- Place first event on FEL
- Define termination condition
  - Place special stop-event $E_s(T)$ on FEL to stop at predetermined time T
  - Define stop condition to check for at every event

Event scheduling/time-advance algorithm: summary

- Remove next event from FEL
- Execute this event
- Repeat

Example: 1 Server queue

- State = (Q,S)
  - Q = queue length = 0,1,2,3,…
  - S = 0|1, idle or busy
- Event set = {A,F,S}
  - A = arrival of customer
  - F = server finishes
  - S = stop simulation event

Example: 1 Server queue

- 3 Rules for the 3 events, R_A,R_F and R_S (t_0 is prev. event time)
- R_A : state response to A(t)
  - If $S(t_0)=0 \rightarrow \{Q(t)=0,S(t)=1\}$
  - else $\rightarrow \{Q(t)= Q(t_0)+1,S(t)=1\}$
- R_A : Event notices caused by A(t)
  - Create event notice A(t + getArrivalTime())
  - If $S(t_0)=0 \rightarrow$ Create F(t+getServiceTime())

Example: 1 Server queue

- R_F : state response to F(t)
  - If $Q(t_0)=0 \rightarrow \{Q(t)=0,S(t)=0\}$
  - else $\{Q(t)= Q(t_0)-1,S(t)=1\}$
- R_F : Event notices caused by F(t)
  - If $Q(t_0)>0 \rightarrow$ Create F(t + getServiceTime())
- R_S: Stop simulation and process results

Example: 1 Server queue

- Initialize t=0: {Q(0)=0,S(0)=0}
- FEL = {A(getArrivalTime())}

Example 2: Server queue with impatient customers (nib)

- State = (Q,S)
  - Q = [c1,c2,c3,…] (queue of customers)
  - S = 0|1, idle or busy
  - Customer c new entity, with unique id
- Event set = {A,F,L,S}
  - A = arrival of customer
  - F = server finishes
  - L = Customer leaves queue
  - S = stop simulation event

Example 2 (nib)

- 4 Rules for the 4 events, R_A, R_F, R_L and R_S (t_0 is prev. event time)
- R_A : state response to A
- Create customer entity c
  - If $S(t_0)=0$ → {Q unchanged,S=1}
  - else → {Q.Enqueue(C)),S=1}
- R_A : Event notices caused by A(t)
  - Create event notice A(t + getArrivalTime())
  - If $S(t_0)=0$ → Create F(t+getServiceTime())
  - If $S(t_0)=1$ → Create L(t+getFedupTime(),c)

Example 2 (nib)

- R_F : state response to F
  - If Q.len(t_0)=0 → {Q unchanged,S=0}
  - else {Q.dequeue, S(t)=1}
  - (Q.dequeue removes first in line)
- R_F : Event notices caused by F(t)
  - If Q.len(t_0)>0 → Create F(t+getServiceTime())

Example 2 (nib)

- R_L : state response to L
  - If L.c is in Q, Q.remove(c)
  - Else do nothing (defunct event)
- R_S: Stop simulation and process results

Note that L is a more complex event with a customer property besides a time property.

Example 2: Variant with event removal optimization (nib)

- R_F : state response to F
  - If Q.len(t_0)=0 → {Q unchanged,S=0}
  - else {c=Q.dequeue, S(t)=1}
  - Check FEL for L event for customer c, if so remove from FEL
- R_F : Event notices caused by F(t)
  - If Q.len(t_0)>0 → Create F(t+getServiceTime())

Example 3: Server queue, do it differently (nib)

- State = (Q,S)
  - Q = queue length = 0,1,2,3,…
  - S = 0|1, idle or busy
- Event set = {A,B,F,N,S}
  - A = arrival of customer
  - N = eNqueue customer
  - B = server begins
  - F = server finishes
  - S = stop simulation event

Example 3 (nib)

- 5 Rules for the 5 events, R_A, R_B, R_F, R_N and R_S
- R_A : state
- R_A : Event notices
  - A(t + getArrivalTime())
  - If $S(t_0)=0 \rightarrow B(t)$
  - N(t) (may want to do N(t+epsilon))

Example 3 (nib)

- R_B : state
  - Q--, S=1
- R_B : Event notices
  - F(t + getServiceTime())
- R_F : state
  - S=0
- R_F : Event notices
  - If(Q>0) →B(t) (may want to add small delay)

Example 3 (nib)

- R_N : state
  - Q++
- R_N : Event notices
  - If(S=0) →B(t) (may want to add small delay)

Code Examples

- OneServer.java
- OneServer2.java
- DumpTruck.java

DumpTruck Model

- State = {LQ,L,WQ,W}
  - LQ = 0,1,2,…
  - WQ = 0,1,2,…
  - L = 0,1,2
  - W = 0,1
- Events = {A,FL,FW}
  - Arrive, Finish Loading, Finish Weighing

DumpTruck Model

- Time models:
  - T_L Load time
  - T_W weigh time
  - T_T travel time
- See book for probabilities

DumpTruck Model

- Performance measures:
  - B_L, B_W, loader and weigher utilizations

Let t_k index the event times.

$$B_L = \sum_k L(t_k)(t_k - t_{k-1})/2T_{\text{tot}},$$

where $L(t_k)$ is taken after getting the next event from the FEL but before applying the rule for this. T_{tot} is the total runtime.

$$B_W = \sum_k W(t_k)(t_k - t_{k-1})/T_{\text{tot}}.$$

Activity Scanning

- Uses condition as well as time
- Event happens when its time has come
- Activity starts when conditions are right (this may be time condition)
- Two phase scans at fixed time intervals
- Three phase uses event scheduling to advance time but adds condition controlled activities

Process Interaction

- Create interacting processes
- Each entity lives in a process
- Processes communicate
- Needs event based infrastructure
- Could use threads, but is very hard tp program like this
- Normally used within simulation software

Process Interaction, 1 server example

Customer process:

```
c = new Customer();
queue.enqueue(c);
// sleep till is turn
wait(notification by server);
exit();
```

Server Process:

```
forever {
  c=queue.dequeue(); //blocks
  wait(getServTime());
  notify(c);
}
```

Main program:

Create Server process;
Create Customers at interarrival times;