Outline

• Fundamental concepts: Processes, resources
• Simulation primitives
• Example
• Implementation
Process Oriented Simulation

• Focus simulation program around behavior of entities
  – Aircraft: arrives, waits for runway, lands, departs

• Process-oriented simulation
  – Process: thread of execution describing entity behavior over time
  – Resources: shared resource used by entities (e.g., runway)

• Execution: alternate between
  – simulation computations at a single instant of simulation time, and
  – advances in simulation time (no computation)
**Event vs. Process Oriented Views**

**Event oriented view**

**State variables**
- Integer: InTheAir;
- Integer: OnTheGround;
- Boolean: RunwayFree;

**Event handlers**
- Arrival Event
  - `{ … }`
- Landed Event
  - `{ … }`
- Departure Event
  - `{ … }`

Entities modeled by event handlers

**Process oriented view**

**State variables**
- Integer: InTheAir;
- Integer: OnTheGround;
- Boolean: RunwayFree;

**Processes**
- Aircraft 1
  - `{ Arrive Land Depart }`
- Aircraft 2
  - `{ Arrive Land Depart }`
- Aircraft n
  - `{ Arrive Land Depart }`

Entities modeled by processes
Simulation Primitives

Primitives needed to advance simulation time

• **AdvanceTime(\(T\))**: advance \(T\) units of simulation time
  – Also called “hold”
  – E.g.: AdvanceTime(R) to model using runway R units of simulation time

• **WaitUntil(\(p\))**: simulation time advances until predicate \(p\) becomes true
  – Predicate based on simulation variables that can be modified by other simulation processes
  – E.g.: WaitUntil(\text{RunwayFree}) to wait until runway becomes available for landing

• **Other combinations**
  – **WaitUntil(\(p, T\))**: Wait up to \(T\) units of simulation time for predicate \(p\) to become true
  – Not used in the air traffic example
Process Model Example: Aircraft

A new aircraft process is created with each Arrival event

/* simulate aircraft arrival, circling, and landing */

Integer: InTheAir;
Integer: OnTheGround;
Boolean: RunwayFree;

1  InTheAir := InTheAir + 1;
2  WaitUntil (RunwayFree); /* circle */
3  RunwayFree := FALSE;    /* land */
4  AdvanceTime(R);
5  RunwayFree := TRUE;
   /* simulate aircraft on the ground */
6  InTheAir := InTheAir - 1;
7  OnTheGround := OnTheGround + 1;
8  AdvanceTime(G);
   /* simulate aircraft departure */
9  OnTheGround := OnTheGround - 1;
Execution Example

State Variables

<table>
<thead>
<tr>
<th>RunwayFree</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>InTheAir</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OnTheGround</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simulation Time

Flight 1
1 InTheAir := InTheAir+1;
2 WaitUntil (RunwayFree);
3 RunwayFree := FALSE;
4 AdvanceTime(R);
5 RunwayFree := TRUE;
6 InTheAir := InTheAir-1;
7 OnTheGround := OnTheGround+1;
8 AdvanceTime(G);
9 OnTheGround := OnTheGround-1;

Flight 2
1 InTheAir := InTheAir+1;
2 WaitUntil (RunwayFree);
3 RunwayFree := FALSE;
4 AdvanceTime(R);
5 RunwayFree := TRUE;
6 InTheAir := InTheAir-1;
7 OnTheGround := OnTheGround+1;
8 AdvanceTime(G);
9 OnTheGround := OnTheGround-1;
Implementation

Process-oriented simulations are built over event oriented simulation mechanisms (event list, event processing loop)

• Event computation: computation occurring at an instant in simulation time
  – Execution of code section ending with calling a primitive to advance simulation time

• Computation threads
  – Typically implemented with co-routine (threading) mechanism

• Simulation primitives to advance time
  – Schedule events
  – Event handlers resume execution of processes
## Aircraft Process

Identify computation associated with each simulation event

```plaintext
/* simulate aircraft arrival, circling, and landing */

1  InTheAir := InTheAir + 1;
2  WaitUntil (RunwayFree); /* circle */

3  RunwayFree := FALSE; /* land */
4  AdvanceTime(R);

5  RunwayFree := TRUE;

/* simulate aircraft on the ground */

6  InTheAir := InTheAir - 1;
7  OnTheGround := OnTheGround + 1;
8  AdvanceTime(G);

/* simulate aircraft departure */

9  OnTheGround := OnTheGround - 1;
```

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>InTheAir := InTheAir + 1;</td>
<td>Aircraft Arrival</td>
</tr>
<tr>
<td>2</td>
<td>WaitUntil (RunwayFree); /* circle */</td>
<td>Aircraft Arrival</td>
</tr>
<tr>
<td>3</td>
<td>RunwayFree := FALSE; /* land */</td>
<td>Aircraft Landing</td>
</tr>
<tr>
<td>4</td>
<td>AdvanceTime(R);</td>
<td>Aircraft Landing</td>
</tr>
<tr>
<td>5</td>
<td>RunwayFree := TRUE;</td>
<td>Aircraft On The Ground</td>
</tr>
<tr>
<td>6</td>
<td>InTheAir := InTheAir - 1;</td>
<td>Aircraft On The Ground</td>
</tr>
<tr>
<td>7</td>
<td>OnTheGround := OnTheGround + 1;</td>
<td>Aircraft On The Ground</td>
</tr>
<tr>
<td>8</td>
<td>AdvanceTime(G);</td>
<td>Aircraft On The Ground</td>
</tr>
<tr>
<td>9</td>
<td>OnTheGround := OnTheGround - 1;</td>
<td>Aircraft Departs</td>
</tr>
</tbody>
</table>
**Implementation: AdvanceTime(T)**

Causes simulation time in the process to advance by T units

Execute AdvanceTime(T):
- Schedule Resume event at time Now+T
- Suspend execution of thread
- Return execution to event scheduler program

Process Resume event:
- Return control to thread

```
Simulation process
...
RunwayFree := FALSE;
AdvanceTime(R);
RunwayFree := TRUE;
...
```

Scheduler
{
While (sim not done)
    Remove event from PEL
call event handler
}
later

Resume Event Handler
{
Xfer to sim process
}
Implementation: `WaitUntil(p)`

Suspend until predicate `p` evaluates to true

Execute `WaitUntil(p)`:

- Suspend execution of thread, record waiting for `p` to become true
- Return execution to event scheduler program

Main scheduler loop

- For each suspended process, check if execution can resume
- Prioritization rule if more than one can resume

Simulation process

```java
... 
InTheAir:=InTheAir+1;
WaitUntil(RunwayFree);
RunwayFree:=FALSE;
... 
```

```c
WaitUntil(p)
{
    Add to suspended list
    Xfer to Schedule
}
```

Scheduler

```c
{
    While (sim not done)
    Remove ev from PEL
    call event handler
    while (a process's pred is true)
    Xfer sim process
}
```

later
• Theoretically, both views are equivalent:
  – Process-oriented simulations can be transformed to event-oriented simulations and vice versa
  – How?

• Practically, runtime performance differs:
  – Event-oriented views typically execute faster than process-oriented views
  – Why?
Summary

• Process-oriented simulation typically simplifies model development and modification
• Requires threading (e.g., co-routine) mechanism
• Additional complexity and computation overhead to suspend and resume simulation processes