Pipes
Discrete Event Simulator Implementation in Java

Razvan Surdulescu

CS380n, Spring 2003

Part 1

Project Goals

Literature & Implementations

High-Level Design

Goals

- Learn about DES
  - Survey current literature on DES
  - Investigate existing DES implementations
- Implement a DES
  - Write a free, simple Workbench clone in Java
  - Explore high-level architectural aspects
  - Explore low-level implementation aspects

Literature

- Sources
  - CS380n lecture notes, Workbench [1]
  - "Introduction to DES" by Peter Ball [2]
  - "DES in Java" by Keld Helsgaun [3]
- There are many papers, books, tools
  - Limited use on this project
  - Focused on learning a lot on my own

Literature cont'd

- Overview of DES
  - "Discrete event simulation is one way of building up models to observe the time based (or dynamic) behavior of a system."[2]
- DES constituents[2]
  - Entities/relations
    - The components that form the system
    - Entities can be permanent (assembly line machinery) or temporary (parts built on assembly line)

Literature cont'd

- DES constituents cont'd
  - Executive
    - Responsible for advancing the clock or generating events
  - Clock/event queue
    - Orders the progress of entities through the system
  - Distributions
    - The random number generator(s) from which entities or events are drawn or generated

Literature cont'd

- DES constituents cont'd
  - Results collection
    - Compute statistics about the simulation and present the results to the user
- Time advance approaches[2]
  - Time slicing
    - Time advances in discrete intervals
  - Next event
    - Time advances to next significant event

Literature cont'd

- Logic approaches[2]
  - Event
    - Instantaneous change in an entity
    - (+) Easy to understand, computationally efficient
    - (-) More difficult to implement than activity
  - Activity
    - Duration in an entity
    - (+) Relatively easy to understand
    - (-) Poor execution efficiency
  - Process
    - Joins events and activities to describe life cycle
    - (+) Less common, more planning required to implement
    - (-) Computationally efficient

Literature cont'd

- OO techniques in simulation
  - DES libraries are extensible and self-contained
    - Can be easily adapted to many different kinds of problems
    - Easy to use, do not require a lot of setup time or documentation
  - DES libraries can be shared and integrated
    - Easy to incorporate them into existing systems

Literature cont'd

- Continuous vs. discrete simulation[2]
  - A continuous approach computes measures by means of equations (or systems of ~)
  - Continuous and discrete simulation approaches can be integrated nicely
    - Events trigger the computation of a continuous measure upon arrival or departure

Implementations

- HyPerformix's Workbench
  - Familiar with it from CS380n
- University of Edinburgh's SimJAVA
  - Process based DES package in Java
  - A simulation is a collection of entities each running in a thread, connected by ports and communicating via event objects.
  - A central system controls all the threads, advances the simulation time, and delivers the events.
  - Progress is recorded through trace messages.

High-Level Design

- Workbench clone
  - Use what works in Workbench
    - Similar way of constructing the model
    - Identical model components
  - Improve what is outdated in Workbench
    - Modern menus, frames, components, CUA
    - Models stored in XML
    - OS independent (Java)

High-Level Design cont'd

- Approach
  - Event based simulator
  - A few concepts from SimJAVA ("ports", etc.)
- Automation
  - Model independent of UI
    - Models can be simulated programmatically
  - Extensible design
    - New widgets or statistics can be added easily

High-Level Design cont'd

- Clean-room implementation
  - Purposefully avoided existing tools
    - More opportunity for learning
    - Tease out own design patterns
    - No 3rd party dependencies or idiosyncrasies
- Small
  - Final binary is ~160k
  - Can run as an applet, great for demos or teaching

High-Level Design cont'd

- Features
  - Components
    - Nodes: Source, Sink Service, Delay, Resource, Allocate, Release
    - Arcs: select transactions by probability, category
  - Transactions
    - Have category and priority
  - Resources
    - Are passive, non-addressable

High-Level Design cont’d

- Features cont’d
  - Queues
    - Time rules: FIFO, LIFO
    - Queuing priorities: No Priority, Next Priority, Highest Priority
  - Statistics
    - Arrival rate, population, queue population, utilization, response rate, response time

High-Level Design cont'd

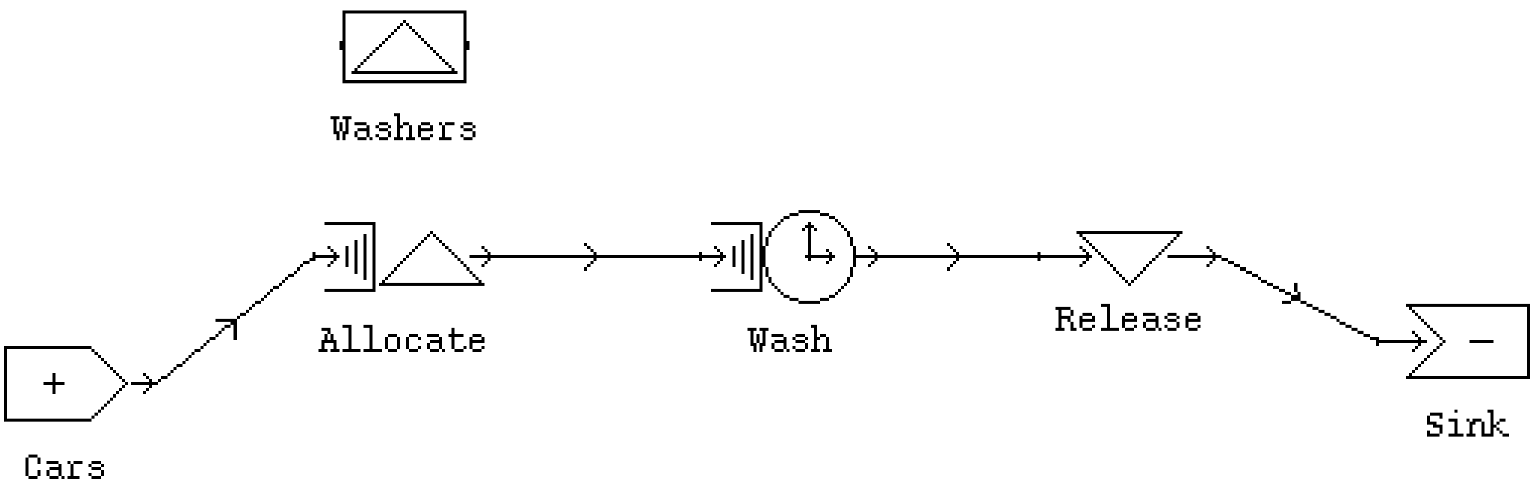
- Features cont'd
  - Simulation
    - Warm-up time, maximum time length
    - Log can be exported to text file
    - Entirely event driven: Java is very poor at RT

Demo

- The car wash problem [3]
  - A car wash services cars one at a time. When a car arrives, it goes straight into the car wash if this is idle; otherwise, it waits in a queue. The car wash operator waits when he has no work to do; otherwise he is in continuous operation serving on a FIFO basis.
  - Each wash takes exactly 10 minutes. The average time between car arrivals has been estimated at 11 minutes.
  - Predict the maximum queue length and average wait time if one more car wash (an one more car wash operator) is installed.

Demo cont'd

- Workbench model
  - Washers contains 1, then 2 tokens



Demo cont'd

- Results (1 token, 2000 transactions):
  - Workbench:
    - Allocate node:
      - Queue length: 4.70
      - Response time: 51.798
  - Pipes:
    - Allocate node:
      - Queue length: 4.80
      - Response time: 52.298

Demo cont'd

- Results (2 tokens, 2000 transactions):
  - Workbench:
    - Allocate node:
      - Queue length: 0.12
      - Response time: 1.41
  - Pipes:
    - Allocate node:
      - Queue length: 0.13
      - Response time: 1.41