Next Event Simulations

October 25, 2025

We skip § 5.1.3–§ 5.2.3 in the text because they are either

i. explanation of, and anti-motivation for, writing complicated and fragile simulation code (which upper level course students should **NOT** need to be convinced of...)
ii. or rather laborious explanations of straight-forward augmentation to our favorite example systems (SSQ, SiS), which are actually *easy to do* **if you write a well-designed simulation to begin with**...

Instead we keep our eyes on the prize, focus on the design pattern used for all respectible discrete event simulations.

Beware Algorithms 5.1.2 (SSQ) and 5.2.1 (SiS) they are unfortunate examples of **HOW NOT TO WRITE GOOD SIMULATIONS**.

If you choose to write your next event simulations like this,
please don't say you learned it from me and **don't expect full-credit for your future work** in this course.

- **Next Event Simulation** (NES) is a design pattern for writing efficient, structured, and malleable DESs.

    a. System State ($\mathcal{S}$)

    b. Events

    c. Simulation Clock (t)

    d. Event Scheduling

    e. Event "List" ($\mathcal{E}$)

- The **System State** ($\mathcal{S}$) is a complete characterization of the system **we are modeling**... which is not the same as a "snapshot of the simulation (which is usually *larger*, containing more information)."
- The definition of $\mathcal{S}$ for a particular simulation evolves from the *Conceptual* through *Computational* model:

  Conceptual: Abstract collection of **values** (or collections of values) and how they change over time.

  Specification: Values of the Conceptual Model restated as **variable names** and their **numerical or collection type**, along with the equations or algorithms associated with them, and data **from the real world** that informs our models and algorithms.

  Computational: The data types or data structures types used to hold specification variables and the implementation of algorithms modifying them.
- The state of an SSQ is simply the number of jobs in the whole SSQ structure (queue and service node).
- SiS state is simply the size of its inventory.

- Conventionally, Events are implemented as **objects** in the programming language, and all events have data members representing their **activation time** (*.at*) and their **event type** (*.type*).
- A **System Event** is an occurance that may change $\mathcal{S}$
  - SSQ: job arrival, job completion
  - SiS: demand, inventory review
  - SiS with delivery delays?

- Conventionally, Events are implemented as **objects** in the programming language, and all events have data members representing their **activation time** (*.at*) and their **event type** (*.type*).
- A **System Event** is an occurance that may change $\mathcal{S}$
  - SSQ: job arrival, job completion
  - SiS: demand, inventory review
  - SiS with delivery delays?
    Events: demand, delivery, inventory review
    State is inventory **and inventory-on-order**
- A **Meta Event** is a simulation specific occurance that **cannot** change $\mathcal{S}$
  - terminating the simulation,
  - writing log or data files,
  - statistical sampling of $\mathcal{S}$ or the simulation architecture run-time

- The Event "List" ($\mathcal{E}$) holds `Event` objects, they are removed from $\mathcal{E}$ prioritized by smallest activation time (the **imminent event**).
- In NES, the simulation clock t **lurches forward from activation time to activation time**, instead of progressing by a fixed increment "time step".
- To Build an NES ...

  i. Determine your system state ($\mathcal{S}$)
  (in the Conceptual and Specification Model)

  ii. Identify your System Event types and how they interact with $\mathcal{S}$
  (the Specification and Computational Models)

  iii. Formalize your System Events in code, define and implement Meta Events as needed for experimentation, data output.
  (Computational Model)

(Real) Next Event Simulation

(Wordy Version)

1. Initialize simulation clock, simulation state, and event list
2. While event list is not empty (or until some simulation terminating event),
   i. Remove the imminent event from the event list (smallest $.at$)
   ii. Advance the simulation clock to this imminent event's activation time
   iii. Process this event (depending on $.type$)
   iv. Schedule the occurance of any future events spawned from this activated event

(Mathy Version)

1. Initialize simulation clock t, state $\mathcal{S}$, and event list $\mathcal{E}$
2. While $\mathcal{E}$ is not empty,
   i. Find $e_m \in \mathcal{E}$ with smallest activation time $.at$, $e_m = \underset{e_i.at}{\arg\min}\ \mathcal{E}$

   (THINK: "m" FOR **IMM**INENT)
   ii. Advance the simulation clock ($t \leftarrow e_m.at$)
   iii. Process e_m, which may[1] add more events e_i to $\mathcal{E}$ with $e_i.at > e_m.at = t$

[1] More "likely" than "may."

Let’s look at LGA-NES