Basic Notions of Discrete-Event Simulation (DES)

Computational Models for Complex Systems

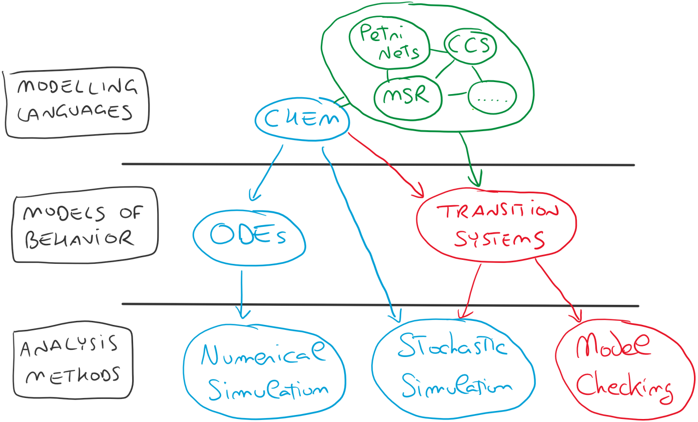
Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://pages.di.unipi.it/milazzo
milazzo@di.unipi.it

Laurea Magistrale in Informatica
A.Y. 2019/2020

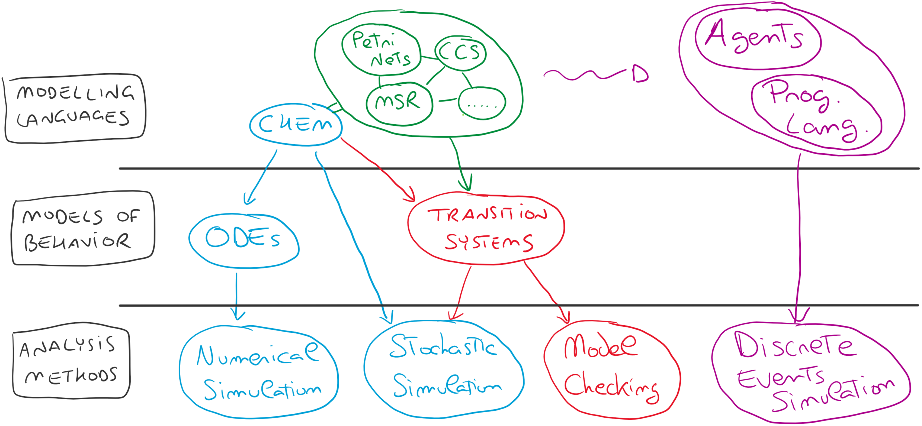
Roadmap

What we did...



Roadmap

... and what we are going to do!



Introduction

We have seen Stochastic Simulation (e.g. Gillespie's algorithm and PRISM simulation feature) as a simulation approach for discrete-state systems

Stochastic simulation produces descriptions of possible system's behaviours as sequences (traces) of instantaneous events

The frequency of events is based on an exponential distribution

The memoryless property of exponential distribution makes each simulation step independent from the previous ones

Gillespie's Stochastic Simulation Algorithm (SSA)

Gillespie's algorithm (REMINDER):

The state of the simulation:

- is a vector representing the multiset of molecules in the chemical solution (initially $[X_1, \ldots, X_n]$)
- a real variable t representing the simulation time (initially $t = 0$)

The algorithm iterates the following steps until t reaches a final value t_{stop}.

1. The time $t + \tau$ at which the next reaction will occur is randomly chosen with τ exponentially distributed with parameter $a_0 = \sum_{\nu=1}^{M} a_\nu$;
2. The reaction R_μ that has to occur at time $t + \tau$ is randomly chosen with probability $\frac{a_\mu}{\sum_{\nu=1}^{M} a_\nu}$ (that is $\frac{a_\mu}{a_0}$).

At each step t is incremented by τ and the multiset representing the chemical solution is updated by subtracting reactants and adding products.

Introduction

Some weaknessess of Stochastic Simulation:

- in general, events of a system could be not instantaneous
- the frequency of events could follow different distributions, such as:
  - fixed delay
  - uniform distribution within an interval
  - gaussian distribution
  - conditional delay (wait until a certain condition is satisfied)
  - ...

Discrete-Event Simulation is a more general simulation approach that allows all of these timing issues to be easily dealt with

Classical example: customers queue

Consider the following (classical) example of customer service with one operator:

- Customers arrive and join the queue
- When the operator is free, he/she starts serving the next customer in the queue
- Serving a customer requires some time, after which the operator is free again

Classical example: customers queue

We may imagine that the different events have different timings...

Event	*Timing*
arrival of a new customer	exponentially distributed with rate λ
customer moving from queue to service	condition based (i.e. "*when the customer is the first of the queue and the operator is free*")
customer served	gaussian distribution with mean μ and variance σ^2

Memoryless-property cannot be assumed!

- we cannot choose one of the enabled events, update the simulator clock, handle the event and continue (as in Gillespie's case)

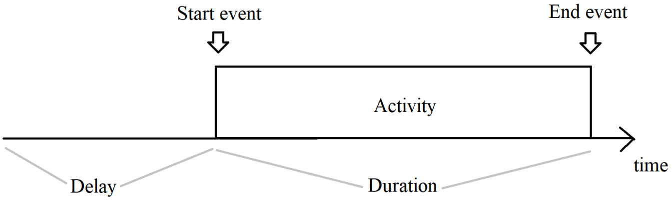
Discrete-Event Simulation

Discrete Event Simulation (DES) allows these systems to be simulated by maintaining an event list

Some terminology:

- State: is a description of a system configuration in terms of a set of variables
- Activity: is a process that involves a sequence of updates of the state variables (events) over time. It has a duration
- Event: is an instantaneous update of the state variables. For example they can correspond to the start and end of an activity
- Event notice: is the description of a future event with the time at which it will happen
- Future Event List (FEL): is a list of event notices ordered by time

Discrete-Event Simulation



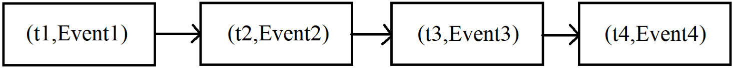
In general, an activity may trigger other events between Start and End...

The Future Event List (FEL)

The Future Event List (FEL) controls the simulation

The FEL contains notices of all future events that are scheduled

The FEL is ordered by increasing time of event notice



$$t1 \leq t2 \leq t3 \leq t4$$

The Future Event List (FEL)

Idea of the simulation algorithm:

1. initialize state variables, the FEL (with one or more pre-schedueled events) and a global clock variable $T = 0$
2. iteratively:
   - remove the first event notice $(t, Event)$
   - handle $Event$. This may require updating state variables, adding one or more event notices in FEL and possibly (but unusually) removing one or more scheduled events from FEL
   - update the global clock $T = t$
3. until a stop time $T = T_{stop}$ is reached or the FEL is empty

Conditional and Primary Events

Actually, events that have to wait for a condition in order to be enabled make the picture a little bit more complex

Let us distinguish:

- Primary events: events whose occurrence is scheduled at a certain time
  - The arrival of a new customer
- Conditional events: events that are triggered by a certain condition becoming true
  - Customer moving from queue to service
    (waits for the operator to be free)

Conditional events are untimed.

- They could be stored either at the beginning of the FEL or in a separate data structure

The Simulation Algorithm

Revised simulation algorithm (with conditional events):

1. Initialize variables, FEL and T
2. Iterate:
   - If there is a conditional event enabled, remove and process it (T does not change)
   - Otherwise, remove and process the first primary event (and update T)
3. Until $T = T_{stop}$ or FEL empty

The Customer Queue

State variables:

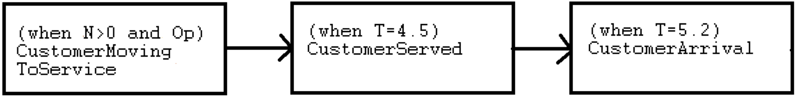
- `int N`: the length of the queue
- `bool Op`: the availability of the operator

Events (in pseudo-code):

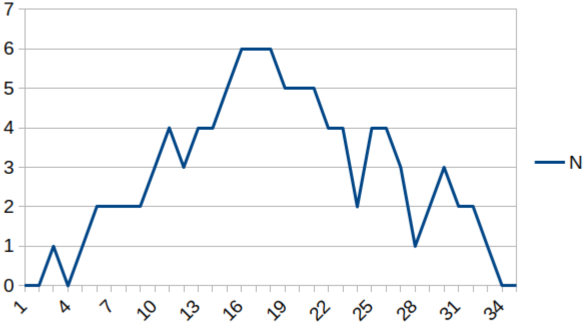
- [CustomerArrival]
  ```
  N:=N+1;
  schedule(CustomerArrival,T+Exp(λ));
  ```
- [CustomerMovingToService]
  ```
  WHENEVER (N>0) and (Op=true)
  N:=N-1; Op:=false;
  schedule(CustomerServed,T+Gauss(μ,σ^2));
  schedule(CustomerMovingToService);
  ```
- [CustomerServed]
  ```
  Op:=true;
  ```

The Customer Queue

Example of FEL at run-time:



Tracking the value of the state variables over time we obtain a possible dynamics of the system



Implementation of a Discrete-Event Simulator

Many specialized modeling languages exist for DES. Some examples:

- Arena (`https://www.arenasimulation.com/`)
  Commercial, but with free student licence
- FlexSim (`https://www.flexsim.com/`)
  Commercial, but with free student licence
- SimPy (`https://simpy.readthedocs.io`)
  Open source Python Library
- System C (`http://www.systemc.org/`)
  Open source C++ Library

Implementation of a Discrete-Event Simulator

A Discrete-Event Simulator can be easily implemented in any general purpose programming language

Typically, object-oriented languages are preferred (Java is the most used in this context) since interfaces and inheritance mechanisms make the management of the FEL very natural

```
public class CustomerArrival implements Event { ... }
```

Implementation of a Discrete-Event Simulator

The implementation of a Discrete-Event Simulator with a general purpose language require to pay some attention on the implementation of the FEL

Typically, most of the CPU time is spent in FEL operations:

- Removals of the first element (very common)
- Insertion in the middle of the list (very common)
- Removals in the middle of the list (quite rare)

It is often better to resort to some tree-based representation of the FEL in order to pay $O(\log n)$ for the insertion operations rather than $O(n)$ (but sacrifying something on the removals)