Structure of the Simulator

Event Scheduling Approach

Discrete-event simulation

- As mentioned, we have two main approaches to discrete-event modeling
  - we can focus on the actions to be performed when each event happens -> *event scheduling*
  - we consider all the actions of each entity while it is inside the system, focusing on its relations with other entities -> *process interaction*
- Given the approach, the simulators have some fundamental elements

Event scheduling: Simulator structure

- The basic elements of a discrete-event simulator are:
  - the **main cycle for event analysis (Event Loop)**
  - the **Future-Event Set (FES)**
  - the **functions to be executed when each event happens**
- To these key elements we can add:
  - the procedures for data collection and measurement analysis
  - the procedures for the initialization of the simulator
  - the termination criteria for the simulation

Queuing system (*event scheduling*)

- Arrival
  - Compute the inter-arrival time T_{ia} for next client
  - Schedule an arrival at time $T_{curr}+T_{ia}$
  - Create a record for the client
  - Insert the record in the queue
  - If the server is idle
    - Make the server busy
    - Determine the service time T_s
    - Schedule departure at time $T_{curr}+T_s$

Queuing system (*event scheduling*)

- Departure
  - Extract from the queue the record for the client who has just been served
  - Destroy the record
  - If someone is in the waiting line
    - Select a client to be served
    - Determine its service time T_s
    - Schedule the departure at time $T_{curr}+T_s$
  - else
    - Make the server idle

Structure of the simulator

- Fundamental terms
  - The **simulation time**
  - The **events** and their **attributes**
  - **State variables**

The simulation time

- The simulation time is a compressed/expanded representation of the time in the real system
- In the model, time can be
  - Continuous
  - Discrete
- We can represent time through
  - a floating-point variable
  - an integer variable

The simulation time

- Continuous time:
  - The time representation in floating-point format is the nearest one to the real nature of time
  - The probability for two events to happen exactly at the same time is zero
  - We do not need to worry in any special way about the management of contemporaneous events

The simulation time

- Discrete time:
  - The time representation in integer format is the most suited for those systems where time evolves naturally in steps (slotted systems)
  - The probability for more than one event to happen at the same discrete time may be not negligible
  - We need to define an order criterion among contemporaneous events

The simulation time

- Independently from the representation format, we need a global indicator of the time evolution: a system clock (*current time*)
- Must be accessible from any point of the simulator, representing the current value of the simulation time

Events: data structure

- Each event must be represented in our program using a structured data type, whose nature depends on the implementation choice for the Future-Event Set
- Each event contains at least:
  - the time at which the event is scheduled
  - an identifier for the type of event
    - In the example of the queue, two event types: “arrival” and “departure”

Events: data structure

- The event scheduling time is the key field, since it is needed to decide which is the next event to be executed, i.e., the one with the smallest scheduling time in the FES
- The event type is needed to identify which kind of event must happen, and, hence, which specific function should be executed

```
# the simulation time
time = 0
# the list of events in the form: (time, type)
FES = PriorityQueue()
# schedule the first arrival at t=0
FES.put((0, "arrival"))
```

Ordered list, based on the first attribute of the items (i.e., time in this case)

Events: data structure

- Often it is useful to associate attributes to each event
- This allows us to aggregate events according to their general type, to discriminate them later at execution time
- Example:
  - In a queuing system with two different kinds of server, instead of defining distinct events for the two servers, we define a single event “service” and we store the information on the involved server as an attribute

Event Loop

- The heart of a discrete-event simulator is the Event Loop
- It’s a loop in which:
  - the next event is extracted from the FES
  - the *current time* is advanced to the scheduling time of the event
  - depending on the event type, the associated procedure is executed

Event Loop: code

```
# the simulation time
time = 0
# the list of events in the form: (time, type)
FES = PriorityQueue()
# schedule the first arrival at t=0
FES.put((first_arrival_time, "arrival"))
while time < SIM_TIME:
    (time, event_type) = FES.get()
    if event_type == "arrival":
            arrival(time, FES, queue)
    elif event_type == "departure":
            departure(time, FES, queue)
```

Event Loop: termination

- The Event Loop is executed until the termination condition is reached
- Some possibilities
  - The FES is empty: the system evolution ended naturally
  - The maximum simulation time has been reached
  - The maximum number of events has been reached
  - The precision conditions on the measurements have been satisfied
  - An anomalous condition has been reached

Future-Event Set

- The Future-Event Set is the key structure of a discrete-event simulator
- The efficiency of the simulator depends on its implementation
- We need to consider the trade-off between implementation simplicity and execution efficiency
  - simple structures are often little efficient, while the most efficient may require a complex implementation

Future-Event Set

- The Future-Event Set is the set of all the events scheduled for future execution
- From this set we extract each time the one with the minimum scheduling time (Next Event)
- Any data structure can be used to represent the FES, but obviously some structures are more suitable and more efficient than others

Future-Event Set

- The selected data structure must support in the most efficient way the common operations for the FES
  - Insertion of a new event
  - Extraction of the next event
  - Lookup and deletion of an event
- The management efficiency for the first two operations is fundamental, while lookup and deletion may be required only for advanced cases and their implementation might be not optimized

Future-Event Set

- The management efficiency depends a lot on the specific conditions of the simulator
  - The sequence of the insertion and deletion operations
  - The average number of events in the FES
- We use as a reference some estimations of the efficiency in *worst-case* scenarios and order-of-magnitude evaluations

Example: queue

```
# state variable: no. of clients in the queue
users = 0


# *****************************************
# Client
# *****************************************
class Client:
    def __init__(self,type,arrival_time):
        self.type = type
        self.arrival_time = arrival_time
```

Event functions: arrival

```
def arrival(time, FES, queue):
    global users
    # sample the time until the next event
    inter_arrival = random.expovariate(1.0/ARRIVAL)
    # schedule the next arrival
    FES.put((time + inter_arrival, "arrival"))
    # Now manage the event;  update state variable
    users += 1
    # create a record for the client
    client = Client(TYPE1,time)
    # insert the record in the queue
    queue.append(client)
    # if the server is idle start the service
    if users==1:
        # sample the service time
        service_time = random.expovariate(1.0/SERVICE)
        # schedule when the client will finish the server
        FES.put((time + service_time, "departure"))
```

Event functions: departure

```
def departure(time, FES, queue):
    global users

    # get the first element from the queue
    client = queue.pop()

    # Update state variable
    users -= 1

    # see whether there are more clients to in the line
    if users >0:
        # sample the service time
        service_time = random.expovariate(1.0/SERVICE)

        # schedule when the client will finish the server
        FES.put((time + service_time, "departure"))
```

Wrap-up

- Fundamental elements of a discrete-event simulator:
  - Future-Event Set: whose management is fundamental for the efficiency of the simulation
  - Event loop: to make time advance and events to be executed in the proper order
  - Functions for each event type: to execute the events
- Variables:
  - Current time
  - State variables
  - Indicators of the performance measures