Simulation

Discrete event systems

Discrete event simulation 1

- Consider systems with finitely many components.
- Each component has only finitely many states.
- Components interact through events.
- Event takes place at a particular time (it has no duration).

Discrete event simulation 2

- Event can change states, generate other events (for the same or later time).
- Typical structural components
  - ”machine resources” (busy/free)
  - ”human resources” (busy/free)
  - ”raw materials” (availability/quantity)
  - ”products” (stage of production/availability)
- Events are beginnings and endings of actions

Wash machine

- Components of car wash
  - Wash machine (free/busy)
  - Queuing space (M available slots)
  - Clients (unwashed/washed)
- Events
  - Client arrival/departure
  - Wash start/end
  - Entering/leaving the queue
- Some events occur always together

Main simulation functionalities 1

- Simulation software has 5 main functionalities
  - Description of model structure
    - System parts -> state variables
    - Interaction logic -> ”flow chart”
    - Event logic-> ”code”
  - Random processes
    - Random numbers from desired distribution
  - Collecting and reporting statistics
    - Visualisation, confidence intervals, analysis

Main simulation functionalities 2

- Time management
  - Advancing the clock event by event
  - Activating events in right order
- Management of simulation experiment
  - Starting/ending simulation
  - Adding/removing events
  - Controlled replication of experiments

Main simulation functionalities 3

- Some functions are common to all models and experiments
  - Time management
  - Random numbers
  - Data collection and reporting
- Some are model and case dependent
  - Model structure and logic
  - Control flow in (series of) experiment(s)

Simulation paradigms

- Three approaches to simulation
  - Event based
    - State changes linked to certain time
  - Process based
    - Life cycle of events related to a system component.
  - Activity based
    - Activities that tie up resources of system components
- These lead to different model/code structures
  - Fit to different modelling situations

Event based simulation

- Event routines have central role
  - One routine for each type of event
  - Model logic is in the event routines
  - Event routine can change state variables and create event notices.
  - Scheduler manages event notices (time, event)
- One routine at a time is active.

Process/object based s.

- Subprocesses as objects with own state variables and event routines.
    - All actions related to a system component are within a single object
- Specific methods to communicate with sceduler and other objects.
    - No separate event notices
- Several processes active simultaneously (threads, coroutines).

Activity based s.

- Logic within activity routines
  - Each routine is linked to some resource
  - Two interfaces
    - Activation (if conditions are true, then reserve the resource and fix ending time)
    - Passivation: free the resource at given time
- All activities are scanned systematically
    - If conditions are true, routine is activated .
    - If no routine activates, time is incremented to next known ending time.

Simulation

Event based simulation

Event based simulation

- Historically oldest approach
- Logic is within sequentially executed routines
  - Easy to implement with any procedural language
  - Logic gets easily fragmented
    - Successive/dependent events are in separate routines

Wash machine (event b.)

- At least two types of events (arrival and departure (see introduction))
  - Both events can reserve the machine and generate departure
  - Potential maintainability problem
- Use 4 atomic events
    - Arrival (generates the client)
    - Start (reserve the resource and start service)
    - End (end service, free resource)
    - Departure (exits the client)

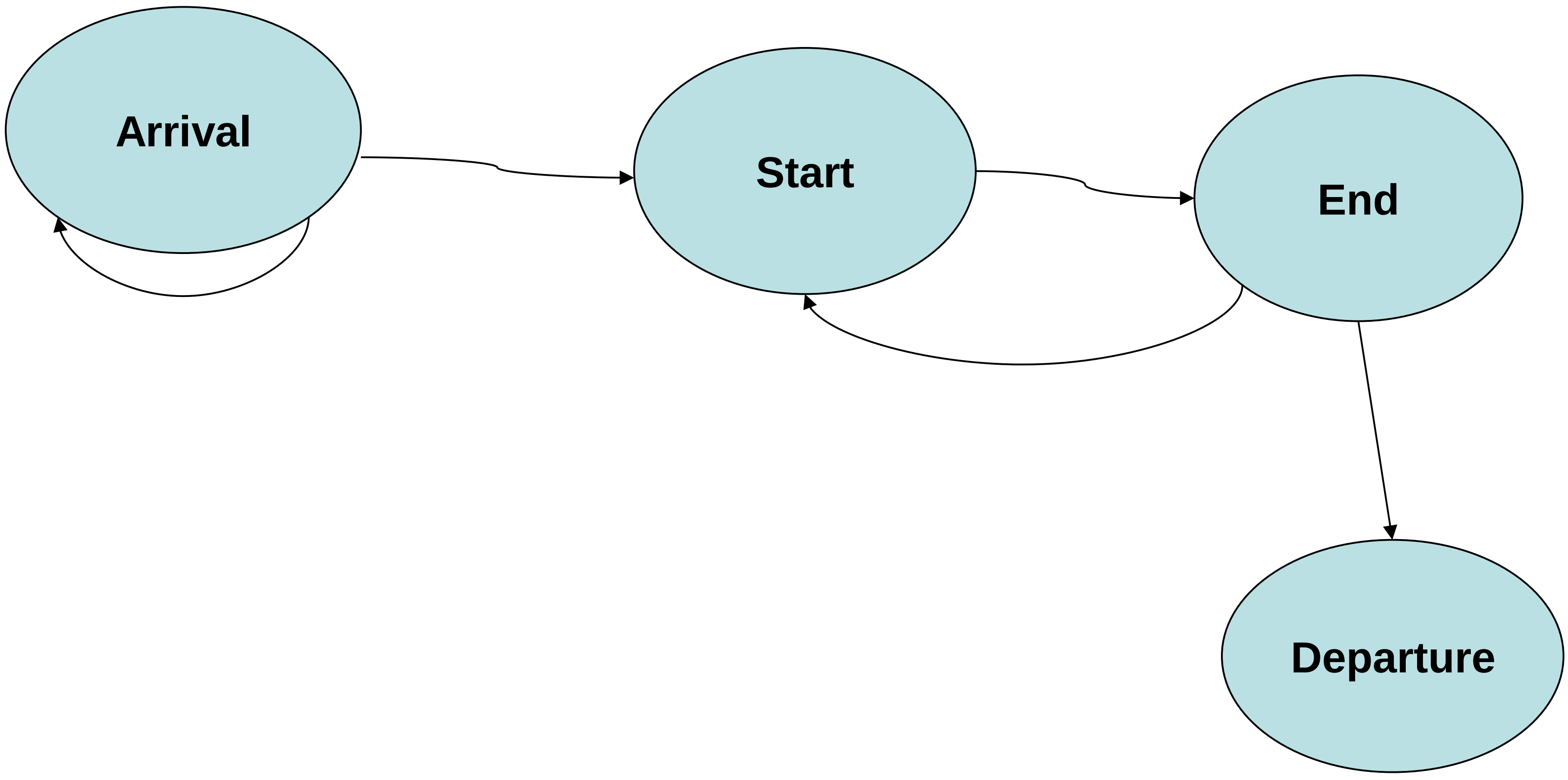
Wash machine 2

- Arrival
  - If queue not full
    - Create new client and put to the queue
    - Create a Start-event
  - Create new Arrival event (for later time)
- Start
  - If machine is free and clients in the queue
    - Take client from queue
    - Set machine busy
    - Crate an End-event (for later time)

Wash machine 3

- End
  - Set machine free
  - Create Departure-event (for same time)
  - Crate Start-event (for same time)
- Departure
  - Collect needed information from the client (if any)
  - Remove client

Wash machine



Wash machine - implementation

- 4 event (sub)routines
- For events `EventType` (Arrival, Start,End, Departure)
- For bookkeeping `EventNotice(Time, Event)`
- Event list to keep `EventNotice`
  - Methods
    - `NextEvent`
    - `AddEvent (Time, Event)`
    - `(RemoveEvent (Event))`
- Queue
  - Contains instances of `Client` –type
  - Methods `Add, Next, Length`
  - Serves Start-event
  - Another queue (or like) is needed for Departure

Wash machine - main

```
Initialize
T=0;
AddEvent(ArrivalTimeDistribution(),Arrival);
While (T< TMax) \\ (ending condition)
  Notice=NextEvent();
  T=Notice.Time;
  Type=Notice.Event;
  CASE Type of
    …
    \\ call for corresponding event routine
  END CASE
End While
```

Arrival

```
Arrival_Event()
  ClientTypePointer :: Car
  {
  AddEvent(ArrivalTimeDisribution(),Arrival);
  If Queue.Legth() < M then
    Car= New Client();
    Queue.Add(Car)
    AddEvent(0.,Start)
  EndIf
  }
```

Start

```
Start_Event()
  ClientTypePointer :: Car
  {
  If(Machine.Free() and Queue.Length()>0) then
      Car=Queue.Next();
      Machine.Reserve(Car);
      AddEvent(ServiceTimeDistribution(),End)
  Endif

  }
```

End

```
End_Event()
  ClientTypePointer :: Car
  {
  Car= Machine.Free()
  Departure.Reserve(Car) \\ To keep track of
  the client pointer
  AddEvent(0.,Departure)
  AddEvent(0.,Start)
  }
```

Departure

```
Departure_Event()
  ClientTypePointer :: Car
  {
   Car=Departure.Free()
  // Collect statistics
  RemoveClient(Car)
   }
// Reserve-Free for departure is a hack to keep the
   client pointer in absence of a real queue.
```

Observations

- Different queuing strategies can be hidden within Queue
- In case of several services, routing, client types etc requires replication or parametrization of events.
- In practice the service and its queue can be modeled as one entity where to the client is routed.

Simulation

Event based harbour network

Container harbours

- Main events
  - Ship i arrives to harbour j
    - Ship i to queue of harbour j at time t
    - Try to start loading (if queue empty)
  - Loading begins at dock
    - Ship i from queue, dock k reserved, loading end event for time t2

Container harbours

- Main events
  - Unloading of the ship ends
    - Dock k becomes free at t3
    - Try to start loading (if ships in queue)
  - Ship leaves for the next harbour
    - Ship i is scheduled to arrive to harbour j’ at t4

Questions ?

- Main events
  - Ship i arrives to harbour j
    - Ship i enters the queue of j at time t
    - What information is contained in the event notice. How the rest is communicated.
  - Unloading begins
    - Ship i taken from the queue, dock k reserved, end unloading –event for time t1
    - Is reference to dock k needed, where to keep link to ship i

Questions?

- Main events
  - Unloading ends
    - Dock k becomes free at time t3
    - Where is knowledge about the dock, about the ship
  - Ship leaves for next harbour
    - Arrival of ship i to harbour j' is scheduled at time t4
    - Who knows the value of j' for ship i

Event notices

- For traditional languages event notices are problematic
  - Static data types
  - Limited amount of information can be communicated
- Use of objects and inheritance helps
  - Inherited notice class for each type of event