

Object Oriented Simulation

Single-Server Systems

José M. Garrido C.

Department of Computer Science
Kennesaw State University

Spring, 2017

General Form of a Model

General steps

- 1 Define all classes for the active objects (processes), these classes inherit from the library class *Process*.
- 2 Define the main class, which is also a class for an active object of the model.
- 3 Include method *main* in the main class.

Class for active objects

All class definitions for active objects must include a method called *Main_body*. This method is the sequence of execution for the active object (process).

General Form of a Model

General steps

- 1 Define all classes for the active objects (processes), these classes inherit from the library class *Process*.
- 2 Define the main class, which is also a class for an active object of the model.
- 3 Include method *main* in the main class.

Class for active objects

All class definitions for active objects must include a method called *Main_body*. This method is the sequence of execution for the active object (process).

The Main Class

- 1 Declare variable(s) for the queue size
- 2 Declare the workload parameters and system parameters
- 3 Declare variables for the total calculations
- 4 Declare a reference to an object of class *Simulation*
- 5 Declare references for the active objects of model
- 6 Declare references to queues and other passive objects
- 7 Declare references for the output files
- 8 Define the *initializer* method
- 9 Define method *main*
- 10 Define the main body of the class

Method *main*

- 1 Create an object of the library class *Simulation* using a title for the model.
- 2 Create all the objects used in the model.
- 3 Start the simulation by invoking the *start_sim* function of the simulation object.
- 4 Setup the output files.
- 5 Create the queues and other passive objects.
- 6 Create and start an object of this class, the main object of the model.

Method *Main_body*

- 1 Create and start the active objects declared in this class.
- 2 Start the simulation run using the declared simulation period.
- 3 Calculate total results.

Illustration of a Single Server System

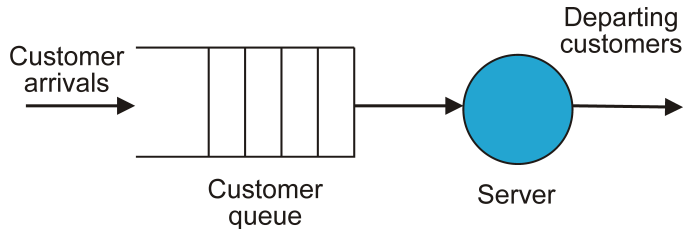


Figure: Model of a single-server system

Single Server System

In a model of a single-server system, there is only one server process that provides service to the customer processes.

Components

- The arrival of customers, indicated with an arrow pointing to the tail of the queue
- The customers that arrive requesting service
- The customer queue, which is the waiting line for customers
- The server, which removes the next customer from the queue and services it

The Queue

- A queue is a data structure that represents a waiting line and has two ends, the *head* and the *tail*.
- A queue is defined as a passive entity.
- An arriving customer process enters the queue at its tail.
- A customer can only leave the queue if it is at the head.
- The server removes the customer at the head of the queue then provides service to this customer.
- The order of arrival of customers is retained in a simple queue, which follows first-in-first-out (FIFO) ordering.

Illustration of a Queue

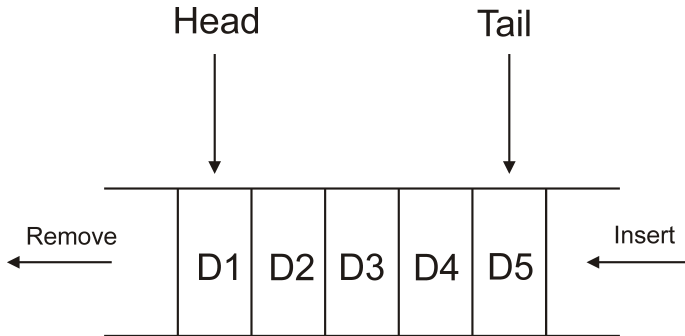


Figure: Model of a queue

The Car-Wash System

- Arriving cars join a queue to wait for service.
- The car at the head of the queue is the one that is next serviced by the Carwash machine.
- After the car wash is completed, the car leaves the system.
- Cars are considered the customers of the system, as they are the processes requesting service from the server.

Carwash Model View

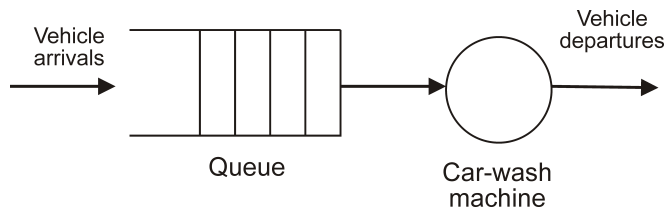


Figure: Graphical view of the Carwash system

The Conceptual Model

Processes

- The *Car* processes, which represent the customers to be serviced by the Carwash machine. There are many instances of this process that are created and actually executed (each one is an object of class *Car*).
- The *Arrivals* process, which creates car arrivals according to the inter-arrival period. This process represents the environment.
- The *Machine* process, which represents the server process that services the car processes, according to the service demand of car processes.

The UML Class Diagram of the Model

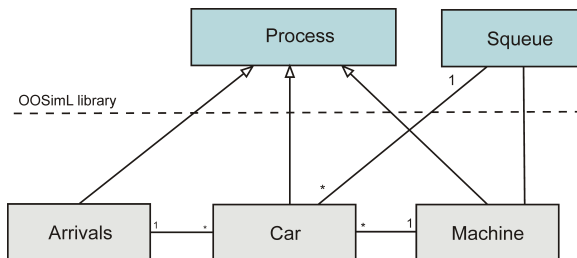


Figure: The UML class diagram for the Carwash system

The Events in the Model

- 1 The arrivals of car processes, which have to join the queue.
- 2 The start of service of a car
- 3 The completion of service of the current car process. At this same instance, the server process attempts to get the next car process from the queue, and if there is none, will transition to the idle state.

The UML Communication Diagram

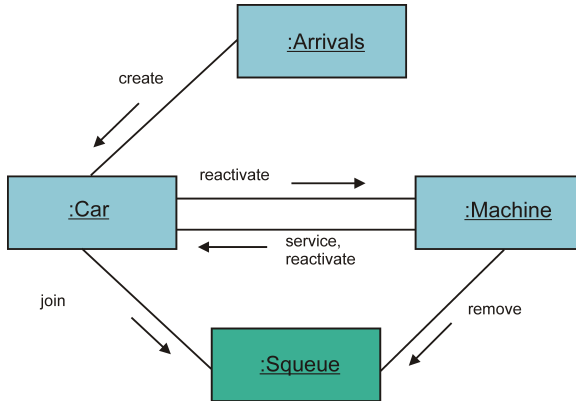


Figure: The UML communication diagram for the Carwash system

The UML State Diagram

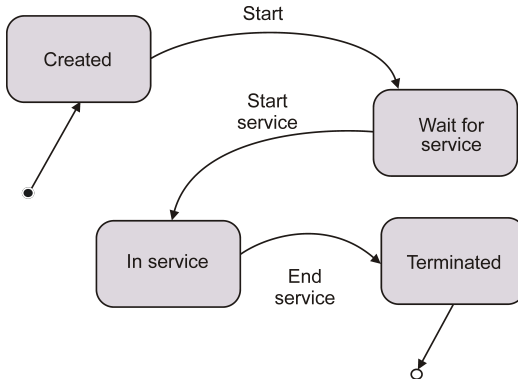


Figure: A state diagram for a Car process

The Car-Wash Deterministic Model

A deterministic version of the model for the Carwash system has predetermined arrivals and service periods.

Workload

- 1 Customer processes arrive at time instants: 0, 2, 5, 8, 12, 13, 17, 22, and 23
- 2 A Carwash service takes 4 time units

The Arrivals Process

The arrivals process represents the environment that generates new customers. There is a single arrivals process in this model

Sequence of Activities

- 1 Wait until the clock advances to the next arrival event, i.e., by a time interval equal to the inter-arrival period.
- 2 Create a customer (car) object with service period equal to 4.0 time units.
- 3 Repeat from step 1 for all arriving car processes.

The Server Process

There is only one object of the class *Server* in this model

Sequence of Activities

- ❶ While the arrival queue is not empty, execute the following activities:
 - ❶ Remove the car process at the head of the queue.
 - ❷ Carry out the service on the car process.
 - ❸ Reactivate the car process.
- ❷ Wait (in the idle state) until the next car process arrives, then start from step 1.

The Car Process

A car process includes the service period as one of its relevant attributes

Sequence of Activities

- 1 Join the arrival queue if it is not full. Terminate if the queue is full (i.e., the car is rejected).
- 2 Reactivate the server if the server is idle.
- 3 Suspend self.
- 4 Continue with own activities when the server process has completed the service.
- 5 Terminate.

Table of Results

Table: Events in the Car-Wash Model

Time	Process	Event	Next Event
0	arrivals	starts	4
0	server	starts	
0	car 1	arrives	
0	server	begins service	
2	car 2	arrives	4
4	server	completes service	
4	server	begins service	8
5	car 3	arrives	8
8	car 4	arrives	
8	server	completes service	

Table of Results (2)

Table: Events in the Car-Wash Model

Time	Process	Event	Next Event
8	server	begins service	12
12	car 5	arrives	
12	server	completes service	12
12	server	begins service	16
13	car 6	arrives	
16	server	completes service	16
16	server	begins service	20
20	server	completes service	20
20	server	begins service	24
24	server	completes service	24
24	server	starts waiting	

Number of Cars in the Queue

Table: Queue Size in the Car-Wash Model

Time	Q Size
0	0
2	1
4	0
5	1
8	1
12	1
13	2
16	1
20	0

Wait Intervals of Cars

Table: Processes Waiting Time

Process	Arrival	Start Service	Wait Time
car 1	0	0	0
car 2	2	4	2
car 3	5	8	3
car 4	8	12	4
car 5	12	16	4
car 6	13	20	7

The Car-Wash Stochastic Model

Differences in deterministic and stochastic models

- The arrival events are random events, i.e., it is not known exactly when a car arrival event will occur.
- The service periods are also random, i.e., it is not known exactly how much time it will take the machine to wash a particular car.

Random variables in the carwash model

- the inter-arrival period for each car process, and
- the service period for each car process.

Output of Simulation Runs

The results of a simulation run of the stochastic Carwash model is produced on the two output files generated by the simulation run:

- `carwtrace.txt`
- `carwstatf.txt`

One file includes a listing of the sequence of events and the other file includes the summary statistics and performance metrics computed.

Partial Output of a Simulation Run

```
End Simulation of Simple Model of Car-Wash System  
date: 11/16/2016 time: 12:37  
Elapsed computer time: 54 millisec
```

```
Total number of cars serviced: 46  
Car average wait period: 90.844  
Average period car spends in the shop: 87.152  
Machine utilization: 0.898
```

Diagrams

These diagrams help describe the operations of the processes involved and the synchronization that is present when processes interact with each other.

Car processes enter a queue and transition to the idle state. The machine process removes a car process from the queue to service it. All queues are passive objects and are different from the processes.

Diagrams

These diagrams help describe the operations of the processes involved and the synchronization that is present when processes interact with each other.

Car processes enter a queue and transition to the idle state. The machine process removes a car process from the queue to service it. All queues are passive objects and are different from the processes.

Activity Symbols

The flow of control in the diagram is composed of a sequence of symbols that denote different type of activities and synchronization actions, all joined by lines and arrows.

A general activity is represented by a rectangular box; the label in the box is the name of the activity. This activity symbol is usually called a *delay* because a finite interval elapses while the activity is carried out and before the process can proceed to the next activity. The arrows show the direction of the flow activity sequence.

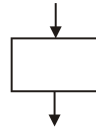
Illustration of Activity Symbols



Start



Suspend



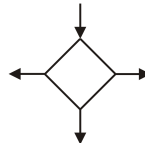
Gen activity



Terminate



Reactivate



Condition

Figure: Activity symbols

Activity Symbols

- The lower semicircle with a bar below a process name, indicates the start of a process life cycle.
- The lower semicircle symbol also indicates reactivation of the activity sequence; this symbol can also have an arrow in a non-vertical inclination that shows synchronization with another process.
- The inverted form of this symbol indicates the termination of the process.
- The upper semicircle indicates that the object enters its idle state
- a possible arrow in a non vertical inclination denotes the synchronization with another process.

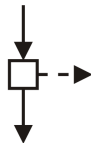
Interaction Symbols

- Process interaction symbols indicate signals and direct synchronization among processes.
- The *synchronization send* denotes when a signal is sent to a condition queue.
- This symbol can also denote an interrupt to another process.
- This symbol can also be used to send an item to a queue.
- The *synchronization receive* indicates an item being received from a queue.

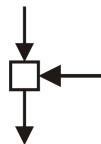
Interaction Symbols (2)

- The symbols for *cooperate* describe the effect of the cooperation among processes.
- The symbol for *schedule* describe a process that schedules itself or another process after some time interval
- This symbol can also denote the reactivation of another process.

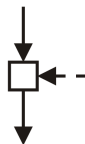
Illustrations of Interaction Symbols



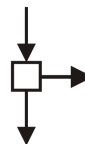
Send



Cooperate



Receive



Schedule

Object Oriented Simulation

