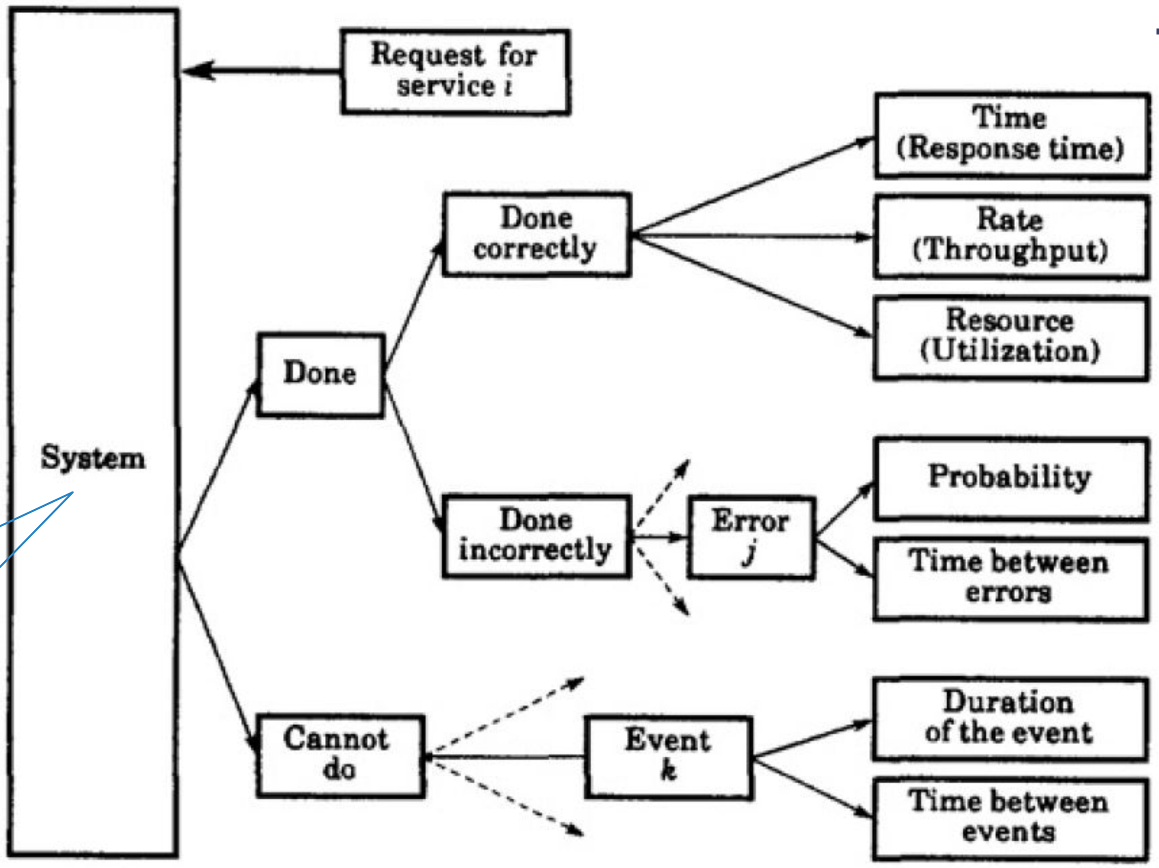
Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 25 : BUILDING A PERFORMANCE MODEL 2 – SIMULATION

Recall: Very Basics for Dependability Evaluation



Note that it is assumed that the system is available for measurements!

Collect measures

When analytical models are not enough?

Many systems are highly complex

_> **Valid mathematical models of them are themselves complex, precluding any possibility of an analytical solution** (like the ones we've seen for M/M/1 or Jackson's Networks)

In this case, **the model must be studied by means of simulation**, i.e., **numerically exercising the model for the inputs in question to see how they affect the output measures of performance**

= Model the main behaviours of the system through a computer program

System Simulation Intro

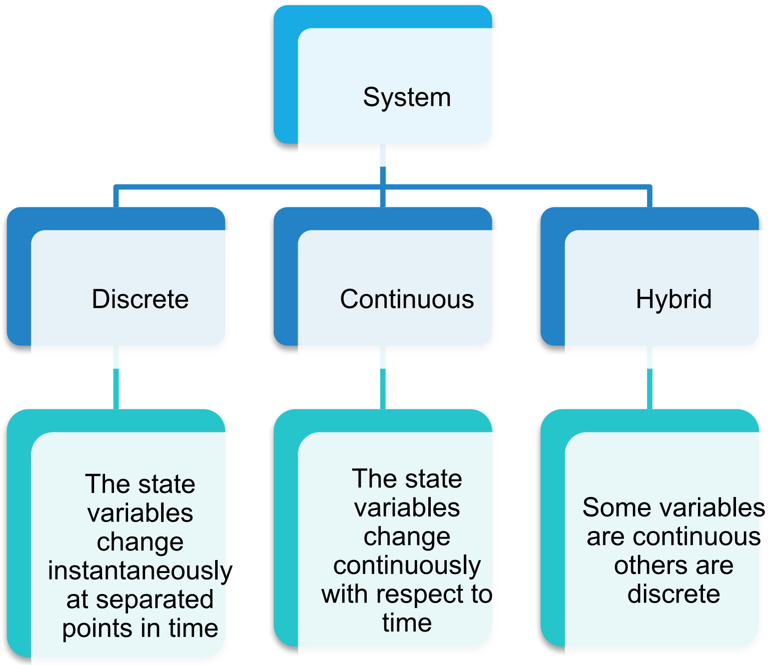
A **system** is defined to be a collection of entities that act and interact together toward the accomplishment of some logical end.

We define the **state of a system** to be **that collection of variables necessary to describe a system at a particular time**, relative to the objectives of a study.

We categorize systems to be of two types, *discrete and continuous*.

A *discrete* system is one for which the state variables change instantaneously at separated points in time.

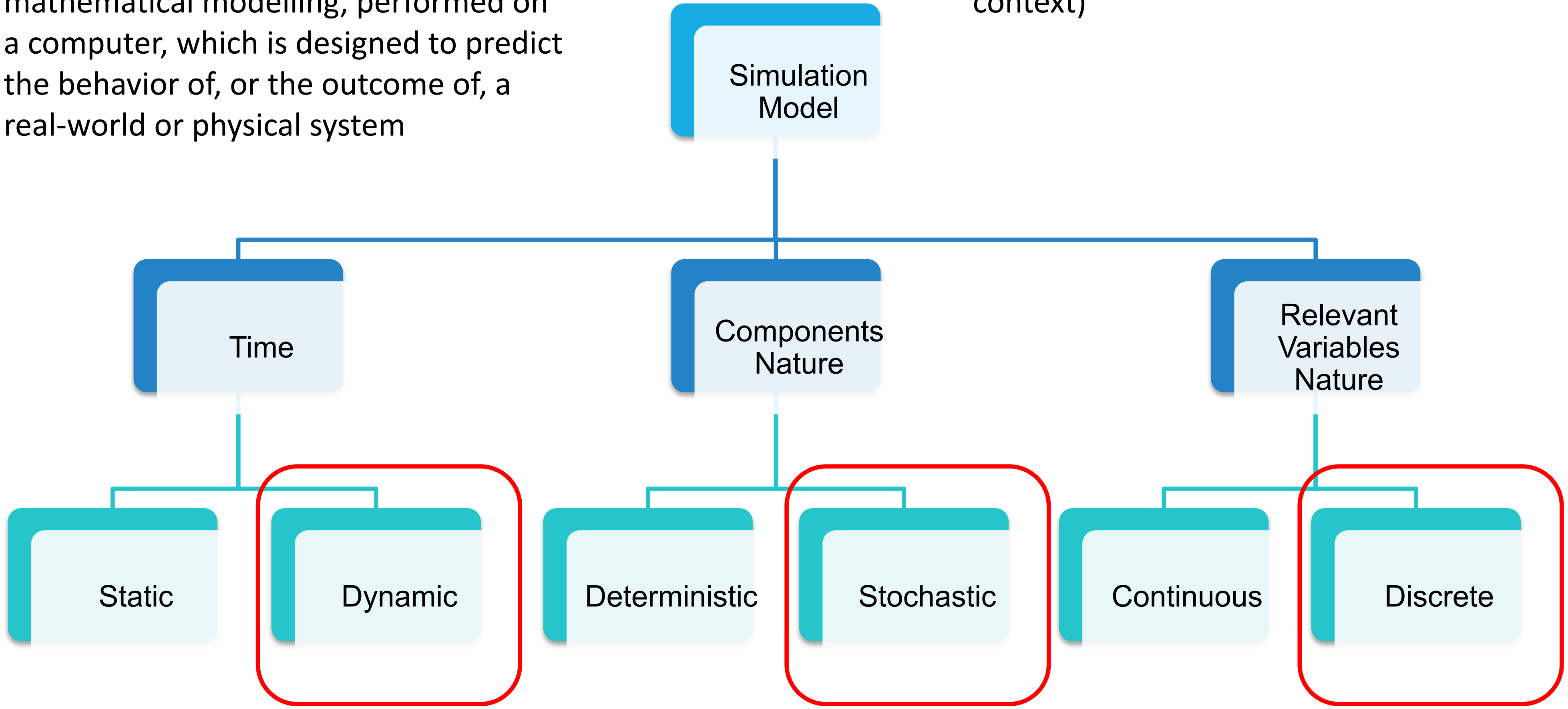
A *continuous* system is one for which the state **variables change continuously** with respect to time.


System
Discrete
Continuous
Hybrid
The state variables change instantaneously at separated points in time
The state variables change continuously with respect to time
Some variables are continuous others are discrete

Simulation Model

Computer simulation is the process of mathematical modelling, performed on a computer, which is designed to predict the behavior of, or the outcome of, a real-world or physical system

Simulation = Computer programs (in our context)

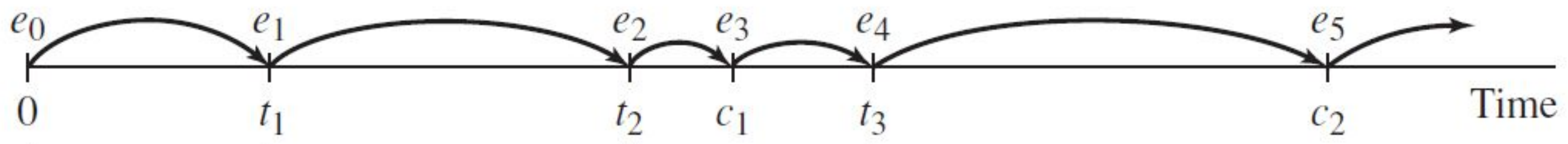


Discrete-Event Simulation

Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously **at separate points in time** (In more mathematical terms, we might say that the system can change at only a countable number of points in time)

These points in time are the ones at which an event occurs, **where an event is defined as an instantaneous occurrence that may change the state of the system**

Between consecutive events, no change in the system is assumed to occur



Discrete-Event Simulation

We must keep track of the current value of **simulated time** as the simulation proceeds, and **we also need a mechanism to advance simulated time from one value to another**

simulation clock is the variable in a simulation model that gives the current value of simulated time

Two principal approaches have been suggested for advancing the simulation clock

- **next-event time advance**: simulation time can directly jump to the occurrence time of the next event
- **fixed-increment time advance**: where time is broken up into small time slices and the system state is updated according to the set of events/activities happening in the time slice

There is generally no relationship between simulated time and the time needed to run a simulation on the computer

Components and Organization of a Discrete-Event Simulation Model

System state	The collection of state variables necessary to describe the system at a particular time
Simulation clock	A variable giving the current value of simulated time
Event list	A list containing the next time when each type of event will occur
Statistical counters	Variables used for storing statistical information about system performance
Initialization routine	A subprogram to initialize the simulation model at time 0
Timing routine	A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur

Components and Organization of a Discrete-Event Simulation Model

Event routine	A subprogram that updates the system state when a particular type of event occurs (there is one event routine for each event type)
Library routines	A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model
Report generator	A subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends
Main program	A subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.

How Simulation Evolves: next-event time progression

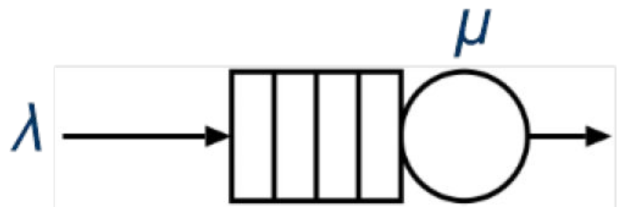
With the **next-event time-advance approach**, the simulation clock is initialized to zero and **the times of occurrence of future events are determined**.

The simulation clock is then advanced to the time of occurrence of the most imminent (first) of these future events, at which point the state of the system is updated to account for the fact that an event has occurred, **and our knowledge of the times of occurrence of future events is also updated**. Then the simulation clock is advanced to the time of the (new) most imminent event, the state of the system is updated, and future event times are determined, etc.

This process of advancing the simulation clock from one event time to another **continues until some prespecified stopping condition is satisfied**. Since all state changes occur only at event times for a discrete event simulation model, periods of inactivity are skipped over by jumping the clock from event time to event time.

Example: Simulation of a Single-server Queueing System

Consider a single-server queueing system for which the interarrival times A_1, A_2, . . . are independent and identically distributed (IID) random variables



The simulation will begin in the "empty-and-idle" state, i.e., no customers are present and the server is idle.

At time 0, we will begin waiting for the arrival of the first customer, which will occur after the first interarrival time A_1

The simulation will stop when the nth customer enters service

Example: Simulation of a Single-server Queueing System

To measure the performance of this system, we will look at:

- d(n): the expected average delay in queue of the n customers
- q(n): average number of customers in the queue (but not being served)
  - Note that indicating n is necessary in the notation to underline that this average is taken over the time period needed to observe the n delays defining our stopping rule.
- u(n): expected utilization of the server i.e., the expected proportion of time during the simulation that the server is busy (i.e., not idle).

Example: Simulation of a Single-server Queueing System

The **events** for this system are

- the **arrival** of a customer and
- the **departure** of a customer (after a service completion)

The **state variables** necessary to estimate d(n), q(n), and u(n) are

- the **status of the server** (0 for idle and 1 for busy)
- the **number of customers** in the queue,
- the **time of arrival of each customer** in the queue

Example: Simulation of a Single-server Queueing System

EXAMPLE

We begin our explanation of how to simulate a single-server queueing system by showing how its simulation model would be represented inside the computer at time $e_0 = 0$ and the times $e_1, e_2, \ldots, e_{13}$ at which the 13 successive events occur that are needed to observe the desired number (n = 6) of delays in queue.

All these times are computed by the simulation! How to compute them must be coded

	1	2	3	4	5	6	7	8	9	…
A_i	0,4	1,2	0,5	1,7	0,2	1,6	0,2	1,4	1,9	…

Inter-arrival times

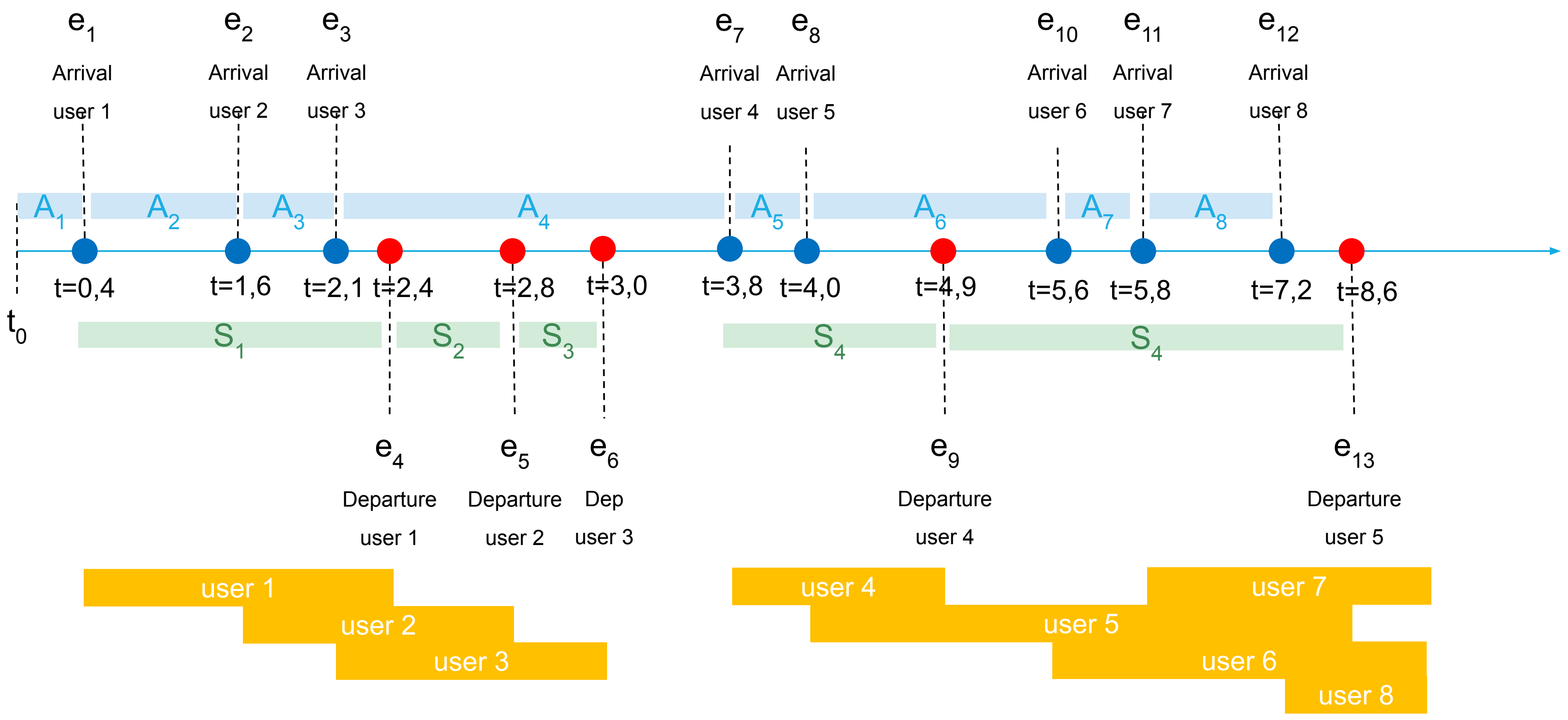
	1	2	3	4	5	6	7	8	9	…
S_i	2,0	0,7	0,2	1,1	3,7	0,6	…	…	…	…

Service times

WARNING!

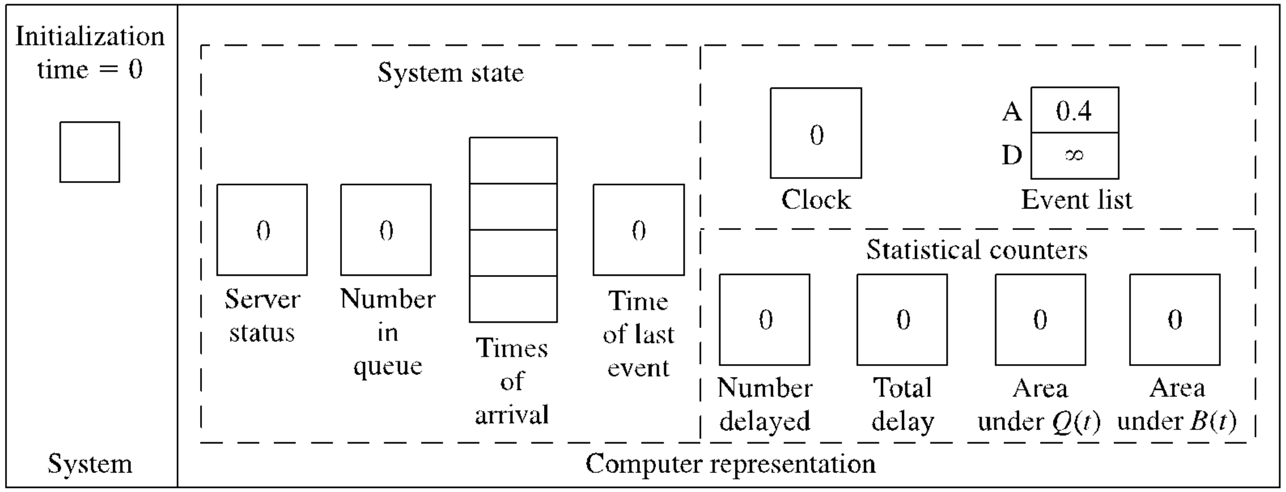
It is not necessary to declare what the time units are (minutes, hours, etc.), but only to be sure that all time quantities are expressed in the same units

Example: Simulation of a Single-server Queueing System



Example: Simulation of a Single-server Queueing System

	1	2	3	4	5	6	7	8	9	…
A_i	0,4	1,2	0,5	1,7	0,2	1,6	0,2	1,4	1,9	…

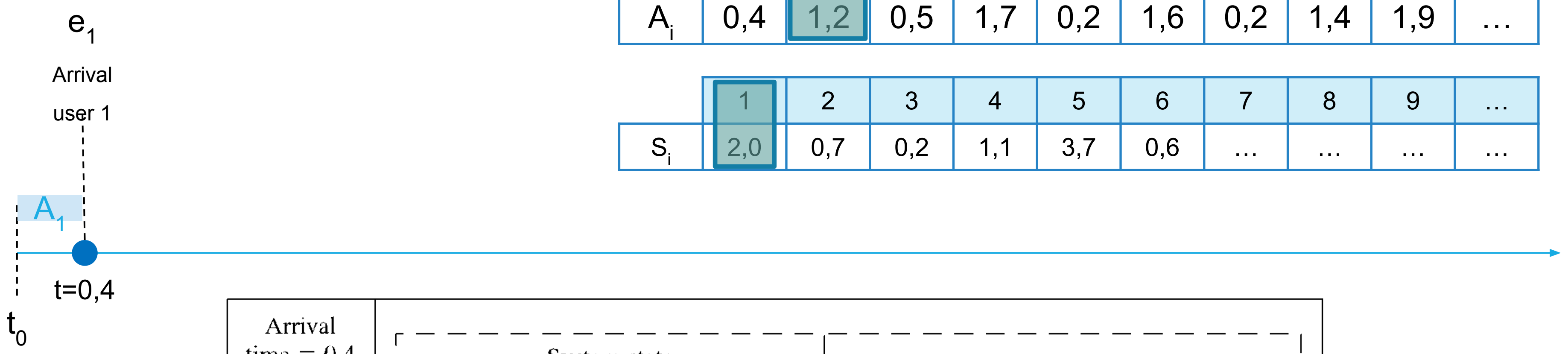


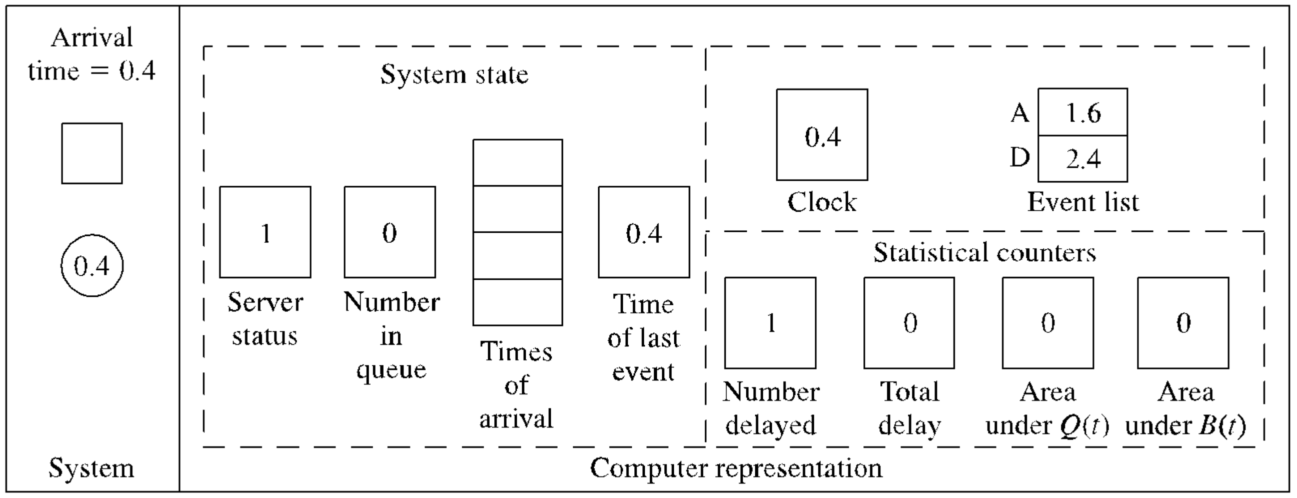
Example: Simulation of a Single-server Queueing System

EXAMPLE

	1	2	3	4	5	6	7	8	9	…
A_i	0,4	1,2	0,5	1,7	0,2	1,6	0,2	1,4	1,9	…

	1	2	3	4	5	6	7	8	9	…
S_i	2,0	0,7	0,2	1,1	3,7	0,6	…	…	…	…

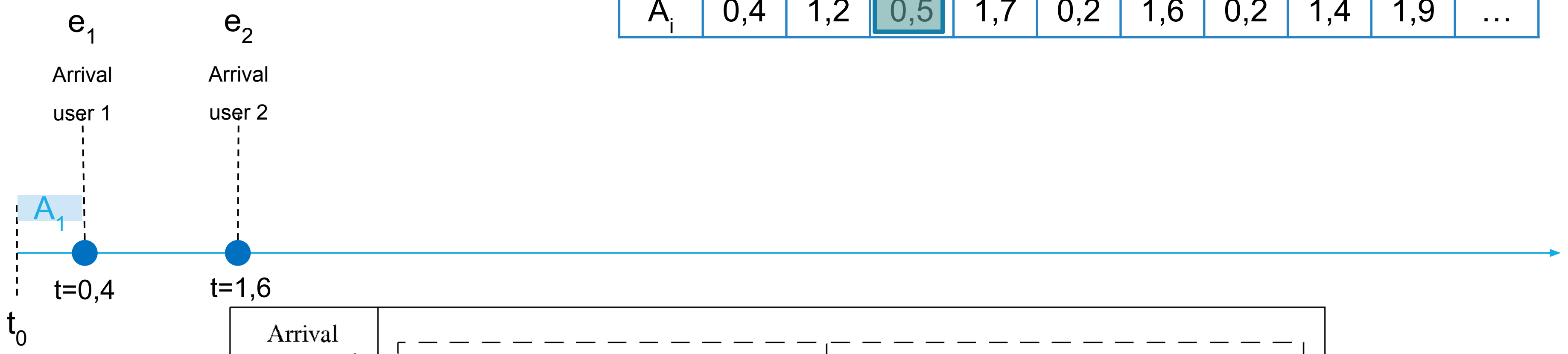


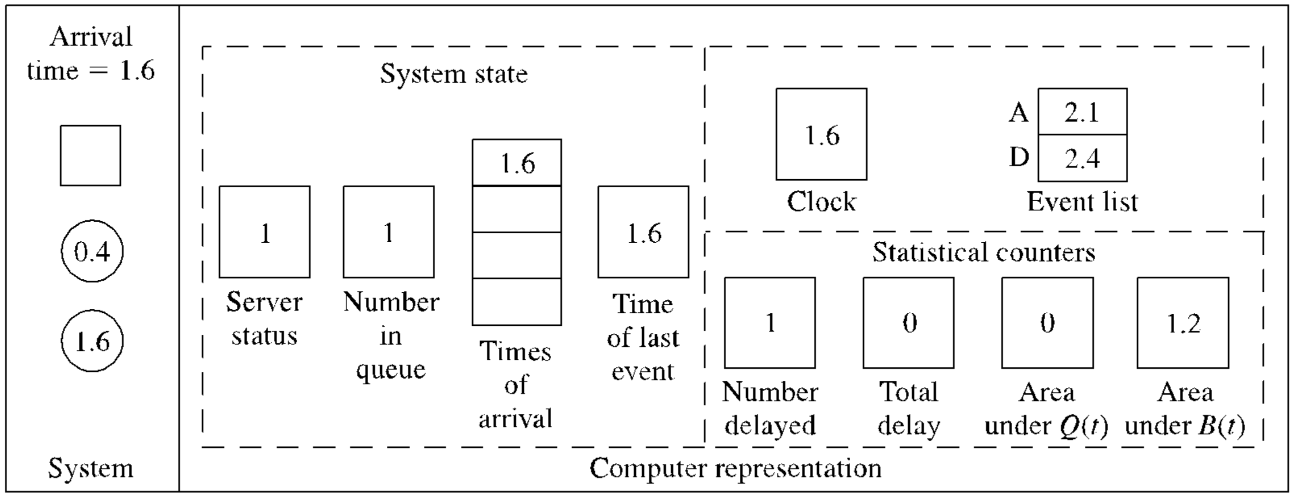


Example: Simulation of a Single-server Queueing System

EXAMPLE

	1	2	3	4	5	6	7	8	9	…
A_i	0,4	1,2	0,5	1,7	0,2	1,6	0,2	1,4	1,9	…

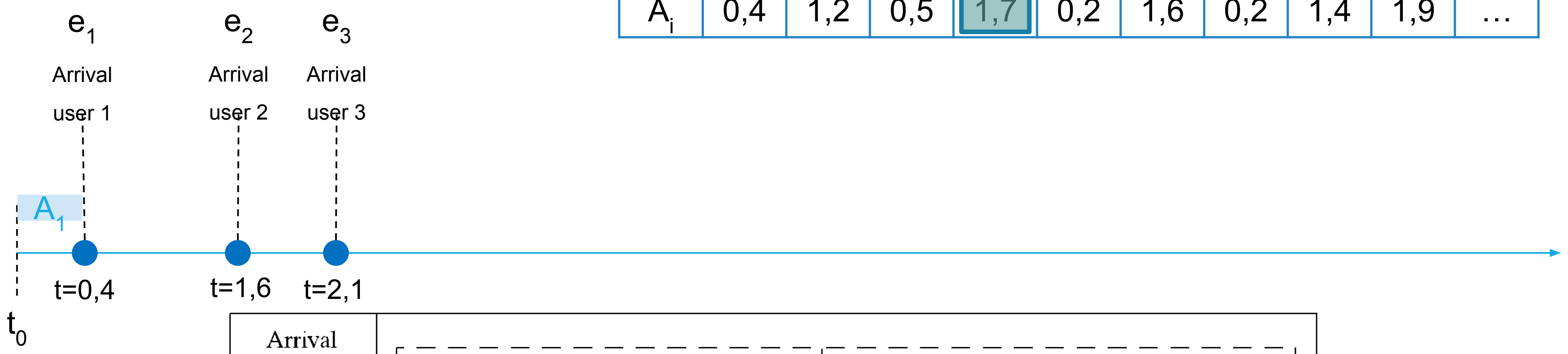


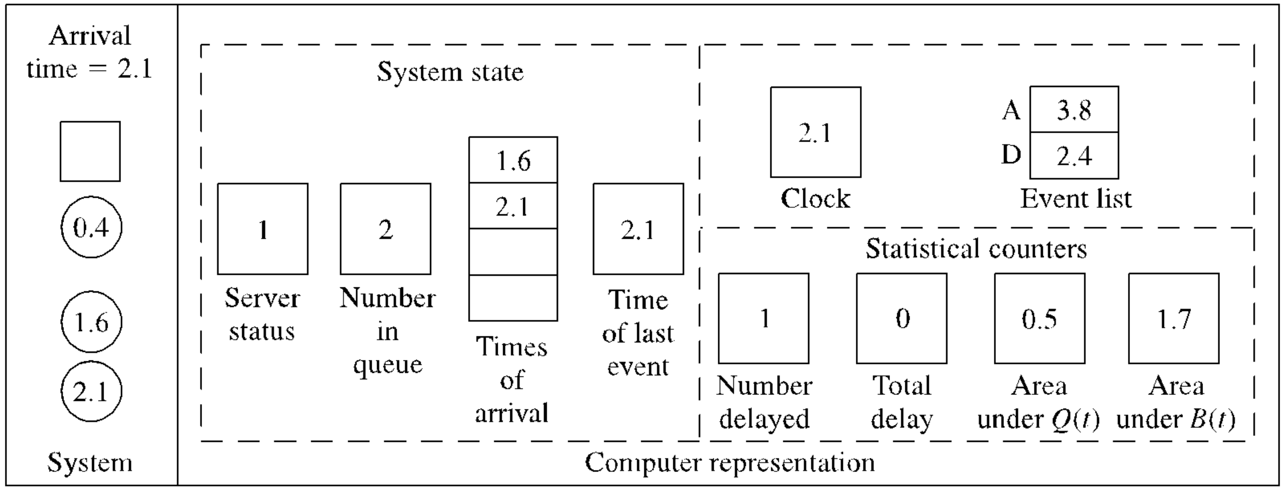


Example: Simulation of a Single-server Queueing System

EXAMPLE

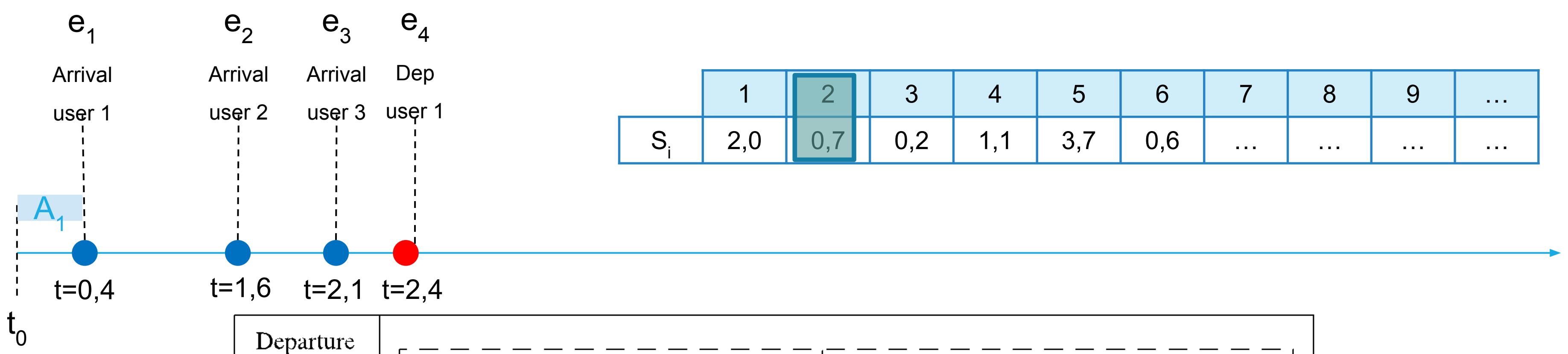
	1	2	3	4	5	6	7	8	9	…
A_i	0,4	1,2	0,5	1,7	0,2	1,6	0,2	1,4	1,9	…



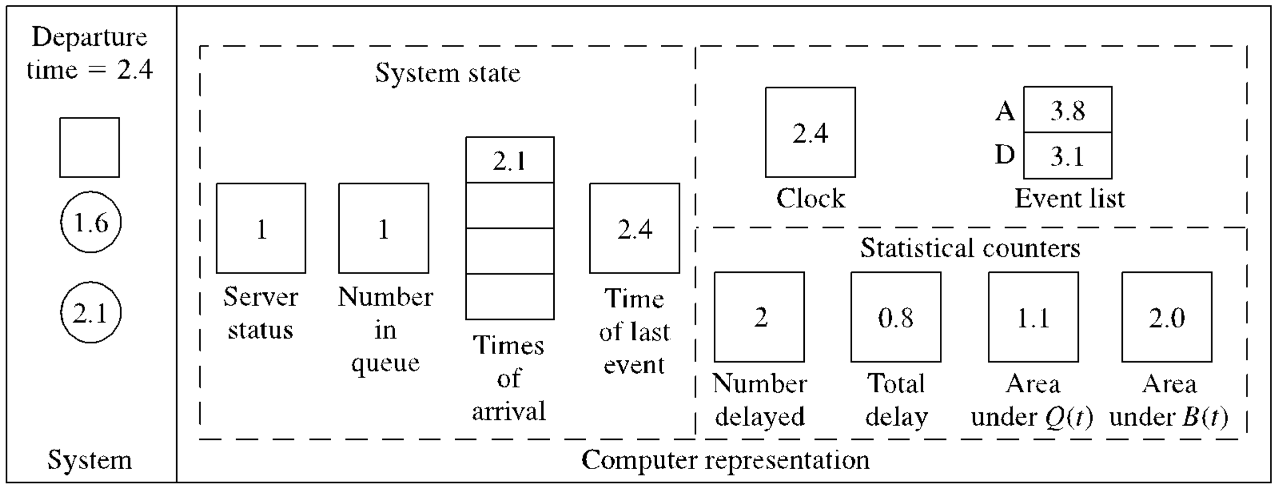


Example: Simulation of a Single-server Queueing System

EXAMPLE

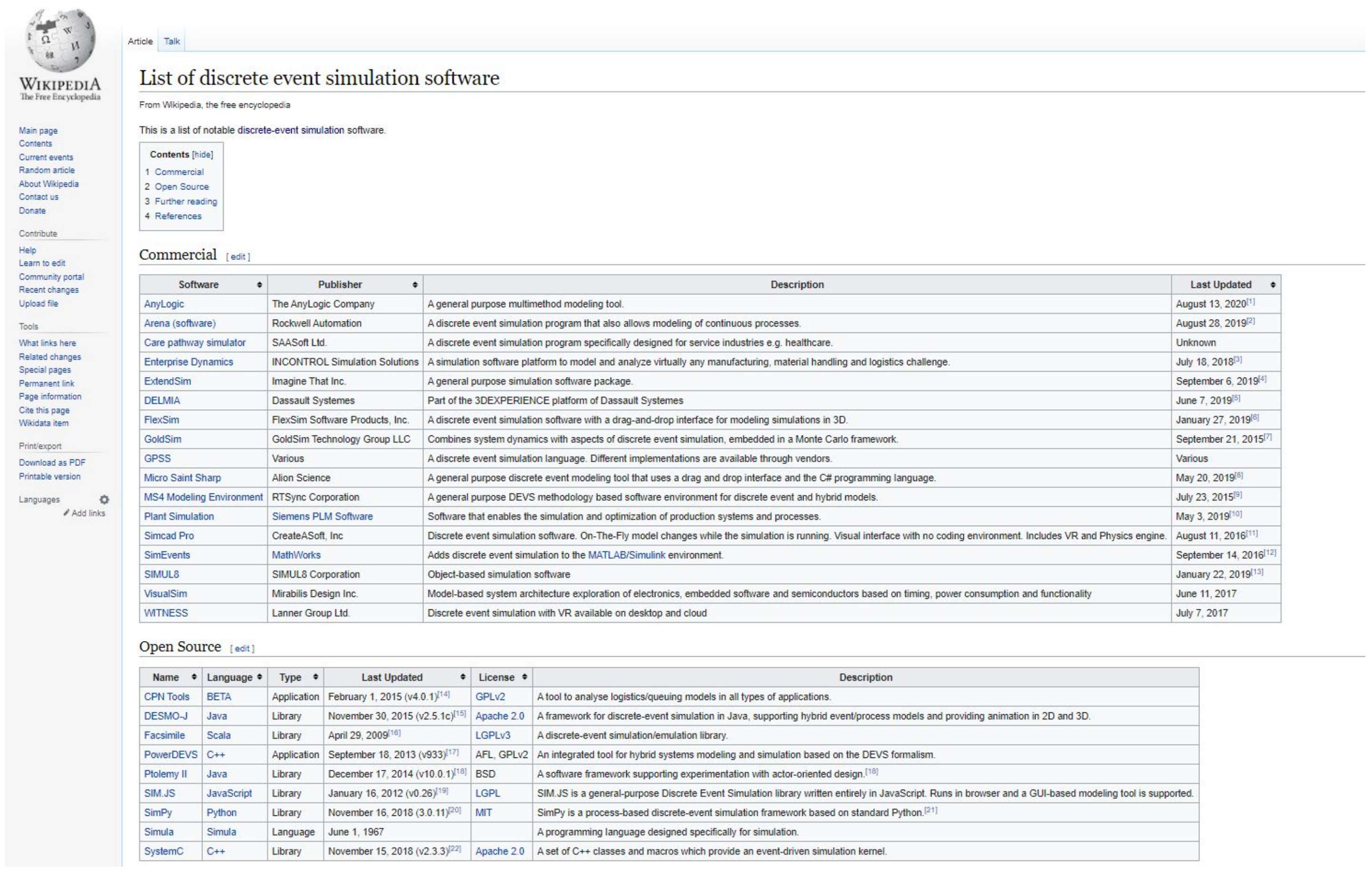


	1	2	3	4	5	6	7	8	9	…
S_i	2,0	0,7	0,2	1,1	3,7	0,6	…	…	…	…



Discrete Event Simulation Softwares

https://en.wikipedia.org/wiki/List_of_discrete_event_simulation_software



List of discrete event simulation software

From Wikipedia, the free encyclopedia

This is a list of notable discrete-event simulation software.

Contents [hide]
1 Commercial
2 Open Source
3 Further reading
4 References

Commercial [edit]

Software	Publisher	Description	Last Updated
AnyLogic	The AnyLogic Company	A general purpose multimethod modeling tool.	August 13, 2020[1]
Arena (software)	Rockwell Automation	A discrete event simulation program that also allows modeling of continuous processes.	August 28, 2019[2]
Care pathway simulator	SAASoft Ltd.	A discrete event simulation program specifically designed for service industries e.g. healthcare.	Unknown
Enterprise Dynamics	INCONTROL Simulation Solutions	A simulation software platform to model and analyze virtually any manufacturing, material handling and logistics challenge.	July 18, 2018[3]
ExtendSim	Imagine That Inc.	A general purpose simulation software package.	September 6, 2019[4]
DELMIA	Dassault Systemes	Part of the 3DEXPERIENCE platform of Dassault Systemes	June 7, 2019[5]
FlexSim	FlexSim Software Products, Inc.	A discrete event simulation software with a drag-and-drop interface for modeling simulations in 3D.	January 27, 2019[6]
GoldSim	GoldSim Technology Group LLC	Combines system dynamics with aspects of discrete event simulation, embedded in a Monte Carlo framework.	September 21, 2015[7]
GPSS	Various	A discrete event simulation language. Different implementations are available through vendors.	Various
Micro Saint Sharp	Alion Science	A general purpose discrete event modeling tool that uses a drag and drop interface and the C# programming language.	May 20, 2019[8]
MS4 Modeling Environment	RTSync Corporation	A general purpose DEVS methodology based software environment for discrete event and hybrid models.	July 23, 2015[9]
Plant Simulation	Siemens PLM Software	Software that enables the simulation and optimization of production systems and processes.	May 3, 2019[10]
Simcad Pro	CreateASoft, Inc	Discrete event simulation software. On-The-Fly model changes while the simulation is running. Visual interface with no coding environment. Includes VR and Physics engine.	August 11, 2016[11]
SimEvents	MathWorks	Adds discrete event simulation to the MATLAB/Simulink environment.	September 14, 2016[12]
SIMUL8	SIMUL8 Corporation	Object-based simulation software	January 22, 2019[13]
VisualSim	Mirabilis Design Inc.	Model-based system architecture exploration of electronics, embedded software and semiconductors based on timing, power consumption and functionality	June 11, 2017
WITNESS	Lanner Group Ltd.	Discrete event simulation with VR available on desktop and cloud	July 7, 2017

Open Source [edit]

Name	Language	Type	Last Updated	License	Description
CPN Tools	BETA	Application	February 1, 2015 (v4.0.1)[14]	GPLv2	A tool to analyse logistics/queuing models in all types of applications.
DESMO-J	Java	Library	November 30, 2015 (v2.5.1c)[15]	Apache 2.0	A framework for discrete-event simulation in Java, supporting hybrid event/process models and providing animation in 2D and 3D.
Facsimile	Scala	Library	April 29, 2009[16]	LGPLv3	A discrete-event simulation/emulation library.
PowerDEVS	C++	Application	September 18, 2013 (v933)[17]	AFL, GPLv2	An integrated tool for hybrid systems modeling and simulation based on the DEVS formalism.
Ptolemy II	Java	Library	December 17, 2014 (v10.0.1)[18]	BSD	A software framework supporting experimentation with actor-oriented design.[18]
SIM.JS	JavaScript	Library	January 16, 2012 (v0.26)[19]	LGPL	SIM.JS is a general-purpose Discrete Event Simulation library written entirely in JavaScript. Runs in browser and a GUI-based modeling tool is supported.
SimPy	Python	Library	November 16, 2018 (3.0.11)[20]	MIT	SimPy is a process-based discrete-event simulation framework based on standard Python.[21]
Simula	Simula	Language	June 1, 1967		A programming language designed specifically for simulation.
SystemC	C++	Library	November 15, 2018 (v2.3.3)[22]	Apache 2.0	A set of C++ classes and macros which provide an event-driven simulation kernel.

SimPy

The easiest to use! (Python)

https://simpy.readthedocs.io/en/latest/

General Discrete Event Simulator

You have to define the evolution of everything!

The behavior of active components (like vehicles, customers or messages) is modeled with ***processes***.

All processes live in an ***environment***.

They interact with the environment and with each other via ***events***

SimPy

Processes are described by simple **Python generators**.

Generator functions are a special kind of function that return a lazy iterator. These are objects that you can loop over like a list. However, unlike lists, lazy iterators do not store their contents in memory.

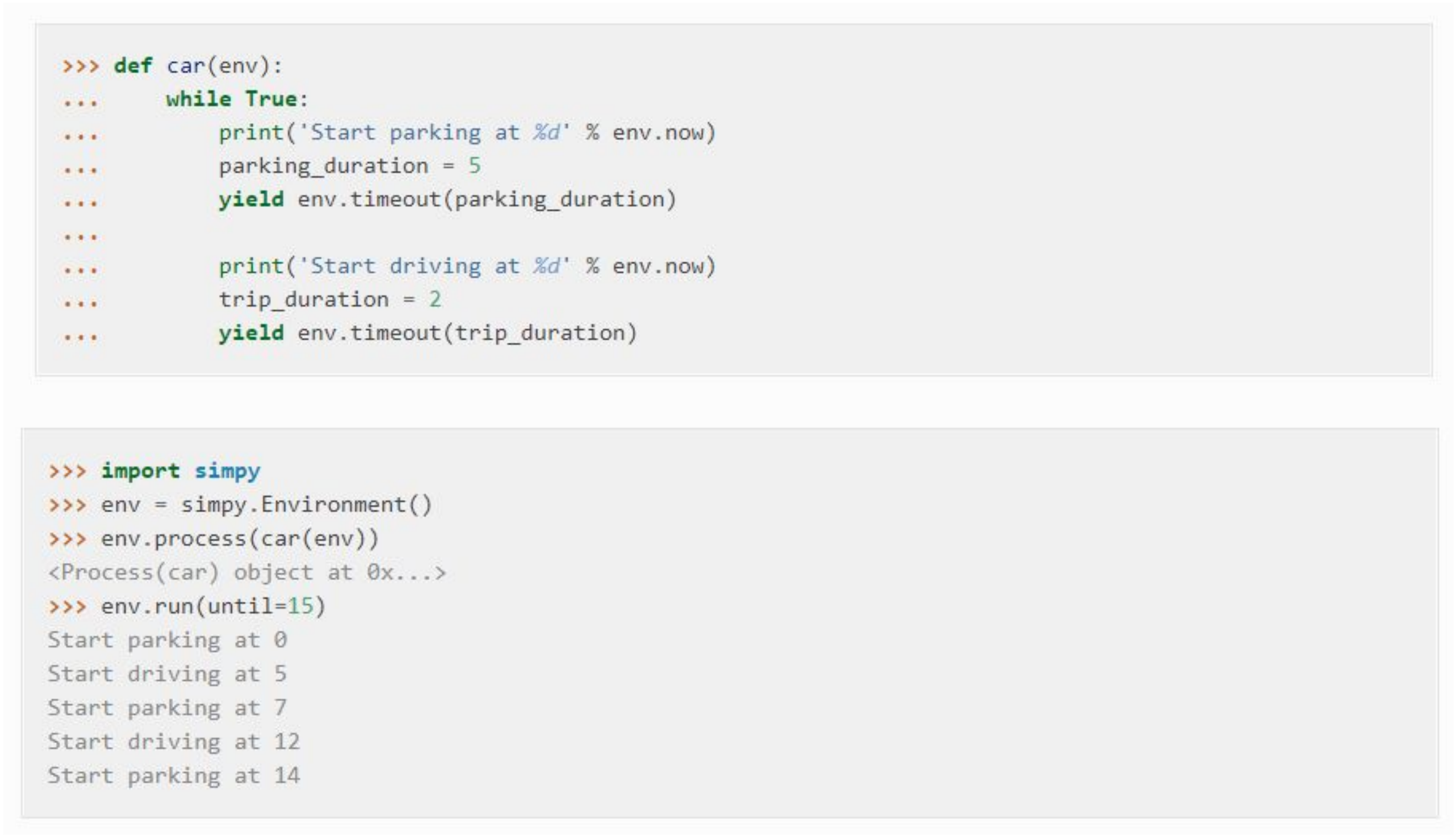
During their lifetime, **Processes create events** and **yield them in order to wait for them to be triggered.**

When a process yields an event, the process gets *suspended*. SimPy *resumes* the process, when the event occurs (we say that the event is *triggered*).

An important event type is the **timeout**.

Events of this type are triggered after a certain amount of (simulated) time has passed (e.g. to model service time).

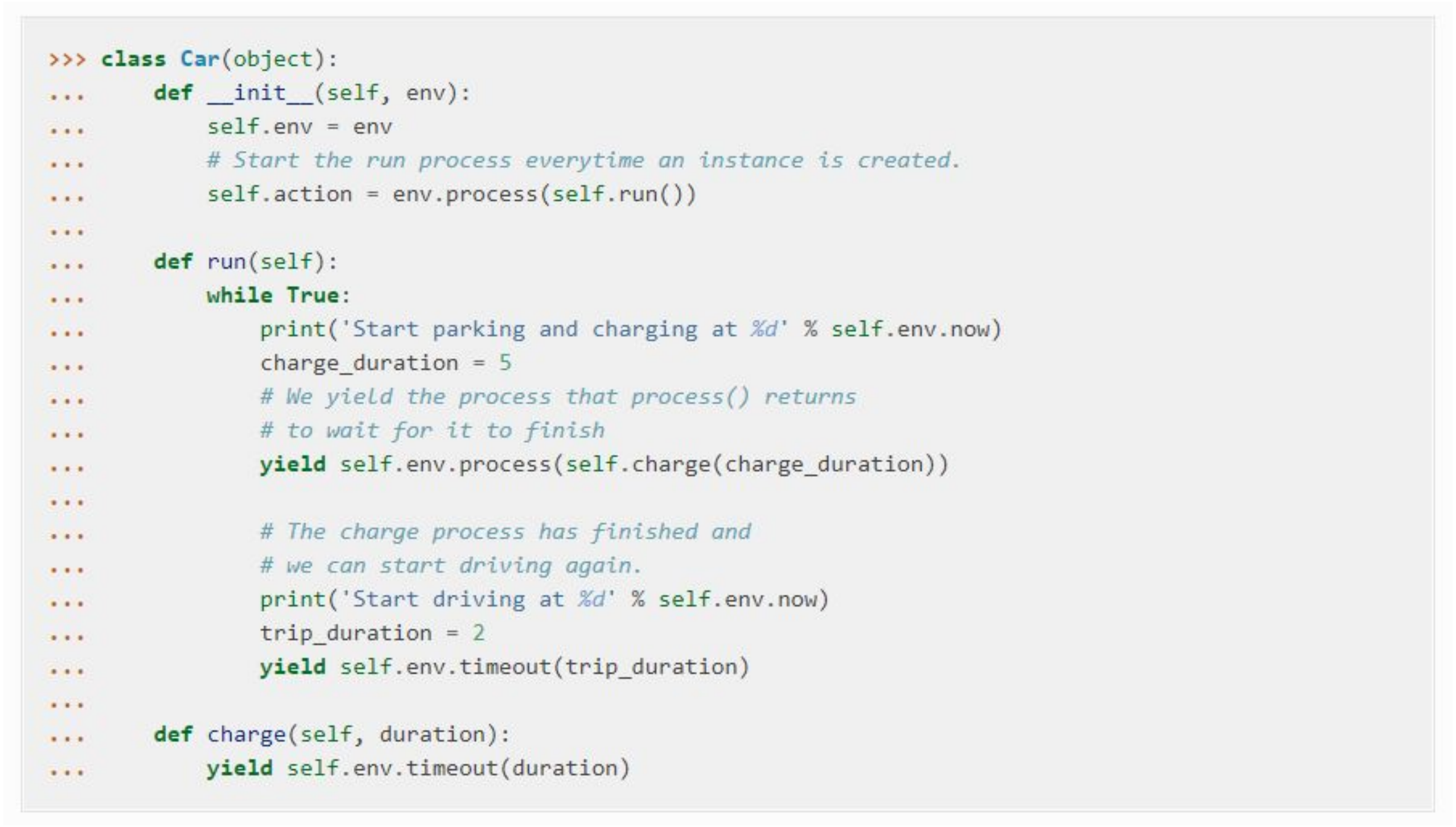
Simply Simple Example



```
>>> def car(env):
...     while True:
...         print('Start parking at %d' % env.now)
...         parking_duration = 5
...         yield env.timeout(parking_duration)
...
...         print('Start driving at %d' % env.now)
...         trip_duration = 2
...         yield env.timeout(trip_duration)
```

```
>>> import simpy
>>> env = simpy.Environment()
>>> env.process(car(env))
<Process(car) object at 0x...>
>>> env.run(until=15)
Start parking at 0
Start driving at 5
Start parking at 7
Start driving at 12
Start parking at 14
```

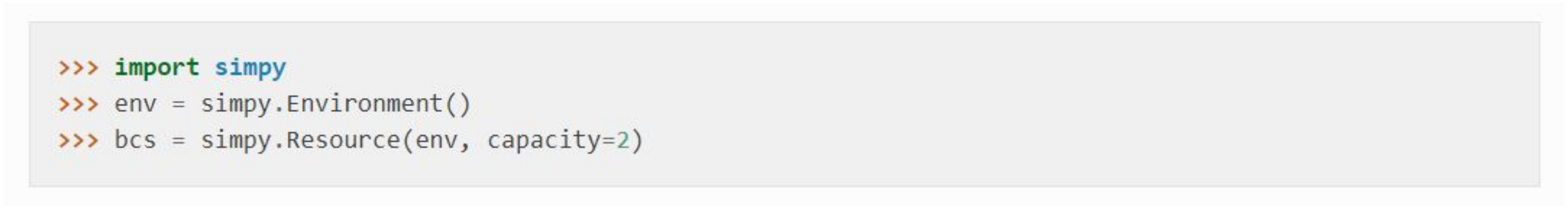
Simply Process Interaction



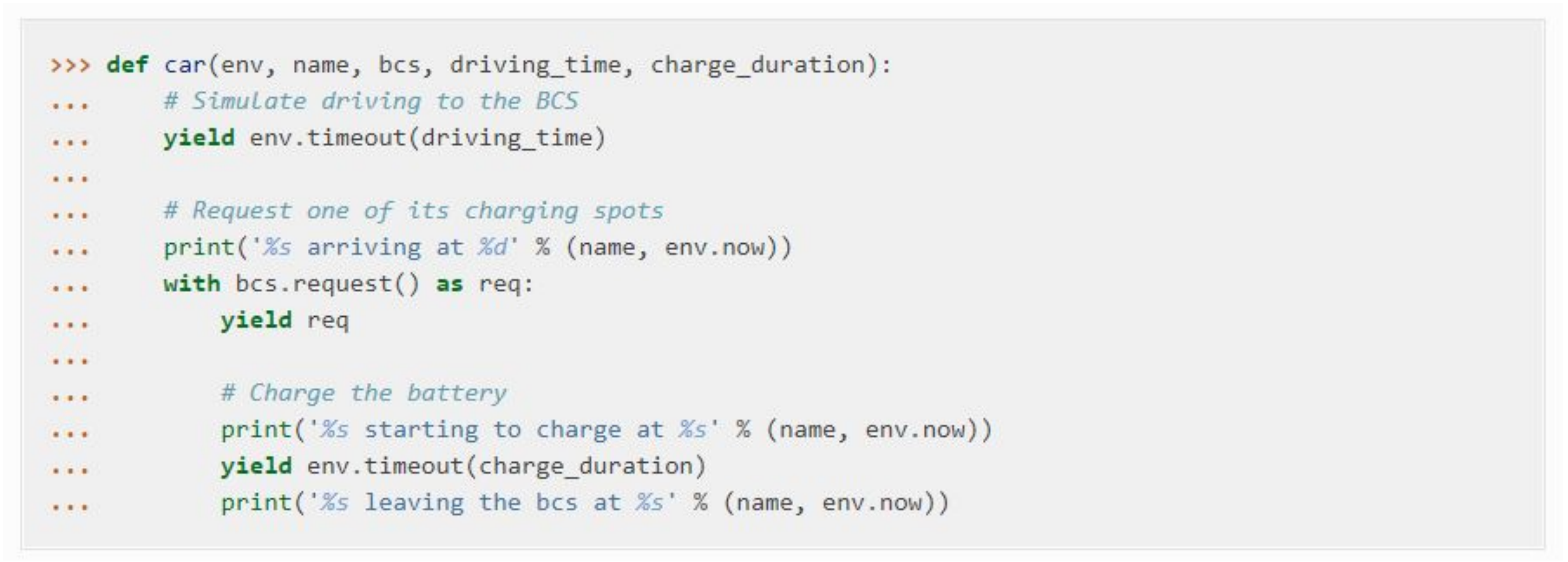
```
>>> class Car(object):
...     def __init__(self, env):
...         self.env = env
...         # Start the run process everytime an instance is created.
...         self.action = env.process(self.run())
...
...     def run(self):
...         while True:
...             print('Start parking and charging at %d' % self.env.now)
...             charge_duration = 5
...             # We yield the process that process() returns
...             # to wait for it to finish
...             yield self.env.process(self.charge(charge_duration))
...
...             # The charge process has finished and
...             # we can start driving again.
...             print('Start driving at %d' % self.env.now)
...             trip_duration = 2
...             yield self.env.timeout(trip_duration)
...
...     def charge(self, duration):
...         yield self.env.timeout(duration)
```

SimPy Resources

SimPy offers three types of **resources** that help you modeling problems, where multiple processes want to use a resource of limited capacity

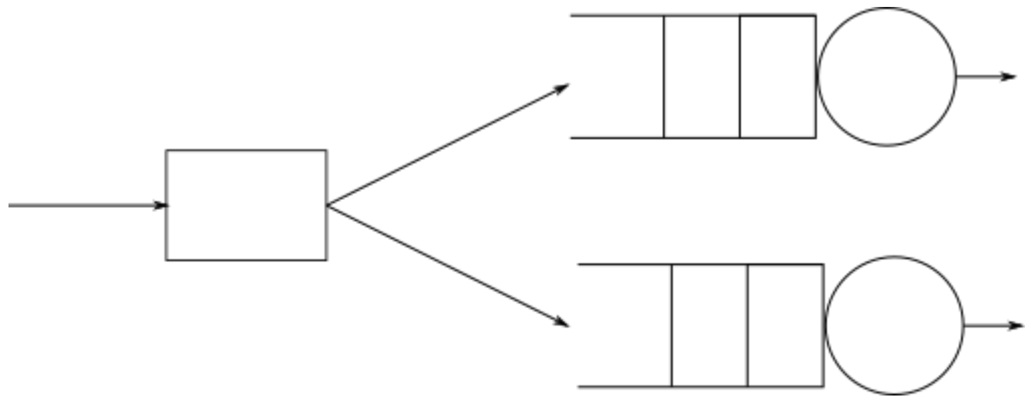


```
>>> import simpy
>>> env = simpy.Environment()
>>> bcs = simpy.Resource(env, capacity=2)
```



```
>>> def car(env, name, bcs, driving_time, charge_duration):
...     # Simulate driving to the BCS
...     yield env.timeout(driving_time)
...
...     # Request one of its charging spots
...     print('%s arriving at %d' % (name, env.now))
...     with bcs.request() as req:
...         yield req
...
...         # Charge the battery
...         print('%s starting to charge at %s' % (name, env.now))
...         yield env.timeout(charge_duration)
...         print('%s leaving the bcs at %s' % (name, env.now))
```

SimPy Queues Example



Join the Shortest Queue (JSQ)

https://bit.ly/2R1yWou

OMNeT++

Issue: how to model a specific network? It is always possible to measure the real-world?

OMNeT++ is an object-oriented modular **discrete event network simulation framework**. It has a generic architecture, so it can be (and has been) used in **various problem domains**:

- modeling of wired and wireless communication networks
- protocol modeling
- modeling of queueing networks
- modeling of multiprocessors and other distributed hardware systems
- **evaluating performance aspects of complex software systems**
- in general, modeling and simulation of **any system** where the discrete event approach is suitable and **can be conveniently mapped into entities communicating by exchanging messages**.

OMNeT++

OMNeT++ provides infrastructure and tools for writing simulations.

An OMNeT++ model consists of **modules that communicate with message passing**.

Simple modules can be grouped into compound modules and so forth; the number of hierarchy levels is unlimited

The whole model, called network in OMNeT++, is itself a compound module.

OMNeT++

Modules communicate with messages that may contain arbitrary data, in addition to usual attributes such as a timestamp.

Simple modules typically send messages via **gates**, but it is also possible to send them directly to their destination modules.

Gates are the input and output interfaces of modules: messages are sent through output gates and arrive through input gates.

An input gate and output gate can be linked by a connection.

OMNeT++ : Building a Simulation

An OMNeT++ model consists of the following parts:

- NED language topology description(s) (`.ned` **files**) that **describe the module structure with parameters, gates, etc.** (it provides a general description of the model). It always includes a *network* description, characterizing the whole model.

- **Simple module sources**. They are C++ files, with `.h`/`.cc` suffix. (they provide the logic of the modules defined through NED files)

- **Configuration .ini file** (**provides the simulation parameter**, used to describe several simulation scenarios)

- (optional) Message definitions (`.msg` files) that let one define message types and add data fields to them.OMNeT++ will translate message definitions into full-fledged C++ classes.

OMNeT++: Parameters

Modules can have parameters. Parameters are used mainly to pass configuration data to simple modules, and to help define model topology.

Parameters such as propagation delay, data rate and bit error rate, can be assigned to connections.

Parameters can be of type double, int, bool, string and xml

Parameters can be assigned in either the NED files or the **configuration file** `omnetpp.ini`., usually employed to describe several simulation scenarios

Simple Modules

In OMNeT++, events occur inside simple modules. **Simple modules** encapsulate C++ code that **generates events and reacts to events**, implementing the behaviour of the module.

Every simple module is derived from a basic class (cSimpleModule).

The **handleMessage()** function will be called for every message that arrives at the module.

The function should process the message and return immediately after that. **The simulation time is potentially different in each call. No simulation time elapses within a call to handleMessage()**.

Modules with handleMessage() are NOT started automatically. This means that you have to schedule **self-messages** (i.e. an event) from the **initialize()** function if you want a handleMessage() simple module to start working "by itself", without first receiving a message from other modules.

Simple modules may also define the **finish()** fuction which is called at the end of the simulations. It is usually employed to store collected statistics.

Starting an OMNeT++ Simulations

Work on Linux or WSL: you will cry less!

https://omnetpp.org/

– before launching the simulation, we have to configure it by creating an initialization file (with suffix .ini) through New/Initialization File (ini).

This file has to indicate at least one ned file containing the description of a network;

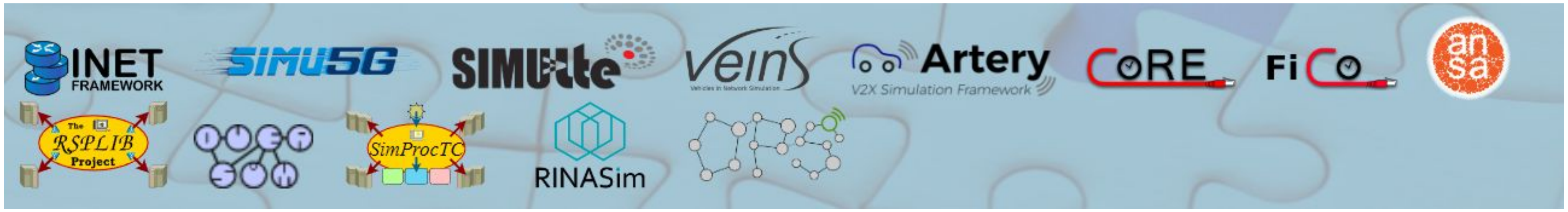
– the launching of a simulation under IDE is done by clicking on a file with suffix ini (generally omnetpp.ini).

Some OMNeT examples
https://drive.google.com/drive/u/2/folders/1DMu8A8nao5PKP-fjQGgExANjlI3qTQA0

Plugins

There are a lots of plugins defined for Omnet++

- INET model suite for wired, wireless and mobile networks https://inet.omnetpp.org/
  - INET provides modules that act as bridges between the simulated and real domains: you can simulate both the network and the application/system or only one of them
  - You can write your service and connect it to a simulated network in Omnet++ https://inet.omnetpp.org/docs/users-guide/ch-emulation.html
- Oversim: overlay and peer-to-peer network simulation framework http://www.oversim.org/
- Many others:



Many other simulators

There exist so **many simulators**, each **specialized in characterizing a specific domain** and providing certain **predefined models**.

QUANTAS: Quantitative User-friendly Adaptable Networked Things Abstract Simulator

https://dl.acm.org/doi/10.1145/3524053.3542744

Quantitative performance analysis of distributed algorithms.

SimGrid https://simgrid.org/

A framework for developing simulators of distributed applications targeting distributed platforms, which can in turn be used to prototype, evaluate and compare relevant platform configurations, system designs, and algorithmic approaches

Dependability Study Example

Perform a dependability evaluation of a distributed protocol

1 - Perform **theoretical evaluation**

- Is the **number and size of** exchanged **messages** upper and/or lower bounded?
- Is the kind and number of tolerable **fault** characterizable?
- If the system is synchronous, it is possible to establish upper and lower bound on the **time required** to execute protocol's operations?
- Is the **load** of the processes/link homogenous?
- …

Dependability Study Example 1

Perform a dependability evaluation of a distributed protocol

2 - Perform **experiments**

a. **Simulation (qualitative)**
   i. Use a simulator (e.g., SimPy, Quantas, Simgrid,)
   ii. Code (Python, Java, etc.) a simulation the protocol execution (instantiate in a single script all the variables off all processes and simulate the execution of the distributed protocol) under certain scenario (i.e., with particular inputs or mimicking faults)

b. **Implementation (quantitative)**
   i. Code a concrete implementation of the protocol and deploys it (multiple OS-process deployment on a machine, Docker, Cloud provider)

Dependability Study Example 1

Perform a dependability evaluation of a distributed protocol

2 - Perform **experiments**

- Define the **workload**
- Perform experiments (**experimental plan**) on specific settings you are interested to analyze
- Test the **occurrence of some faults** if of interest
- The way experiments are performed should be coherent with the aim of the evaluation. E.g. If the aim is to evaluate a concrete deployment, at least a minimal setting should be deployed (Docker for example) to forecast possible behaviours increasing the number of processes

Dependability Study Example 1

Perform a dependability evaluation of a distributed protocol

3 - **Validation and Forecasting**

- If possible, try to define a performance and a dependability model and compare the results between the two models
- Is it possible to attempt to infer how the system may evolve in setting you cannot test/simulate?

References

- A. M. Law - Simulation modeling and analysis https://industri.fatek.unpatti.ac.id/wp-content/uploads/2019/03/108-Simulation-Modeling-and-Analysis-Averill-M.-Law-Edisi-5-2014.pdf

- https://en.wikipedia.org/wiki/Computer_simulation

- https://en.wikipedia.org/wiki/Discrete-event_simulation

- Ken Chen. Performance Evaluation by Simulation and Analysis with Applications to Computer Networks https://onlinelibrary.wiley.com/doi/book/10.1002/9781119006190