Object Oriented Discrete-Event Simulation with OOSimL
CS 4632

Spring 2016

Dr. José M. Garrido
Department of Computer Science
College of Computing and Software Engineering
Kennesaw State University
Definitions Review

● System
  ● a collection of entities that act and interact together toward the accomplishment of some logical end

● Model
  ● a descriptive, functional, or physical representation of a system
Simulation

Simulation is the activity of:

● Developing a model of a real or imagined system

● Conducting experiments with the model – running the model
Overview of a Simulation Study

- Understand the system
- Clearly state the goals of the simulation study
- Formulate the model representation
- Design the conceptual model
- Construct the simulation model with modeling software
- Verify the simulation model
- Validate model
- Design experiments; simulation runs
- Analyze, get insight, document results
Simulation Model Development Process

- System
- Data Gathering & Analysis
- Conceptual Model
- Validation
- Verification
- Implementation
- Simulation Model
- Simulation runs
- Model accepted
- Output Analysis
Applications of Simulation

- As an aid to thought
- As an aid to communication
- For purposes of training and instruction
- As a tool for prediction
- As an aid to experimentation

S. E. Elmahraby

Journal of Industrial Engineering,
1968
Applications of Simulation (cont.)

- Allow to ask what-if questions
- Does not provide a definitive answer
  - provides a probability of outcome
  - result only as good as the model
Simulation Application Examples

- Wargaming
- Stock market
- Weather prediction
- Computational fluid dynamics
- Traffic management
- Manufacturing
- Computer Systems
- Computer networks
- Security
- Astrophysics
- Molecular dynamics
- Business enterprise
- Equipment training
Conceptual Models

- Description of the system, including its goals
- This results from the analysis phase of the development process
- UML and other modeling notations are used to describe structure and behavior of a model
Conceptual and Simulation Models

Diagram:

- Real world
  - Abstract level
    - Software level
  - Conceptual model
    - Simulation model
  - Real system
Simulation Model

- Built from the conceptual model of a system.
- Implemented in an appropriate programming or simulation language.
- Is needed to mimic the behavior of the real system under certain constraints.
Development of Simulation Models

Two main objectives:
- To study some relevant aspects in the dynamic behavior of a system
- To estimate various performance measures of the system
Simulation Run

A simulation run is an experiment carried out for some observation period using the simulation model to study (predict) the behavior of the system.
Behavior of a Simulation Model

Depends on:

- The passage of time
- Input data
- Events generated by the environment
Types of Models

- Physical models
- Graphic models
- Mathematical models
- Other kinds of models
Stochastic Models

- A stochastic model includes some **uncertainty** in its behavior.
- One or more attributes change value according to **probability distribution**. Random variable generation is used.
- For example, in the carwash model:
  - arrival of cars
  - service time for a car
Continuous and Discrete Models

- **Continuous models** change their state continuously with time. Mathematical notation is used to completely define the behavior. For example, the free-falling object.

- **Discrete models** only change their state at discrete instants. For example, the carwash model.
Consider the simple carwash model, its dynamic behavior is defined by:

- When an arrival event occurs, the model changes state.
- When a service completion occurs, the model also changes state.
Queue Size in the Carwash Model

Example of a state variable
Continuous Model
The SIMULA language was developed by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center (NCC) in Oslo around 1962.

Simula was the first language to introduce the concepts of classes and objects. The language also introduced the idea of prefixing and subclasses.

The key concepts have evolved since then and have been implemented in most modern OO programming languages.
Demos was developed by Graham Birtwistle and extends Simula.

It is a library of standard process synchronization facilities for data collection, random number generation, tracing and automatic reporting.
SIMULA and DEMOS

- The Demos package added more powerful simulation facilities to the Simula language.
- Simula and Demos have not become widely used, despite the significant contribution to object orientation and simulation.
In the early 1970’s, Alan Kay and his group at Xerox PARC used Simula as a platform for their development of Smalltalk.

Their development extended object-oriented programming importantly by the integration of graphical user interfaces (GUI) and interactive program execution.
C++ and Other OO Languages

- In the mid 1980’s, Bjarne Stroustrup started his development of C++ by bringing the key concepts of Simula into the C programming language.

- Other object-oriented programming languages such as Eiffel, CLOS, Java, and others, were also influenced by Simula.
Important Quotes

- “Simulation is one of the most powerful techniques available for the study, design, and operation of complex systems” (R. Shannon).
- “Mathematical modeling is essentially a conceptual way of thinking” (L. Schruben)
- “... the task of modeling a system is in itself worthwhile ...” (L. Schruben)
Simulation in Education

- Modeling and Simulation is an important part of the Computing curriculum.
- In object-oriented simulation, students
  - Apply understanding of OO concept when modeling, i.e. building the conceptual model
  - Implementing a simulation model in an OOP language.
Simulation in Education (2)

- Use of OO packages in teaching object-oriented simulation reinforces object-oriented design and programming in Java.
- Students have learned OO programming principles using Java in introductory courses.
- OO simulation exposes students in programming in the large.
- Students have taken courses in systems analysis and design using UML for modeling.
Simulation Software Tools

At KSU, the following general tools are available to all students and faculty:

- Arena simulation software (academic version)
- Psim3, a C++ simulation package
- PsimJ2, a Java simulation package
- jGRASP & Eclipse integrated development environments (IDE)
- The new OOSimL Object Oriented Simulation Language, part of the OOPsim project
OOPsim Project

- The main purpose of OOPsim project is to investigate improved methods and tools for the education of object oriented modeling and process approaches to discrete-event simulation.

- Provide open access to the software and documentation for educational and research purposes.
OOPsim Simulation Project

- Started in 1996 with the Psim simulation package in C++
- Enhanced in 1999 (Psim2)
- Re-implemented in Java in 2000 (PsimJ)
- Enhanced and updated in 2002 and 2003
- Psim3, new version C++ redesigned using POSIX Pthreads, in 2003
PsimJ2 Simulation Package

- A portable library package of Java classes
- Object-oriented modeling
- Supports the process interaction approach
- Java threads are used to implement the active objects in the simulation models
MODSIM III & Simscript III (CACI)

- Commercial OO simulation languages
- Object-oriented
- Supports process-based discrete event simulation
- Modsim III No longer supported
Downloading PsimJ2, OOSimL, and Sample Models

- Free for educational and research purposes
- Download from the following URL:

ksuweb.kennesaw.edu/~jgarrido/psim.html
The advantages and disadvantages of an object-oriented simulation programming language compared to an integrated icon-based simulation tool are:

- Flexibility and
- Power of the simulation language
- May be more difficult to use than an integrated simulation tool.
Other Advantages

- The OOSimL language also supports further research in: simulation languages, developing Simulation Modeling Specification Languages for relevant family of systems, higher-level DEVS support, advanced Cellular and Component Simulation modeling, the next generation of Distributed and Parallel Simulation Languages, and other related areas of research.

- The language supports the application of simulation in industry; it provides a viable and short transition to current commercial simulation (programming) languages such as Simscript III.
System Components

- **Active (or live) components**: the processes. These are the major components in a model, and have a life of their own.
- **Passive components**:
  - Explicit and implicit queues
  - Resources that can be acquired by the processes.
Using A Simulation Model

A user must be allowed to:

- Manipulate the model by supplying it with a set of inputs
- Observe its behavior or output
- Predict the behavior of the real system by analyzing the behavior of the model
Supporting Software

- The simulation executive. The program that controls every simulation.
- The set of utility functions for housekeeping tasks.
- Reporting simulation trace and output statistics.
Facilities Provided by Simulation Software

- Maintain the sequence of event records (pairs of event and event time)
- Provide mechanisms for the generation and cancellation of event records
- Maintain the simulation clock
- Provide functions to generate random numbers from common probability distributions
The Time Dimension

● Simulation time, the time referenced by executing simulation models. This time is advanced by:
  ● Fixed time increments
  ● Variable time increments (event scan)

● Run time, the absolute time referenced by the OS while a program executes.
World View with Discrete-Event Simulation

- Event approach
- Activity approach
- Process interaction approach
The Process Interaction

- A simulation model consists of a description of how processes (active entities) are going to interact among themselves, as time passes.
- Also includes a description of passive objects
- A simulation run consists of creating and starting the processes interacting among themselves, synchronizing and using resources.
Advantages of the Process Style of Simulation

- Compatible with the Object-Oriented approach to modeling and programming, every process is an active object.
- Suitable for modeling large and complex systems.
- C++, Java and any higher-level object-oriented simulation language are used to implement the models.
Processes

- Processes are active objects implemented as threads in Java and C++.
- The main body in a thread consists of a set of instructions and calls to various functions (or methods).
- The general behavior of a thread is defined in class Process of the OOsimL language.
Process View

- Static Process Description
- Dynamic Process Management
  - Initiate Process
  - Delay (Time) Process
  - Suspend Process
  - Activate Process
  - Terminate Process
Resource View

- Standard/Tailored Resource Handler
- Resource Management
  - Create resource
  - Request resource
  - Acquire resource
  - Release resource
  - Status of resource
  - Destroy resource
Simulation Executive

- Schedule processes at particular instants, i.e., place processes in the event list
- Remove processes from the event list (to make them idle or to terminate them)
- Re-schedule processes, i.e., change position of processes in the event list.
- Execute the “running” process.
The Simulation Executive

1. Initialize & set clock to zero
2. Search event list for the event that is to occur next.
3. Advance clock to the event time. Transfer to that event in appropriate process.
4. Thread 1
   React. point
5. Thread 2
   React. point
6. Thread 3
   React. point
7. Thread n
   React. point

No events pending → Stop
Threads

- A thread is an independent execution sequence of an application that may interact with other threads
- A thread is executed and controlled by the simulation executive
- The threads of a model are executed concurrently
Threads

- Simulation models implemented in OOSimL use oosimlib, an extended version of the PsimJ2 simulation package
- PsimJ2 and OOSimL use Java threads
  - Basically, class Process inherits class Thread
- Psim3 and C++ version of OOSimL use Pthreads
Every simulation model defines a set of classes that will be instantiated appropriately.

The classes that represent processes inherit class `Process`, which inherits class `Thread`.

The other classes defined are used to create passive objects.
Simulation Packages

- SimPack and Sim++
- GPSS/V & GPSS/H
- CSIM
- Gasp
- SLAM
- Simple++
- Taylor II

- SimJava
- Arena
- Extend
- AweSim
- iThink
- Simul8
Simulation Packages in Java

- JavaSim
- JSIM
- DESMO-J
- Simjava
- Silk
- PsimJ2
Simulation Languages

- Simula
- GPSS
- Simscript 2.5
- Modsim III
- Simscript III
- OOSimL
Overview - Levels of Abstraction

- In modeling a system, different levels of abstraction are used to describe the system.
- The most abstract level is at the top (high), and it is the easiest to understand.
- The most difficult level of abstraction is the implementation level, i.e., the programming of the model into a suitable simulation programming language.
Together with abstraction, the object-oriented approach is applied to deal with complexity, at all levels of abstraction.

Java and C++ are high level OO programming languages and are convenient and powerful to implement simulation models.

A simulation language is considered higher-level.
Process Interaction Simulation

- The major components in a model are active and passive entities.
- The active entities are called processes and have a life of their own.
- The passive entities are resources and queues.
- The processes interact among themselves and use resources.
OOSimL

Object Oriented Simulation Language
The OOSimL Simulation Language

A general-purpose simulation language that supports:

- The object-oriented approach to modeling and the process interaction approach to discrete-event simulation
- Higher-level simulation modeling
The purpose of this language and software tool is to support early introduction to

- Abstraction
- Object-oriented modeling
- Logic
- Discrete-event simulation
- Other software concepts.
An OO Simulation Language

- The simulation modeling in OOSimL follows the philosophy of Simula and Demos.
- The OOSimL compiler carries out syntax checking and automatically generates an equivalent Java program.
Using the OOSimL

- The OOSimL (Java) software consists of:
  - The OOSimL compiler, `oosiml.exe`
  - The run-time library, `oosimlib.jar`

- There are two general procedures for developing programs and models using the OOSimL compiler:
  - In an integrated development environment (IDE), such as jGRASP and Eclipse. The OOSimL Web page includes instructions on how to configure and use the IDEs to compile and run OOSimL programs
  - In a Command window on Windows or in a Terminal window on Linux
Modeling with OOSimL

- Models implemented with OOSimL can be easily be integrated with Java libraries.
- The language supports the reuse of Java components.
- A dynamic system is modeled as a set of interacting processes.
- A process is implemented as a thread object. The other components of a system are modeled as conventional (passive) objects.
Simulation Facilities in OOSimL

Includes language statements for manipulating processes:

- Timing,
- Sequencing,
- Synchronization
Language Simulation Features

- Definition of processes.
- Starting of a simulation run that will execute for a predetermined period, called the simulation period.
- Definition of various queues that are used in the simulation models.
- Definition of resource pools as passive objects.
- Generation of random numbers, each with its own probability distribution.
More Information on OOSimL

For download information on OOSimL, refer to the following URL:

ksuweb.kennesaw.edu/~jgarrido/psim.html
The Structure of a Model

- Definition of all classes for the various active objects (processes)
- These classes inherit the library class Process.
- Definition of the class (model class) for the process that represents the model
- Method main with instructions that starts the simulation
The Main Class in a Model

Every model includes a main class (the model main class). In method \textit{main}:

- Assigns the values for the relevant simulation parameters
- Set up Simulation with the title of the model.
- Create all the passive objects used in the model.
- Create and start the main active object
Main Body

In method *Main body* of the main class

- Create and start the other active objects (processes) of the model
- Start the simulation, then calculate and output the summary statistics.
General Class Structure

class < class_name > as process is

  private, protected, public data definitions

  private, protected, public methods

dendcodeclass < class_name >
class Server as process is
  private
  // private data definitions
protected
  // protected data definitions
public
  // public data definitions
  // private, protected, and public methods
endclass Server
Data Definitions

Data must be defined in the following order:

- Constants
- Variables
- Object references
// data definitions
constants
    define NUM = 45 of type integer
variables
    define machine_name of type string
    define acc_time of type double
...
object references
    define machine of class Server
    define custobj of class Customer
Methods or Functions

function initializer
parameters serv_name of type string
is
begin
    call super using serv_name
    ...
endfun initializer

//
function Main_body is
    ...
begin
    ...
    call get_name of machine return value to machine_name
    ...
endfun Main_body
// other public members
Setting Up and Starting a Simulation

define simperiod = 345.5 of type double

... define run of class Simulation

... simulation title "Carwash Model"

... start simulation with simperiod
Hold (delay) a Process

The following lines of code hold the execution of the current process for a time interval, \textit{task\_dur}.

\begin{verbatim}
define task\_dur of type double
...
hold self for task\_dur
hold self for 2.35
\end{verbatim}
Queuing Models

- Models that include at least one queue
- The queue is a data structure that stores waiting customers
- The server removes from the queue the next customer to service
Carwash Model – Single-server

Vehicle arrivals → Queue → Car-wash machine → Vehicle departures
Basic Categories of Queuing Models

- Single-server models
- Multi-server models
- Queuing networks
- Multi-server, multi-queue with priorities
- Multi-server, multi-queue with priorities and preemption.
Examples of Queuing Models

- Computer systems
- Communication systems, networks
- Teller facilities in a bank
- Production systems
- Transport and materials handling
- Repair and maintenance systems
A queuing model consists of a number of facilities and interconnecting queues. Each facility, or service station, consists of one or more servers.

For these types of systems, it is very common in practice to use exponentially distribution for inter-arrival periods and for service periods.
Multiple Servers Single Queue

- Any of the servers available can provide service demanded by a customer
- A server removes the next customer from the head of the customer queue
- One or more servers can be idle, waiting for customers to arrive
Carwash Model With Multiple Machines
General Description of the Model

- This model has K machines (servers)
- A customer can receive service from any machine (any server)
- A customer waits in the customer queue until it is at the head of the queue and a machine becomes available
- The machine removes the customer from the head of the queue and starts to service it
Idle Servers

- Idle servers are organized in a **server queue**
- A server joins a queue of idle servers if there are no customers
- An arriving customer enters the input queue and reactivates the server at the head of the server queue. If there are no servers available, the customer just waits
Models With Multiple Servers and Multiple Queues
Multi-Server Multi-Queue Model

- Every server has its own queue of customers
- An arriving customer selects the server with the smallest queue
- Alternatively, an arriving customer randomly selects a server.
Multi-Server Multi-Queue Model with Preemption

- Every customer has a priority
- An arriving customer can interrupt the service of another customer with a lower priority
- The interrupted customer is returned to its queue, this customer is the one with the lowest priority and that started most recently.
Performance Measures

- The average number of customers in the system
- The average number of customers in the queue(s) (i.e., that are waiting)
- The average time that a customer spends in the system
- The average time that a customer spends in the customer queue waiting
- The server(s) utilization
Queuing Networks

- Consists of two or more interconnected stations
- Each station provides a different service
- The output from a station are connected to input of another station
- Each station has one or more servers and its queues
Service in a Queuing Network

- Customers enter the network at any of several entry stations
- Customers are routed through the network demanding service from different stations
- Customers can exit from any of several exit stations
A Queuing Network

Queuing Network

Station 1

Station 2

Station 3

Station 4

Station 5

Station 6
Models With Priorities

- Customers are grouped into classes or types.
- Each customer class has its own priority, and workload parameters (e.g., arrival rate and a service demand).
- The customers wait in a priority queue, with highest priority closest to the head.
- A server removes the customer with the highest priority from the queue.
Examples

- Medical service systems
- Computer communication networks
- Operating systems
- Car-wash systems for different types of vehicles
Priority Queue

- A queue that stores customers by priority
- The queue discipline is ordering of the customers by priority
- Customers with the highest priority are always at the head of the queue
Model With Priorities

Cars \rightarrow \text{Priority queue} \rightarrow \text{Server}
Priority Queues in OOSimL

- To declare a priority queue, use class \textit{Pqueue}:
  
  \texttt{define car\_queue of class Pqueue}  // for car objects

- To create object \texttt{car\_queue} (passive object) with \( K \) different priorities:
  
  \texttt{create car\_queue of class Pqueue using “Car Queue”, K}

- In OOSimL, the highest priority is 0
Priorities For the Car Objects

- There is an arrivals object for every customer class
- Each arrivals object creates car objects for the corresponding priority
- A car object gets its priority when created (in the constructor)
- A car object joins the corresponding priority queue to wait for service
Defining and Using Car Priority

The constructor in class *Car*:

function initializer parameters cname of type string, ctype of type integer, servdur of type double

is …

set my_priority = ctype;  // assign priority to car process
fix priority my_priority

In the *Main_body* of class *Car*:

// this object joins the priority queue
insert self into car_queue
Results of a Simulation Run

Two text files are normally generated by a simulation run:

- The **trace file**, which is the sequence of events that occur during the simulation run
- The **statistics file** that include performance measures for every priority:
  - Throughput
  - Average wait period
  - Other measures
Multi-Server Model with Priorities

- Every customer has a priority and joins the priority queue
- An server gets the customer from the queue with highest priority
- If there are no customers, and server joins a server queue and becomes idle
Multiple Servers With Priority

Diagram:
- Cars arriving
- Priority queue
- Machine 1
- Machine 2
- Machine 3
- Machine K
Verification & Validation

Real-world system

Requirements

Conceptual model

Verification

Simulation model

Validation
Questions?