

Object Oriented Modeling and Simulation

The OOSimL Language

Dr. José M. Garrido C.

Department of Computer Science
College of Computing and Software Engineering
Kennesaw State University

Spring, 2017

Simulation Software (Review)

Implementation of a simulation model can be accomplished with a general-purpose high-level programming language, however, it is much easier and convenient to use some type of simulation software. The various levels of simulation support software are:

- 1 A general-purpose (object-oriented) programming language, such as C++ and Java
- 2 A simulation package or library, such as PsimJ2, Silk, and Simjava, Psim3. These are used with a general-purpose programming language
- 3 An object-oriented simulation language, such as OOSimL and Simscript III
- 4 An integrated simulation environment, such as Arena, FlexSim, Simul8, and ProModel.

The OOPsim Project

Goals

- Investigate and develop new simulation **tools and approaches** for education
- Promote construction and use of **simulation models** in various areas of computing (including *Information Security*)
- Provide **free access** to the simulation software and several of the simulation models:

`ksuweb.kennesaw.edu/~jgarrido/psim.html`

The project has received support from Epscor, CSIS department, Yamacraw, and recently from NSF.

OOPsim Tools Developed

- Started in 1996 with the object-oriented Psim simulation package in C++
- Enhanced in 1999 and evolved to **Psim2**
- Re-implemented in Java in 2000 and named **PsimJ**
- A new version of Psim2 (C++) was redesigned using POSIX Pthreads in 2003, named **Psim3**
- **PsimJ2** was developed in 2004, an extended version of PsimJ
- The **OOSimL** simulation language was developed in 2009 with support of an NSF CPATH grant

OOSimL

Object Oriented Simulation Language

Brief History of Object Oriented Simulation

Simula

- 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 in 1979 and extends Simula
- 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, 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 Languages

C++

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 OO Languages

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

Motivation for Language Development

The Language

OOSimL is an experimental simulation language for **research and education** in discrete-event simulation modeling using the object-oriented and process oriented approaches.

Its general goal is to reduce the difficulty and complexity of implementing object-oriented simulation models

Target Audience

The language was developed mainly for **students** of computer science, software engineering, and related disciplines.

The OOSimL Language

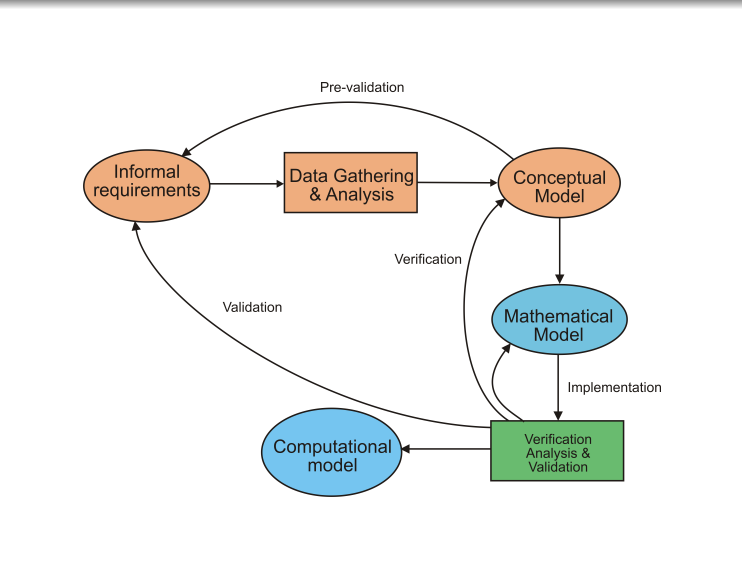
Goal

The language emphasizes object oriented modeling and programming, multi-threading, and promotes an early introduction to simulation in the curricula.

Influences

The language design was influenced by Simula, Eiffel, Ada, Java, C++, and the Demos simulation package.

General Method for Model Development



OOSimL Language Features

- Supports object orientation
- Supports the **process style** of discrete-event simulation that facilitates the teaching and learning of modeling and simulation of large and complex systems.
- Helps promote and support *early introduction to simulation* in the computing curricula.
- Promotes the application of object-oriented simulation in **industry**; it provides a viable and short transition to current commercial simulation (programming) languages such as Simscript III.

General Language Statements

The language includes the following general types of language statements:

- Definition of classes for passive objects needed by processes.
- Definition of classes for processes.
- Starting of a simulation run for a predetermined simulation period.
- Definition of queues.
- Definition of resource pools and other passive objects.
- Generation of random numbers, each with its probability distribution.

The Output of a Simulation Run

Text Files

- The complete simulation trace
- The summary statistics and performance metrics.

Other Output

The other types of output will usually be the various GUI boxes and graphical animations that the model implementation include.

Downloading OOSimL Files

The OOSimL simulation translator, libraries, documents, and sample models can be downloaded from the Web site:

`ksuweb.kennesaw.edu/~jgarrido/oosiml.html`

- For generating Java code: download the archive file:
`oosiml.zip`
- For generating C++ code: download the archive file:
`oosimlc.zip`
- Additional document files are also available in the web page.

The OOSimL Software

The simulation software consists of two sets of components:

- ① For generating Java code:
 - ① the compiler, which is an executable program named `oosiml.exe`
 - ② the run-time library, `oosimlib.jar`.
- ② For generating C++ code:
 - ① the compiler, which is an executable program named `oosimlc.exe`
 - ② the run-time library named `oosmlibc.a` and the header files.

The OOSimL Compiler

The OOSimL compiler has the following functions:

- 1 Syntax checking of a source program in OOSimL; a list of errors is displayed and written to an error file with the same name as the source program
- 2 Generation of an equivalent program in Java or in C++

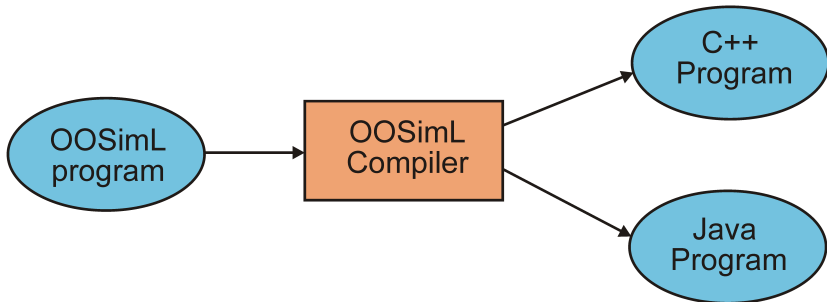
Developing OOSimL Models

Currently, there are three general procedures used for developing programs and simulation models with the OOSimL compiler:

- 1 Developing the simulation model with the [Eclipse](#) integrated development environment.
- 2 Working with the [jGRASP](#) integrated development environment (IDE).
- 3 Working in a [Command](#) window.

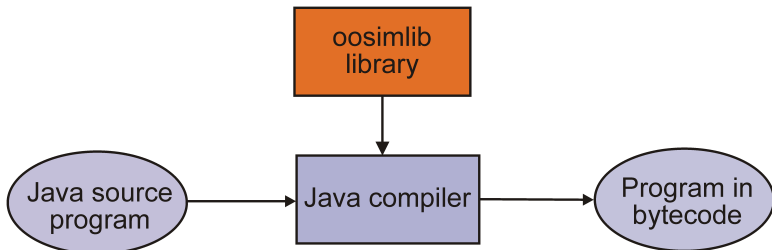
The First Phase of OOSimL Compilation

The first phase involves a translation into an equivalent Java or C++ program.



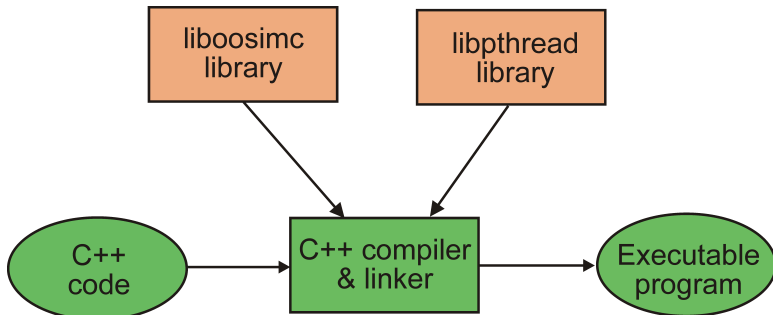
The Second Phase of OOSimL Compilation

When Java code is generated, the second phase involves the corresponding Java compilation to Java byte-code.



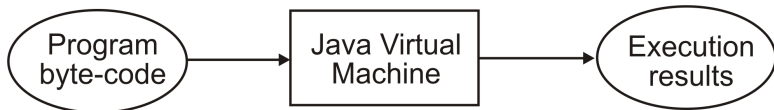
The Second Phase of OOSimL Compilation

When C++ code is generated, the second phase involves the corresponding C++ compilation and linking with the *oosimlc* and the *pthread* libraries.



Running the OOSimL/Java Program

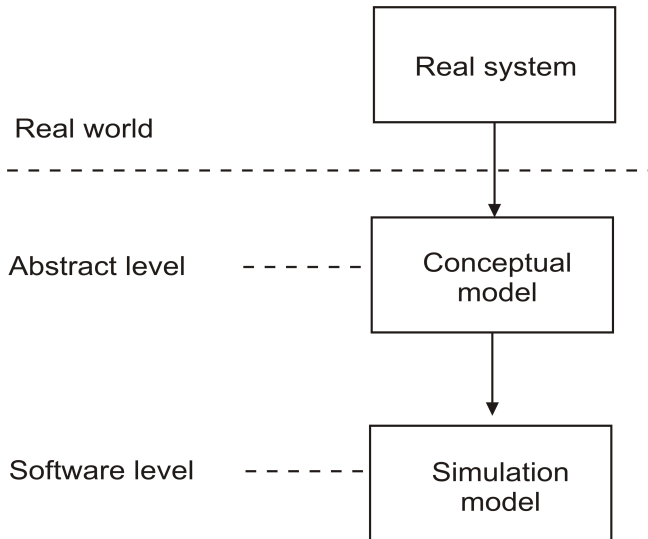
The Java virtual machine (JVM), which is another software tool from Oracle, carries out the interpretation of the program in bytecode.



- Programs written with OOSimL can be easily integrated with Java programs.
- Any Java library can directly be accessed by a model written in OOSimL.
- The OOSimL language supports the reuse of Java components.
- The run-time library of this language is a collection of Java classes.

- Programs written with OOSimL can be easily integrated with C++ programs.
- A C++ library can directly be accessed by a model written in OOSimL.
- The run-time library of this language is a collection of C++ classes.

Conceptual Model



- The OOSimL language supports not only object orientation but also the conceptual frame-work of the process style of discrete-event simulation. This facilitates modeling and simulation of large and complex systems.
- OOSimL helps reinforce the ability to carry out object oriented modeling and apply software development principles learned previously.
- The following sections include an overview of the main features of the OOSimL language, a brief description of the language translator, and an example of a OOSimL program developed in Eclipse.

The OOSIML Translator

- The OOSimL language translator is an executable program that is implemented as a one-pass language processor that generates Java or C++ source code.
- The translator is actually a C++ program that has been compiled (and linked) on Linux and Windows.
- The generated code can be integrated conveniently with any Java or C++ library. The general block diagram of the OOSimL translator is shown in the next figure.

More on Translation

- After the OOSimL translation, the next steps in program development are the standard Java and C++ steps of compiling, linking, execution, debugging, and so on.
- Eclipse is one of the freely available and most widely known IDEs for program development.
- Kevin Sealy has developed a plugin for Eclipse to improve the editing and manipulation of OOSimL source code and to integrate the editing and translation of OOSimL programs with the rest of the program development stages.

General Program Structure in OOSimL

The general structure of a class definition in OOSimL is:

```
description      . . .  
class <class_name> is  
  private  
    data declarations (attributes)  
    object reference declarations  
    function definitions  
  public  
    data declarations (attributes)  
    object reference declarations  
    function definitions  
endclass <class_name>
```

General Program Structure of a Function

The general structure of a function (or method) definition is:

```
description . . .  
function <function_name> is  
    constants . . .  
    variables . . .  
    object references . . .  
    begin . . .  
endfun <function_name>
```

Generating C++ Programs

For generating C++ programs, the general structure of a OOSimL program includes the following sequence of steps:

- 1 Import the necessary header files
- 2 Forward references: function prototypes and class list
- 3 Global declarations: constants, variables, and object references
- 4 Class specifications
- 5 Implementations: classes and non-member functions

Examples of OOSimL Programs

Complete examples of OOSimL programs for generating Java and C++ code, the complete software, which includes the compiler, runtime libraries and several examples, can be downloaded from the Psim Web page:

<http://ksuweb.kennesaw.edu/~jgarrido/psim.html>

OOSimL Class Specification Example

description

This class computes the area and perimeter
of a triangle, given its three sides.
*/

class Triangle inherits Genfig is

private

variables

define x of type double // first side

define y of type double // second side

define z of type double // third side

public

description

This function sets values for the three
sides of the triangle. */

function initializer parameters a of type double,
b of type double, c of type double

OOSimL Class Specification Example

description

This function computes the perimeter of a triangle. */

function perimeter return type double

description

This function computes the area of a triangle. */

function area return type double

endclass Triangle

OOSimL Class Implementation Example

```
class body Triangle inherits Genfig is
  description
    This function sets values for the three
    sides of the triangle.      */
  function initializer parameters a of type double,
                                b of type double, c of type double
  super using a
  is
  variables
    define j of type integer
  begin
    set j = 3
    set x = a
    set y = b
    set z = c+j
  endfun initializer
```

OOSimL Class Implementation Example

```
description
  This function computes the perimeter of a
  triangle. */
function perimeter return type double is
variables
  define lperim of type double
begin
  set lperim = x + y + z
  return lperim
endfun perimeter
```

OOSimL Class Implementation Example

```
description
    This function computes the area of a
    triangle.      */
function area return type double is
variables
    define s of type double
    define r of type double
    define larea of type double
begin
    set s = 0.5 * (x + y + z)
    set r = s * (s - x)*(s - y)*(s - z)
    set larea = call sqrt using r
    return larea
endfun area
endclass Triangle
```