

OPEN-SOURCE SIMULATION SOFTWARE “JAAMSIM”

D.H. King
Harvey S. Harrison

Ausenco
855 Homer Street
Vancouver, BC, V6B 2W2, CANADA

ABSTRACT

JaamSim is a free, open-source simulation package written in the Java programming language. A modern graphical user interface is provided that is comparable to commercial software, including drag-and-drop model building, an Input Editor, Output Viewer, and 3D graphics. Users are able to create their own palettes of high-level objects using standard Java and modern programming tools such as Eclipse. If you are writing hundreds or thousands of lines of code in the proprietary programming language provided with your commercial software, you would be far better off to write your code in Java and use JaamSim.

1 INTRODUCTION

This paper introduces ‘JaamSim’, a free, open-source simulation package developed by [Ausenco](#), a global engineering company. JaamSim is aimed at a broad audience of professionals, researchers, and students, and includes a modern graphical user interface (GUI) that is comparable to those provided by commercial off-the-shelf simulation software. The software is written in the Java programming language and can be downloaded from [Github](#). The executable, user manual, programming manual, and examples are available from the authors on request.

The key feature that makes JaamSim different from commercial off-the-shelf simulation software is that it allows users to create their own palettes of high-level objects for new applications. “High-level” objects are ones that simulate the significant objects for a given class of model. For a traffic simulator, the high-level objects would be the various types of vehicles, roadways, and intersections. Conversely, “low-level” objects are more fundamental and have a broader application, for example queues and servers. Complex models can be built much more easily from high-level objects than from low-level objects.

New objects programmed by the user automatically have 3D graphics, can be dragged-and-dropped, have their inputs editable through the Input Editor, and their outputs displayed in the Output Viewer. Programming is done in standard Java using modern development tools such as Eclipse. Unlike commercial simulation software, JaamSim is not restricted by the idiosyncrasies and limited feature set of a proprietary programming or scripting language.

JaamSim is not the first open source simulation offering in Java. Previous open source simulation engines in Java include: CSIM, DESMO-J, DEUS, DSOL, JavaSim, JiST, Jsim, JSL, SimJava, Simkit, SSJ, and Tortuga. Although these packages offer many useful features that make them good choices for some applications, each provides only a library of simulation tools and the user is left to prepare his or her model from Java code. Without a GUI, these packages appeal only to people with the time and programming skills necessary to build models in this way.

One exception to this observation is Simkit, which provides the VisSim visual event graph builder. However, this GUI is limited to constructing models from low-level event graph components, which is only one part of the functionality that is needed for the software to have broad appeal.

To the best of our knowledge, JaamSim is the first open-source simulation software to include a modern GUI that is comparable to commercial off-the-shelf simulation software. We believe that JaamSim will appeal to far more people than has been the case for previous open-source simulation projects.

Finally, the authors would like to make it completely clear that we are offering JaamSim as a genuine open-source project for the benefit of the simulation community. It is free software licensed under GPLv.3. We have no plans to introduce a paid “premium” version of JaamSim.

2 INSTALLING AND USING JAAMSIM

JaamSim is an extremely light-weight application consisting of a single 8 MB executable. The executable can be copied directly to the user's computer. No special installation program is required. JaamSim will run on most modern computers that support OpenGL graphics version 3.0 or later, including laptop computers with Intel Core i5 and i7 series processors that rely on integrated graphics (Second Generation, “Sandybridge”, and later versions).

The graphical user interface (GUI) for JaamSim provides all the necessary tools for model building:

- Control Panel – the main interface to JaamSim that controls the execution of models and provides access to the other GUI components
- Views – one or more windows showing 3D views of the model universe
- Model Builder – tool for dragging and dropping model components
- Object Selector – provides access to each object in the model
- Input Editor – displays the inputs to the selected object and permits editing
- Output Viewer – displays the present value for the outputs from the selected object
- Property Viewer – debugging tool that displays all the internal properties for the selected object

At the time of writing, the following palettes of objects are provided in the Model Builder:

- Graphics components – 2D and 3D components such as maps, structures, graphs, text, and model outputs
- 2D Overlay Graphics – 2D components that appear in a fixed position in a View window
- Probability Distributions – standard probability distributions and user-defined distributions
- Basic Process Flow Components – queue, server, source, sink, conveyor, etc.
- Mathematics and Control System Components – integration, addition, PID controller, etc.
- Fluid Flow Components – tanks, pipes, pumps, fluids, etc.

We frequently create new palettes of objects to support our modeling projects, many of which are added to JaamSim. More information on object palettes and the GUI can be found in the JaamSim User Manual.

3 MODEL INPUTS AND OUTPUTS

The easiest and fastest way to build a simple model is to drag and drop the individual objects and use the Input Editor to set their parameters. However, this method becomes less attractive for more complex models where the graphics become cluttered and the inputs very numerous. For a really large model containing hundreds of objects, it is usually necessary to build the model through an input file..

JaamSim allows models to be constructed either through the GUI or an input file. The input file uses an object-keyword-value structure. For example, the following lines of input:

```
Define Ship { LargeShip }
LargeShip Length { 300 m }
```

would define the object “LargeShip” (an instance of “Ship”) and assign the value three hundred meters to its “Length” input. Note that all model inputs are assigned units where appropriate and unit conversions are done automatically by the program. Internal calculations are done in SI units by JaamSim.

The Input Editor uses the same object-keyword-value structure as the input file. Figure 1 shows an example of the Input editor.

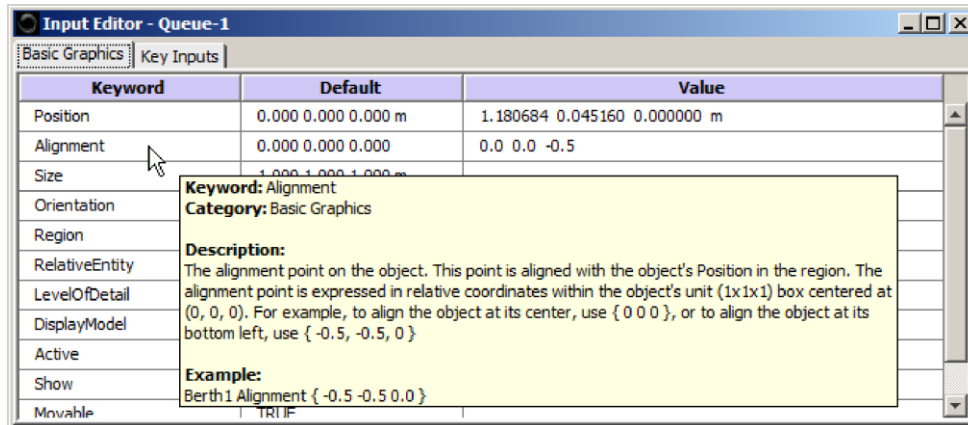


Figure 1: Input editor.

In this example, the Input Editor was opened for the object “Queue-1”, which has its keywords organized into the categories “Basic Graphics” and “Key Inputs”. The tooltip pop-up shows the definition of the keyword “Alignment” and an example of its use. Definitions are given in this way for all model inputs and outputs. The software uses annotations to encode this material, thereby making the program largely self-documenting.

Model inputs are changed by clicking on the entry in the Value column and typing a new value. Drop down menus are provided whenever possible depending on the type of input.

4 EXAMPLE MODEL

Having discussed the basics of a JaamSim model, it is appropriate to show an example. The simple model for a server and queue, shown in Figure 2, is one of the example models provided with JaamSim.

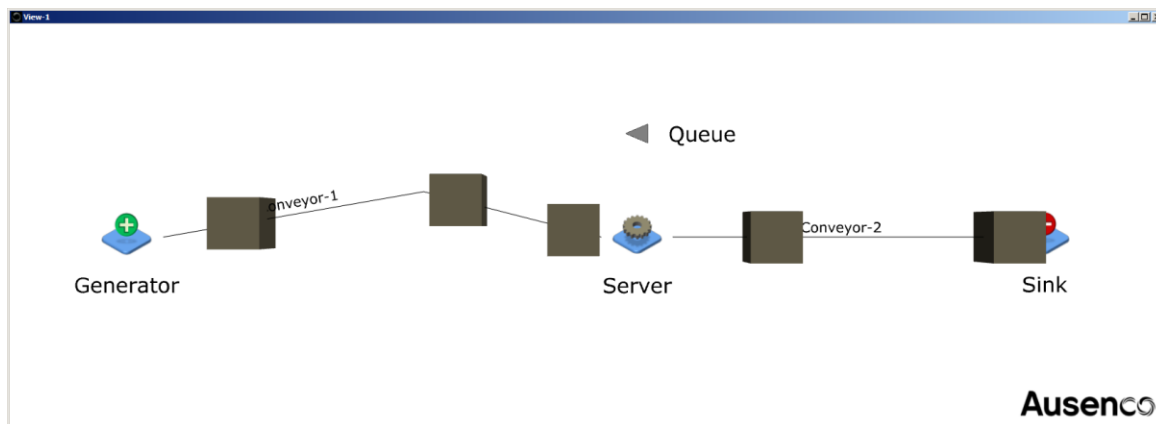


Figure 2: Server and queue example.

In this example, cube-shaped entities are created by the generator and sent via a conveyor to the server. The server processes the entities and sends them via a second conveyor to the sink where they are destroyed.

The key inputs used to create this model, excluding graphics, are given by the following entries in the input file:

```
" DEFINITIONS
Define EntityGenerator { Gen }
Define Server { Serv }
Define Queue { Que }
Define EntityConveyor { Conv1 Conv2 }
Define EntitySink { Sink }
Define NormalDistribution { IATDist ServDist }

" GENERATOR INPUTS
Gen NextComponent { Conv1 } IATdistribution { IATDist }

" SERVER INPUTS
Serv NextComponent { Conv2 } ServiceTimeDistribution { ServDist }
Serv WaitQueue { Que }

" CONVEYOR INPUTS
Conv1 NextComponent { Serv } TravelTime { 7 h }
Conv2 NextComponent { Sink } TravelTime { 5 h }

" PROBABILITY DISTRIBUTION INPUTS
IATDist UnitType { TimeUnit } RandomSeed { 1 }
IATDist MinValue { 0 h } MaxValue { 5 h }
IATDist Mean { 2.5 h } StandardDeviation { 1 h }

ServDist UnitType { TimeUnit } RandomSeed { 2 }
ServDist MinValue { 0 h } MaxValue { 3.5 h }
ServDist Mean { 1.75 h } StandardDeviation { 1 h }
```

5 3D GRAPHICS

JaamSim provides hardware accelerated 3D graphics through a built-in rendering system that was written specifically for this application. Graphics are displayed in real time and are fully interactive. The user can pause the model at any time and query an Entity by clicking on its representation on the computer screen. Model inputs and outputs can be checked using the “Input Editor” and the “Output Viewer”. If necessary, the values of all the internal variables of the Entity can be displayed using the “Property Viewer”. If execution is resumed while these tools are open, the entries are updated automatically as events are executed.

The renderer uses the [JOGL2](#) implementation of OpenGL graphics for Java. It provides modern shader-based graphics that runs efficiently on games-type graphics cards. There is no need to use the more expensive workstation-type graphics cards normally required for engineering software.

3D models can be imported to JaamSim from Collada files (.dae). Complex graphical models are typically created using 3ds Max or Maya and converted to Collada format using the appropriate [OpenCollada](#) plug-in. Engineering content created in AutoCAD can be imported directly by 3ds Max and converted in the same way.

JaamSim renderer can display fully-detailed engineering content directly from the CAD software without the manual re-working and simplification that is normally required to get an acceptable frame rate. High-quality artistic content can also be displayed. Figure 3 shows an example for a historic racing car consisting of approximately 400,000 polygons.



Figure 3: Graphics example.

Complex graphical models can be used freely in JaamSim while maintaining a frame rate of 60 frames per second (fps) with a games-type graphics card. We have stress-tested JaamSim with a scene containing 50 million polygons. A usable frame rate of approximately 10 fps was achieved for this scene using an Nvidia GTX680 graphics card.

The JaamSim renderer was designed to operate independently from the simulation logic so that its effect on execution speed is as small as possible. Each 3D window runs on a separate thread and interacts with the simulation only when it gathers the relevant information on the simulation state at the start of each rendering cycle. The reduction in execution speed caused by opening one or more graphics windows is only about 10 – 20% in most cases.

We often use a video clip to demonstrate the level of detail and quality of our simulation models to our clients. To make video clip preparation more efficient, this capability has been built directly into JaamSim. Video content is captured off-screen and to any specified resolution, independent of what is shown on the computer monitor. A capture rate of 3 frames per second is typically achieved at 1080p resolution. At this speed, a 2 minute video clip at 30 fps can be captured in less than 20 minutes.

6 DISCRETE-EVENT SIMULATION

We now give a brief description of the JaamSim simulation engine that is intended to provide the reader with enough information to understand our approach and the engine's capabilities.

Table 1 lists the basic object classes that were created to implement the discrete-event logic within Java.

Table 1: Basic simulation objects.

Object	Description
EventManager	Maintains simulated time and the list of future and conditional events to be executed.
Entity	The basic object for the simulation. Can be permanent or temporary during the simulation run and can be either active or passive in the model logic.
Process	A sub-class of thread that allows an entity to execute a series of methods in simulated time while other entities execute their own series of methods.

Note that we have separated the two objects Entity and Process. When an Entity is playing an active role in the simulation, it will have one or more Processes underway. When it is playing an inactive role or is temporarily dormant, it will have no Process underway.

Unlike most other simulation software, we have made the distinction between starting a new method, which is done in series with the original method, and starting a new process, which is done in parallel with the original method. Other simulation languages return control to the original method if the called method is halted by a wait – equivalent to starting a new process each time a method is called.

New processes can be created and started using the Entity methods in Table 2.

Table 2: Starting a process.

Method	Description
<code>startProcess(method, arg1, arg2, ...)</code>	Calls the given method with the specified arguments, i.e. <code>this.method(arg1, arg2, ...)</code> . However, it differs from a simple method call in that a new process is created, allowing the method to be executed in parallel to the original method, rather than in series.
<code>scheduleProcess(dur, method, arg1, arg2, ...)</code>	The same function as <code>startProcess</code> except that the given method is called after the specified delay. The new process is created at the end of the delay, minimizing the number of active processes.

Simulated time can be advanced using the Entity methods in Table 3.

Table 3: Scheduled and conditional waits.

Method	Description
<code>getSimTime()</code>	Returns the present simulation time in seconds.
<code>simWait(dur, pri)</code>	Stops the execution of the current method for the given duration of simulated time in seconds. Can be placed anywhere within a method and can be used multiple times within a method.
<code>while(condition) { waitUntil(); } waitUntilEnded()</code>	Code structure used to create conditional waits. Two methods <code>waitUntil()</code> and <code>waitUntilEnded()</code> are used. Stops execution of the current method until the given condition is false.
<code>simWaitLast()</code>	Stops execution of the current method until all other events scheduled for the present simulated time have been executed.

JaamSim allows both the process- and event-orientation to be used freely in the construction of a simulation model:

- `simWait` is the key method for writing a process-orientated simulation model.
- `scheduleProcess` is the key method for writing an event-oriented simulation model.

An event-oriented model is more efficient than a process-oriented model because it minimises the number of active processes and avoids context switching. However, it is easier to follow complex model logic in a process-oriented model. In a large model with complex logic, it is often the model's methods themselves that limit execution speed rather than the context switching required in a process-oriented model. The best approach is to use an event-oriented style (the `scheduleProcess` method) whenever possible and save the process-oriented style (the `simWait` method) for the more complex parts of the model. Often, it is useful to prototype new objects using `simWait` methods and then, after it works correctly, optimize selected portions by converting the code to use `scheduleProcess`.

When writing complex simulation models it is often necessary to interrupt a previously scheduled event and execute it immediately or to cancel it altogether. The Entity methods for these functions are given in Table 4.

Table 4: Interrupting and terminating a process.

Method	Description
<code>getProcess()</code>	Returns the active process executing the method.
<code>interruptProcess(process)</code>	Interrupts the given process and causes its next event to be executed immediately.
<code>killProcess(process)</code>	Interrupts the given process and terminates it.

With these few classes and methods, discrete-event simulation becomes a simple extension of the Java programming language.

7 CREATING NEW OBJECTS AND PALETTES

Our key motivation in creating JaamSim was to give the simulation modeling community an easy way to create new palettes of high-level objects that can be used in their modeling projects. To make use of this capability, the modeler must have some basic programming skills but does not have to be an expert in this field. In fact, the code for the objects in the various palettes is extremely simple and does not require much knowledge of the GUI to understand.

A good example of model code is that for the Server object, which can be found in the file “com.jaamsim.BasicObjects.Server.java”. Two methods contain the core logic for the Server: “addEntity(ent)” and “processEntities()”.

The first method, “addEntity(ent)” is called when the entity “ent” arrives at the Server and is passed to its Queue (a separate object that is stored in the property “queue”). If the Server is idle, it is re-started by calling the method “processEntities()”. The following lines of code implement this logic:

```
/**
 * Add an entity from upstream
 */
public void addEntity( Entity ent ) {
    queue.addLast( ent );
    if ( !busy ) {
        this.startProcess( "processEntities" );
    }
}
```

This method illustrates the need to start a new Process in some circumstances. In this case, a new Process is started when “processEntities()” is called. This step is necessary because the processing of entities by the Server must be performed in parallel with the process that called the “addEntity(ent)” method. If the code had read simply

```
this.processEntities();
```

the method that had called “addEntity(ent)” would not continue to the next line of code until “processEntities()” had terminated. This rule differs from many other simulation software packages that return control to the calling method as soon as a time delay is reached.

The second method, “processEntities()”, loops through the entities in the queue and processes them one by one. The method terminates when all the queued entities have been processed and the Server is set to idle.

```
/**
 * Process entities from the Queue
 */
public void processEntities() {
    busy = true;
```

```
while( queue.getCount() > 0 ) {
    servedEntity = queue.removeFirst();
    double dt = serviceTimeDistribution.nextValue();
    this.simWait( dt );
    numberProcessed++;
    this.getNextComponent().addEntity( servedEntity );
    servedEntity = null;
}
busy = false;
}
```

The “processEntities()” method demonstrates how time is advanced by the method “simWait(dt)”. Note that this method can appear in the middle of a method and within program structures such as loops, allowing for more readable code.

The Server object is connected to the GUI by the following lines in the “autoload.cfg” file that is automatically loaded when JaamSim is first launched:

```
Define Palette      { 'Basic Objects' }
Define ObjectType   { Server }
Server JavaClass    { com.jaamsim.BasicObjects.Server }
Server Palette      { 'Basic Objects' }
```

These inputs use the same format as a normal input file. The first two lines define the palette ‘Basic Objects’ and the object type “Server”. The third line identifies the Server object type with the Server class in the Java code files. The last line assigns the Server to the Basic Objects palette.

The “autoload.cfg” file is used to create nearly all the objects in JaamSim. Only a few high-level instances are created in the Java code.

8 CONCLUSIONS

This article has introduced the JaamSim software and has provided an overview of many of its features. We invite interested readers to contact the authors of this paper to obtain the user manuals, examples, and other documentation for the software.

At its present level of development, JaamSim is suitable for any model builder who is willing to program and who finds the present generation of commercial simulation software ill-suited for the intended application. If you are writing hundreds or thousands of lines of code in the proprietary programming language provided with your commercial software, you would be far better off to write your code in Java and use JaamSim.

Model builders who are unwilling to program should stay with commercial software for the time being. However, the number and variety of objects available in JaamSim will increase over time and eventually it may become an attractive alternative to commercial software for all users.

Our goal in creating JaamSim is to propel the field of simulation beyond the restrictions of proprietary simulation programming languages into the next generation of sophisticated and detailed models. We hope that the introduction of JaamSim will be a useful step in that direction.

REFERENCES

- Agosti, M. “DEUS.” University of Parma. <http://code.google.com/p/deus/>.
- Ausenco. <http://www.ausenco.com/>.
- Barr, R. “JiST – Java in Simulation Time, User Guide.” Cornell University. <http://jist.ece.cornell.edu/>.
- Buss, A. “Simkit.” Naval Postgraduate School, USA. <http://diana.nps.edu/Simkit/>.
- Collada. “OpenCollada.” <http://collada.org/mediawiki/index.php/OpenCOLLADA/>.
- GitHub. “Ausenco Simulation.” <https://github.com/AusencoSimulation/>.
- Howell, F. “SimJava.” University of Edinburgh. <http://www.dcs.ed.ac.uk/home/hase/simjava/>.
- Jogamp Community. “JOGL2.” <http://jogamp.org/>.

Kuhl, F. "Tortuga." The MITRE Corporation. <http://code.google.com/p/tortugades/>.
L'Ecuyer, P. "SSJ." University of Montreal. <http://www.iro.umontreal.ca/~simardr/ssj/indexe.html/>.
Little, M. C. "JavaSim." Newcastle University. <http://javasim.codehaus.org/>.
Mesquite Software. "CSIM." <http://www.mesquite.com/>.
Page, B. "DESMO-J." University of Hamburg. <http://desmoj.sourceforge.net/home.html/>.
Rossetti, M. D. "JSL." University of Arkansas.
http://www.uark.edu/~rossetti/research/research_interests/simulation/java_simulation_library_jsl/.
Verbraeck, A. "DSOL." Delft University of Technology. <http://sk-3.tbm.tudelft.nl/simulation/index.php/>.

AUTHOR BIOGRAPHIES

HARRY KING is the Manager of the Simulation Department at Ausenco. Ausenco is a full-service engineering company with a staff of approximately 3,500 people and head office in Brisbane, Australia. The Simulation department is based in Vancouver, Canada and has a staff of 19 professionals. Dr. King holds a Ph.D. in Theoretical Physics from the University of Texas at Austin and has devoted his career to simulation modeling since 1979. His email address is harry.king@ausenco.com.

HARVEY HARRISON is the Assistant Manager, Software, for the Simulation Department at Ausenco. He was the lead programmer for the development of Ausenco's discrete-event simulation software and has experience in many types of modeling including transportation demand models, population change models, vehicle micro-simulation and toll revenue models. He has an interest in systems programming and has many contributions to the Linux kernel. His email address is harvey.harrison@ausenco.com.