1996

Discrete Event Simulation on the World Wide Web Using Java

Buss, Arnold H.

Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

DISCRETE EVENT SIMULATION ON THE WORLD WIDE WEB USING JAVA

Arnold H. Buss
Kirk A. Stork

Operations Research Department
Naval Postgraduate School
Monterey, CA 93943-5000, U.S.A.

ABSTRACT

This paper introduces Simkit, a small set of Java classes for creating discrete event simulation models. Simkit may be used to either implement stand-alone models or Web page applets. Exploiting network capabilities of Java, the *lingua franca* of the World Wide Web (WWW), Simkit models can easily be implemented as applets and executed in a Web browser. Java's graphical capabilities enable the rapid development of intuitive user interfaces. Java's use of interpreted bytecodes, while imposing a performance penalty, enable development of platform-independent models. The language's inherent internet-awareness make other possibilities, such as distributed simulation, much easier to implement.

1 INTRODUCTION

The Naval Postgraduate School Simkit was developed to provide simple simulation tools in an object oriented computer language supported on a wide variety of computing hardware. The goal was to provide simulation tools for the analyst and researcher that:

- are accessible to analysts without professional programming skills.
- are reliable enough for moderately sized projects.
- are capable enough for real problems.
- are conducive to rapid development of exploratory models.
- are low cost in money, programmer time and resources.
- promote code sharing and reuse.
- promote model sharing and reuse.
- allow for exploration of advanced simulation concepts, such as distributed simulation and remote entities

Examination of these objectives lead to the rejection of specialized high level languages that are typically expensive and require a workstation to run. General high level languages were explored in search of an accessible, available object oriented language, resulting in the selection of Java over other prime candidates such as Objective C, C++, Smalltalk and others. Perhaps most important, Java's internet capabilities make it the only candidate capable of creating simulation models that can be run on the Web itself.

The WWW presents many new possibilities for creation, distribution, and execution of simulation models (indeed, of *all* computer models). Some of the potential uses include the following.

Models can be created and posted to a Web site so any user with a Java-enabled browser can execute the model. Since the browser executes compiled (bytecode) classes, the source coded need not be revealed. Java capable browsers, such as Netscape Navigator 2.0 and Microsoft Internet Explorer 3.0, are becoming the standard, so such simulation models will be widely available for use.

With the wider distribution of running models, a substantially larger potential user base exists. The useful feedback from this user base could be orders of magnitude larger than possible with traditional methods of model distribution.

Modelers at different locations can develop parts of the same project and post their classes for the other members of the team. With Java's ability to import remote classes, new subproject classes could rapidly be created to interface with existing classes. Since the classes themselves, rather than a copy, would be used by the other team members for testing their code, the likelihood of errors creeping in by noisy copies is reduced. Posting compiled bytecodes also helps the reliability of the overall project. By accelerating the availability of prototype classes to the project, the entire project could become more robust as well as

more timely.

Finally, Javasim supports the possibility of distributed simulation through Java's built-in internet awareness. Simulation models my be executed remotely, with different parts executing on machines that could be anywhere. Furthermore, even the client/server structure typically necessary for proper synchronization is also easily implemented in Java itself.

The remainder of this paper is organized as follows. In the next section we will give a brief overview of the Java language with emphasis on features that are key to Javasim. Next, we discuss Object-oriented design as applied to simulation programming. Section 4 describes the Simkit hierarchy and Section 5 illustrates its use in a simulation applet. Finally, conclusions and directions of future research are presented.

2 JAVA

The hype surrounding the Java language gives a casual observer the impression that it is only suitable for writing animated "applets," small programs that typically have cute but inconsequential animations. The explosion of such applets on the WWW, together with Java capability of the major Web browsers, support the notion that Java is not a "serious" language. In fact, Java is a powerful, well-designed language that is an excellent platform for developing complex, object-oriented applications.

Even a modestly comprehensive description of the Java language is obviously beyond the scope of this paper; for an excellent overview, see Cornell and Horstman (1996). However, several properties of Java made it the ideal platform for Simkit, as discussed above.

Java supports most of the language features for object-oriented programming, including classes, encapsulation, polymorphism, and inheritance. Unlike C++, Java does not support multiple inheritance, but rather borrows the protocol concept from Objective-C, called an "interface" in Java. An interface is simply the promise that a given class will implement a specified list of methods. Use of interfaces gives the modeler much more flexibility for extensions to base functionality than inheritance without giving up any power. Indeed, without interfaces the only way to combine different groups of method specifications is through multiple inheritance. It turns out that multiple inheritance is far too restrictive, particularly for the kind of flexibility required in simulation modeling. Interfaces are a critical feature in Javasim, and most of the basic abstract functionality are defined in interfaces (see Section 4).

3 DESIGN CONSIDERATIONS

There is considerable difference between programming in an Object-Oriented (OO)language and utilizing Object-Oriented design concepts (Bootch, 1996). Although an OO language may support OO design, it does not guarantee it. The design view we take is along the lines of Cox's concept of the "Software Integrated Circuit" (Cox, 1986). Cox's "software IC" concept. The fundamental components for a simulation software IC are those of the Event Graph (Schruben, 1995).

In designing Simkit, we attempted to break the act of creating a simulation model into its fundamental functional components. By utilizing available modules that provided functionality, such as Doug Lea's public-domain collections classes, we maintained a certain degree of modularity on the project.

A simulation language typically takes a certain "World View," the lens through which the underlying modeling paradigm views the model landscape. Major world views include Next-Event, Process/Resource Interaction, and Activity Scanning (Law and Kelton, 1991). The Process world view appears to be the most common among commercial languages (GPSS, SIMAN, SIMSCRIPT, etc). Implementations of the Process view involve an Event List operating in the background. Thus, while the user creates the model using high-level block diagrams, the language translates the resulting structure into a Discrete-Event model.

The Process world view allows the modeler to operate at a relatively high level of modeling. Relatively straightforward queueing-oriented models are easy to create and run. However, even modest departures from queueing models forces the modeler out of the pre-constructed blocks. Commercially available languages typically force the user to implement user-defined blocks in conventional programming languages, such as C or Fortran.

The design objective for Simkit was to achieve as much modeling ease as possible by enabling the construction of resident entities and blocking resources, while allowing the user to customize at a higher level than conventional programming. The latter was accomplished by incorporating a Discrete-Event approach based Lee Schruben's Event Graphs (1983, 1995). Thus, the user of Simkit has the choice of implementing a model using a Process orientation, a Discrete-Event Orientation, or some combination. The use of Event Graphs to create customized "blocks" is a powerful modeling capability.

4 OVERVIEW OF Simkit

Simkit consists of a programming class library for discrete event simulation written in the Java computer language. Simkit provides general code for simulation and contains specific code providing modeling capabilities of interest to the military analyst. We will addresses only the general simulation tools provided Simkit, with primary emphasis on Javasim, the core of Simkit.

Simkit consists of three Java packages, each containing a set of related classes that work together to provide functionality in a key area to support the model. Graphical User Interface elements are provided by the classes in the awt package. Utilities, such as data structures, random variate generation, and statistics collection, are implemented in the util classes. Event-driven simulation is facilitated by the classes in the Javasim package. The packages form a hierarchy, much like a class inheritance hierarchy, which is shown as a directed acyclic graph in Figure 1.

The Mil package is an extension to Javasim that provides tools for building simulation models with particular application to military problems. In addition to the intrinsic interest in such models, Mil can be seen as a prototype that demonstrates the extensibility of Javasim. The design of Javasim enables modelers to extend the language itself to incorporate their particular needs. This extensibility makes Javasim an attractive alternative to special-purpose simulation languages which tend to be more restrictive.

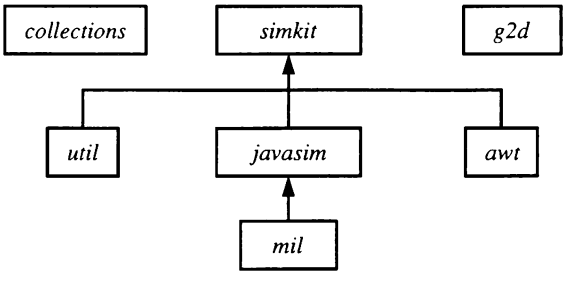

Figure 1: Simkit Package Hierarchy

Classes in one package can use classes in other packages. In fact, Javasim relies on classes in each of the other packages, each providing a distinct set of services to the programmer. We will now briefly describe each package. In the spirit of OO design, we have utilized existing classes to as great extent as possible, specifically the Collections and g2d packages.

4.1 Collections and G2d

The collections package, generously contributed to the public domain by Doug Lea, provides a very complete set of data structures. This package is completely independent of Simkit and is useful in many contexts. Simkit uses collections for most of its data structure needs. Complete documentation for collections is provided with the source code distribution available on the internet.

Charting capabilities are provided by the g2d (Graphics, 2-dimensional) package. Minor modifications were made to the original Graph2D classes provided by Leigh Brookshaw to give them their own identity and to simplify the programming interface for Simkit. The package itself is quite complete, and can be used independently of Simkit. Documentation for the original Graph2D is included with the source code distribution on the internet. The modified code making up the g2d package as shown here is also available on the internet, but the changes are slight. We recommend getting the most recent source code from the original author for uses other than Simkit. While not in the public domain, the g2d package is covered by the GNU General Public License (1991).

4.2 Awt

The classes in the awt package provide a limited set of pre-built graphical user interface elements used by Javasim. The main features of awt include simple windows to display tables and graphs. Awt relies on g2d for its lower-level functions. Several classes originally presented in the book Core Java (Cornell and Horstman, 1996) are used to provide formatted number output and self verifying text input boxes.

4.3 Util

The util package contains classes that give support functionality to the simulation model. These include data collection, random variate generation, and data structures that gather summary statistics on usage.

4.4 Javasim

Javasim is the core of Simkit and provides all the general discrete-event simulation facilities in Simkit. Although incomplete in comparison to commercial simulation environments, Javasim provides the most of the needed functionality for simulation modeling.

The structure of Javasim is shown in Figure 2. There are three main layers to the overall design: The core Javasim classes, an Extension layer, and a User layer. As Figure 2 indicates, Javasim consists of two

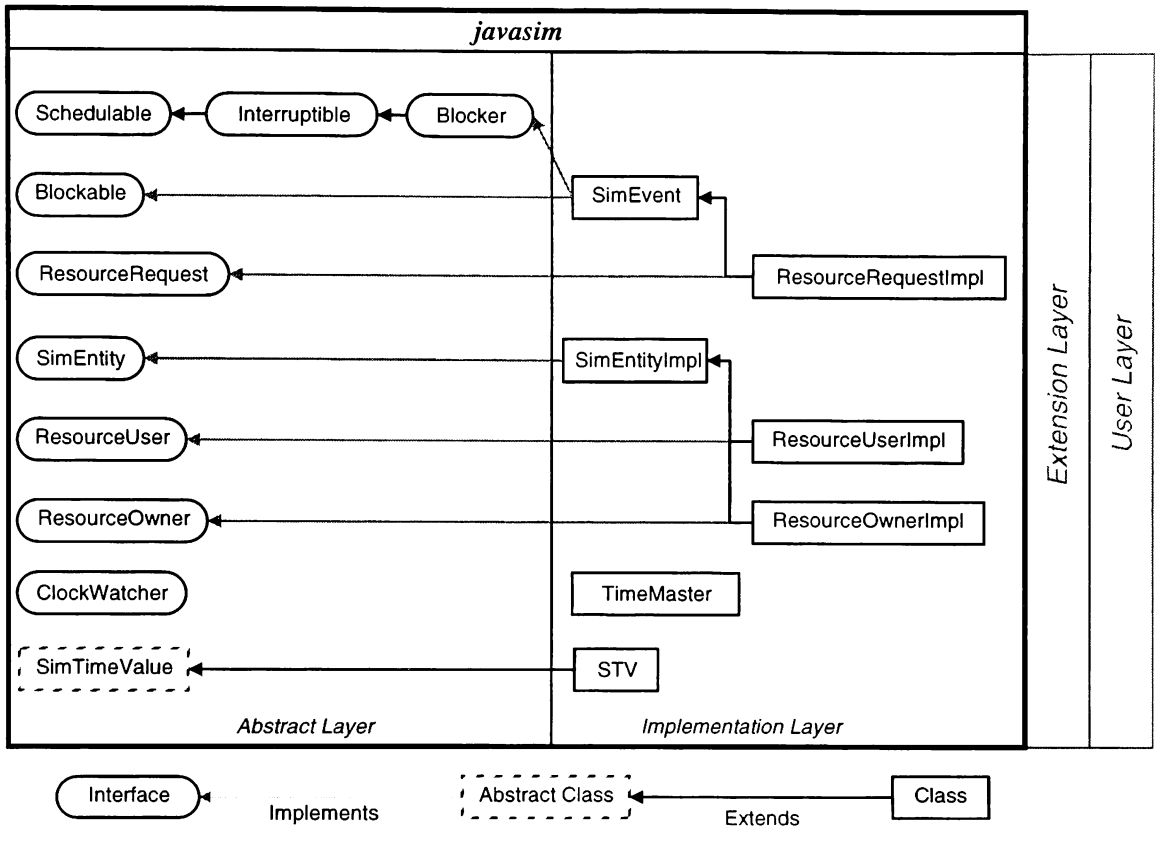

Figure 2: Javasim Design

components: an "Abstract" layer and an "Implementation" layer. The Abstract layer defines functionality mostly through Java Interfaces. Each interface only provides a list of methods that are to be implemented by classes in the Implementation layer. The relationship between interfaces and the classes that implement them is much weaker than the inheritance relationship. Specifically, a class which implements an interface does not have an "is-a" relationship to the interface. Rather, the class simply promises that it will have the interface's methods in some manner. Interfaces are thus identical to Objective-C protocols. The one "Abstract" class, SimTimeValue, provides the timekeeping in a robust manner.

The basic discrete event framework in Javasim is provided by the Timemaster class, which manages the event list. The execution and scheduling of events is done by the SimEvent and SimEntityImp classes, respectively. The Interruptable interface allows the possibility of events being cancelled. The STV class implements simulated time. These classes are sufficient to create any discrete-event simulation models.

The remaining three classes, ResourceRequestImpl, ResourceUserImpl, and ResourceOwnerImpl, together implement the concept of blocking resources, allowing some degree of creating Process/Resource models with Simkit.

Using interfaces rather than an inheritance hierarchy allows Javasim to easily incorporate the Extension Layer. While a modeler will not change anything in Javasim itself, there may still be a need for special-purpose, customized tools. For example, the Mil extension supports the ability for the modeler to easily create such entities as ships, airplanes, submarines, etc. Furthermore, such interactions between model entities, such as sensing and detection, can be created as as part of the language itself by use of the extension layer.

4.5 Mil

Mil is a simulation framework for simulating scenarios of interest to military analysts. In addition to being an example for the Extension layer of Simkit, Mil provides functionality for such military applications as ship-missile defense (Stork, 1996).

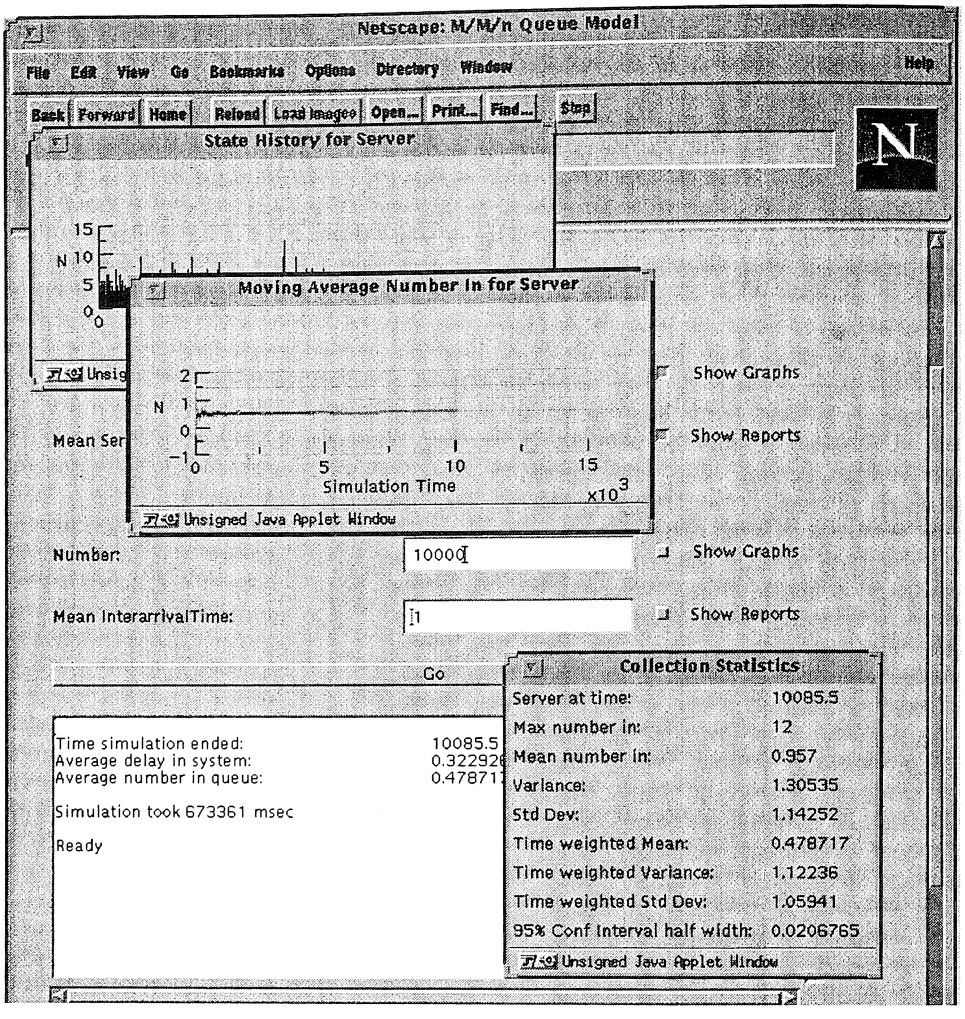

Figure 3: Output of MMn Applet

5 EXAMPLE

We illustrate a model in Simkit with the simple $M/M/n$ queue as an applet. A screen shot of the applet running in Netscape is shown in Figure 3. The user interface is easily implemented and allows parameter input from text boxes and execution with a simple button. Check boxes allow the user to decide at run time to display choices of summary statistics and graphs of output.

To illustrate how Simkit is be used for a Process/Resource view of the model, we show the source code for the ResCustomer class below.

```
class ResCustomer extends ResourceUserImpl {
  private double myServiceDelay;
  private Time myServiceRequestTime;
  private ResourceOwnerImpl myServer;
  private Histogram delayStat;
  private double meanST;

  public void setServiceTime(double d) {
    myServiceDelay = d;
  }

  public void getService(
                  ResourceOwnerImpl server,
                  Histogram reportTo) {
    myServer = server;
    delayStat = reportTo;
    myServiceRequestTime = TimeMaster.SimTime();
    getResources( myServer, 1L );
  }

  public void receiveResources(
                        ResourceRequest res){
    super.receiveResources(res);
    delayStat.getSample(
      ((TimeMaster.SimTime()).value()) -
```

```
        ( myServiceRequestTime.value()));
    ResEndService e = new ResEndService( this );
    e.waitDelay( new Time(myServiceDelay) );
  }

  public void doEndService() {
    returnResources(myServer);
  }
} // END CLASS Customer
```

The ResCustomer class provides the core functionality of the model. The methods are mostly self-explanatory, representing the customer requesting and subsequently receiving the server resource, waiting for the service time, and relinquishing the server. This class also illustrates the use of the Time class wrapper. All variables representing time in Simkit are declared to be an instance of Time. This abstraction of the concept of a time adds robustness to Simkit models.

6 CONCLUSIONS

Simkit is a class library written in Java that provides core functionality for the creation of simulation models. While relatively small, Simkit is nevertheless is sufficiently powerful for many modeling needs. Its user layer allows Simkit to be used as-is for creating simulations, and its extension layer allows language extensions at a level substantially higher than base-level code. Unique network-capable features of Java can be exploited to enable straightforward implementation of simulation applets and distributed models.

ACKNOWLEDGEMENTS

Support for this work from the Naval Postgraduate School is gratefully acknowledged. Portions of this work were part of the second author's Masters Thesis in Operations Research at the Naval Postgraduate School.

REFERENCES

Bootch, G. 1996. *Object Solutions: Managing the Object-Oriented Project*, Addison-Wesley, NY.

Cornell, G. and C. Horstman. 1996. *Core Java*, Sunsoft Press, Mountain View, CA.

Cox, B. 1986. *Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, NY.

Flanagan, D. 1996. *Java in a Nutshell*, O'Reilly & Associates, Inc., Sebastopol, CA.

GNU General Public License. 1991. The Free Software Foundation, Cambridge, MA.

Law, A. and D. Kelton. 1991. *Simulation Modeling and Analysis, Second Edition*, McGraw-Hill, NY.

Schruben, L. 1983. Simulation Modeling with Event Graphs, *Communications of the ACM*, **26**, 957–963.

Schruben, Lee. 1995. *Graphical Simulation Modeling and Analysis Using Sigma for Windows*, Boyd and Fraser Publishing Company, Danvers, MA.

Stork, K. 1996. A Simulation Study of Countermeasure Effectiveness Against Anti-Ship Missiles. Masters Thesis, Department of Operations Research, Naval Postgraduate School, Monterey, CA.

AUTHORS BIOGRAPHIES

ARNOLD BUSS is a Visiting Assistant Professor of Operations Research at the Naval Graduate School. He received his PhD in Operations Research from Cornell University, and is a member of INFORMS, IIE, POMS, and MORS. His research interests include Object-Oriented simulation modeling, manufacturing applications, and project management.

KIRK STORK is a Lieutenant in the United States Navy. After enlisting in 1981 and completing the Navy's Nuclear Power Training, he attended the University of Washington and obtained his BSE and a commission in the Navy in 1989. He received his MSOR at the Naval Postgraduate School and is currently attending the Navy's Submarine Officer's Advanced Course in Groton, CT.