DISTRIBUTED, OPEN SIMULATION MODEL DEVELOPMENT WITH DSOL SERVICES

Niels A. Lang
Erasmus University Rotterdam
Rotterdam School of Management
Dep. of Decision & Information Sciences
Burg. Oudlaan 50, 3000 DR Rotterdam, NL
nlang@fbk.eur.nl

Peter H.M. Jacobs and Alexander Verbraeck
Delft University of Technology
Faculty of Technology, Policy and Management
Systems Engineering Section
Jaffalaan 5, 2628 BX, Delft, the Netherlands
p.h.m.jacobs@tbm.tudelft.nl; a.verbraeck@tbm.tudelft.nl

KEYWORDS

Distributed simulation, object-orientation, simulation services, simulation environment.

ABSTRACT

Information technology innovations, like web-services and object-oriented design, are fast finding their way into Business Information Systems (BIS). At the same time similar developments in the field of business system simulation do not gain momentum. This observation, combined with research needs, has been the starting point for the development of an open, object-oriented, distributed and extendible research test-bed for business simulation, called DSOL. DSOL, which stands for 'Distributed Simulation Object Library' was introduced on Wintersim '02 (see Jacobs, Lang and Verbraeck 2002), and has now evolved into a firm distributed simulation core, extended with several services for visualization, event notification and BIS integration.

This paper illustrates in detail the way in which concepts and principles on discrete-event, multi-formalism simulation have been translated into the object-oriented, distributed DSOL environment. It introduces the simulation concepts underlying the DSOL design, the DSOL implementation itself, example models and finally discusses the outcomes.

SIMULATION FOR BUSINESS DESIGN

Recently, several internet web service based architectures have been introduced that promise to streamline inter- and intra-organizational communications. These developments allow organizations to integrate their distributed business systems fast and effective and, as a result, improve performance of and control over their business processes.

A corresponding movement towards open, distributed interaction standards has not yet gained momentum in the field of simulation. Most simulation packages have limited and incomplete capabilities for interaction with other systems, effectively limiting the potential of linking simulations to other types of information systems, such as ERP systems and databases. Moreover, standards for interoperable simulation systems are not yet widely accepted, limiting the development of multi-model simulation systems.

The Java-based Distributed Simulation Object Library (DSOL, see Jacobs, Lang and Verbraeck 2002) was developed to fulfill the need for such an open integrated platform for business simulation. Section 1 describes the simulation concepts underlying the platform. Section 2 presents its current implementation. Section 3 presents the *M/M/1* queue example. Section 4 finally discusses the results achieved thus far.

We regard simulation to be a model-based approach for ill-structured problem solving (Sol 1982), a model being *a representation of a part of reality, constructed for a particular purpose.* Simulation has been defined by Shannon (1975) to be:

'The process of designing a model of a concrete system and conducting experiments with this model in order to understand the behaviour of a concrete system and / or to evaluate various strategies for the operation of the system.'

We furthermore distinguish between *conceptual models*, that provide a problem-contingent language for system model design and *system models*, instantiations of the conceptual model describing a part of the system of interest. The system models allow experimentation. Describing a system model in terms of a conceptual model eases its description (many system model parts can be related to a single concept) and allows for the definition of consistency rules. Such rules constrain the model designs allowed and therewith ease model

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

understanding.

While our definition of simulation speaks of the design of '*a* model', we follow Simon 1973 in that ill-structured problems by their very nature require the development of *multiple* models. The overall ill-structured problem solving process does not proceed in a straightforward manner, but proceeds by alternating continuously between model development for (chosen) sub-problems and sub-model integration. Changes may not only occur in system models but also in the underlying conceptual model, yielding the requirement for flexible modeling of both.

The simulation approach described thus far could be applied in many fields. We will in this paper, however, focus on its appliance in the field of business systems. Current business systems have three characteristics that we deem relevant for the simulation approach.

First of all, there is little doubt on the *complexity* of many current business systems. Business processes easily involve hundreds of steps, complicated planning and scheduling decision procedures and thousands of distributed resources.

Moreover, businesses are globalizing since the dawn of the 20th century (Acs, Morck and Yeung 2001), under the drivers of market liberalization and cheap communication infrastructure. This trend implies that business design problems essentially has become *distributed* in two ways: not only the object of design (e.g. the business system) has become an inherent distributed system, the process of design, with typically involves many stakeholders worldwide, is moving towards distributed collaboration as well.

Finally, the trend of automation and information drives the *virtualization* of business systems. Business process organization and management is more and more realized in complex ICT systems (such as Enterprise Resource Planning (ERP), Supply Chain Management (SCM) and many others), which allow for (computationally) complex business processes and offer global business connectivity.

As a result, a simulation environment supporting simulation for the current business environment should pay explicit attention towards the aspects of system complexity, distributed systems and the presence of virtual business information.

The remainder of the section presents a conceptual framework for simulation environments which fits the described peculiarities of business simulation.

Simulation as experimenting

Figure 1 introduces an experimentation oriented framework of the simulation environment, in line with our definition of simulation. The framework is based on

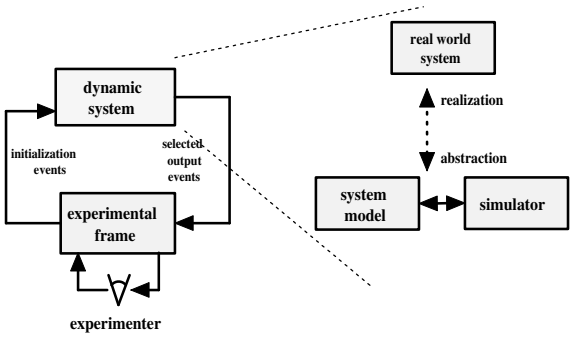

Figure 1: Experiment oriented framework for simulation

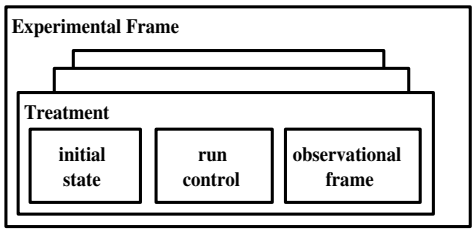

Figure 2: The experimental frame defined

Zeigler, Praehofer and Kim (2000) and introduces two main concepts, that of 'experimental frame' and 'dynamic system'. According to Zeigler, the experimental frame is a *'specification of the conditions under which the system is observed or experimented with'*. The system being experimented with is basically any dynamic system, with the experimental frame defining and limiting the scope of the observations with regard to time, region and aspect. In the context of simulation the dynamic system is either a concrete system, or a corresponding model. We call an experimenter developing and experimenting with several models (in order to improve the concrete system's behaviour) a *designer*. In [Verbraeck / Sol] the concept of experimental frame is defined more precise, by regarding it as a collection of *treatments*, where each treatment comprises an initial state, run control parameters and an observational frame, see figure 2. Using this definition, it is possible to define roles other than experimenter or designer. We define an *observer* as a participant interacting only via an observational frame. We finally define a *player* as a participant interacting using an observational frame and an *action frame*, which allows the player to provide input to the dynamic system during an experiment. The last role essentially turns the simulation model into a game model.

With respect to business simulation, we observe that the framework allows concurrent, distributed collaboration, since no restriction is imposed on the communication used or the number of participants concurrently

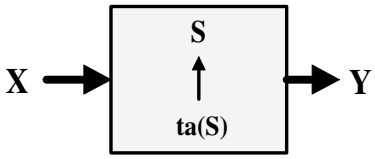

Figure 3: The DEVS model formalism

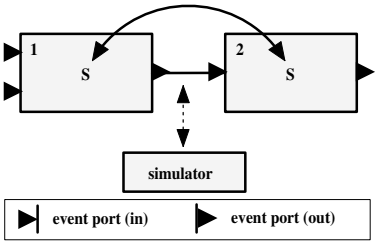

Figure 4: The extended DEVS formalism

involved. We finally recall the 'virtualization' of business systems. This basically implies that real world realizations of business ICT systems do already contain models and data of the business system they support. In potential such models and data could considerably speed up the development of a system model of the business system, if services to convert them into a simulation model are made available. A typical example of this would be to retrieve workflow processes and data from a company's ERP system and use this as the basis for a workflow simulation model.

DEVS: a base formalism

We distinguish between simulation model and simulator (Zeigler, Praehofer and Kim 2000), illustrated in figure 1. Whereas the model defines structure and behaviour, the simulator actually brings the model behaviour about by *executing* the model in a time controlled manner. The simulator allows the experimenter to control speed, start and end times, by discretionizing the overall behaviour into events, e.g. atomic state-changes. The separation between model and simulator raises a need for a *contract* defining the interaction between model and simulator. We'll define a formally defined contract to be a model formalism. It follows that a simulator for a given formalism is able to execute all models specified in that formalism.

In business simulation, many types of models are used, ranging from complex control systems, to simple workflow models and to distributed gaming models. For integrated business analysis, it is often desirable (but not always feasible!) to integrate such models of different but related business system aspects. To ensure the applicability of the DSOL simulation environment for these situations, a choice was made for a base formalism of high expressive power (Zeigler, Praehofer and Kim 2000; Vangheluwe and de Lara 2002): Discrete Event System Specification (DEVS).

Figure 3 schematically introduces the DEVS concept, which is presented here with an emphasis on illustration, not on rigor. **X** defines the set of input values, **Y** defines the set of output values. The model has a state **S**, which is a function of an incoming value and of the internal transition function. The latest is triggered by the function **ta(S)** which defines the time for the next internal state transition. The model output (**Y**) is defined to be a function of the model's state only.

The formalism introduced thus far imposes no constraints on the semantics of input, output or state. It can be extended to encompass specific worldviews like transaction-flow or actor oriented ones.

Modularity and hierarchy

The DEVS formalism as introduced thus suggests the model to be a monolithic entity. As such, it is still of little use to us, since a monolithic model will be as complex (or worse) as the part of reality being modeled. The DEVS formalism has, however, been extended with general systems concepts enabling the description of multi-component, modular and hierarchical DEVS models. This extended DEVS formalism is illustrated in figure 4.

It introduces the concept of a *multi-component* model connected by *event ports*. Between event ports, *connections* can be attached, allowing components to interact by the exchange of events. Actual event propagation is facilitated by the DEVS simulator, which is in this way able to control the speed of inter-component interaction. The formalism does not impose any restrictions on how event interaction is actually realized thus allowing for distributed implementations.

The possibility to break down the model in several components eases model development and enhances understanding. The concept of event ports furthermore allows for *modularity*, e.g. components managing a self-contained, not externally visible state. By restricting component interaction only to take place via event connections, components in effect become modules, with event ports defining their interfaces. In such a setting the direct interaction (bypassing event connections) shown in figure 4 is thus forbidden. The well-known advantages of modular models is that they allow for separation of concerns and freedom of implementation: essential qualities for complex business system modeling.

Finally, figure 5 illustrates the suitability of the DEVS formalism to construct models containing several levels of abstraction, e.g. hierarchical models. Modularity is maintained by restricting internal mod-

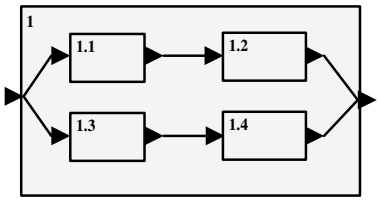

Figure 5: Hierarchical DEVS

els to connect only to event ports of or internal to the parent model. The relevance for business system modeling is that business system design occurs at different levels of abstraction, from individual workplace design to inter-organizational coordination (Sol 1982).

Towards multi-formalism simulation

The extendability of the DEVS formalism has already often been demonstrated (Zeigler, Praehofer and Kim 2000; Vangheluwe and de Lara 2002). However, many simulation packages, although often based on DEVS, only provide access to a higher level set of concepts, such as the transaction-flow worldview. Given the already mentioned business problem diversity the challenge is to realize an environment which is open to new formalisms (realizing conceptual freedom). A sound basic formalism and operators for its extension are therefore essential conditions. We hold DEVS to be such a formalism.

SIMULATION WITH DSOL

This paper introduces DSOL: a Java based research test-bed for simulation in object oriented, distributed environment. Before we introduce DSOL's implementation of the fundamental DEVS concepts, we start here with an overview of the services provided by the framework. Currently DSOL consists of:

- A core DEVS simulator introduced throughout the rest of this section. From a perspective of information system design, this simulator is set up as a remotely accessible service. It fully supports all current enterprise information system standards.
- A basic statistics library consisting of a number of pseudo random number generators, tallies, charts, counters, etc (see Law & Kelton 2000). Planned extensions include dynamic input analyzers, monte carlo analysis and the integration with data mining suites.
- A 2D visualization and representations described in Jacobs, Lang and Verbraeck (2002). 3D animation is currently tested based on Sun's 3D API.

Remote method invocation scheduling

In the previous section we focused in great detail on the DEVS formalism and pointed out that this formalism is based on scheduled interaction. The challenge of the DSOL framework is to see to what extent an object oriented programming language like Java supports this principle of interaction and how it is able to schedule it in a profound and natural way.

The synergy between an object oriented language and the DEVS paradigm becomes clear when we consider that both are based on a principle of interaction. In an object oriented programming language, objects interact by the invocation of methods. An object oriented programming language furthermore distinguishes *public*, *private* and *protected* methods. Where private methods are only accessible to the object itself, protected methods are also accessible to instances of subclasses (in the Java programming language protected methods are also accessible to all classes in the same package) and public methods are accessible to everyone.

For us it became clear that the most essential part of designing a DEVS based simulator was to support this notion of object oriented interaction; instead of the direct method invocation which occurs in normal Java programming, DSOL schedules method invocation based on the following requirements:

- The scheduled method invocation must map on the scheduled state change illustrated in figure 3. A *SimEvent* illustrated in figure 6 maps the event introduced by the DEVS framework.
- All public methods must be able to be scheduled and invoked by the DSOL simulator. There may be no constraint on the method name, return type or any argument.
- Overloading and polymorphism must be supported. This results in a simulation framework fully supporting all potential distinctions between methods.
- Though both the simulator and the object on which interaction is scheduled may reside within the same Java virtual machine, DSOL is designed for distributed interaction. All simulation events must therefore be serializable.

The modular approach introduced in the first section encouraged us to accept that potentially better implementations of the scheduled method invocation may be developed. DSOL nevertheless provides a reference implementation based on all above requirements (figure 6).

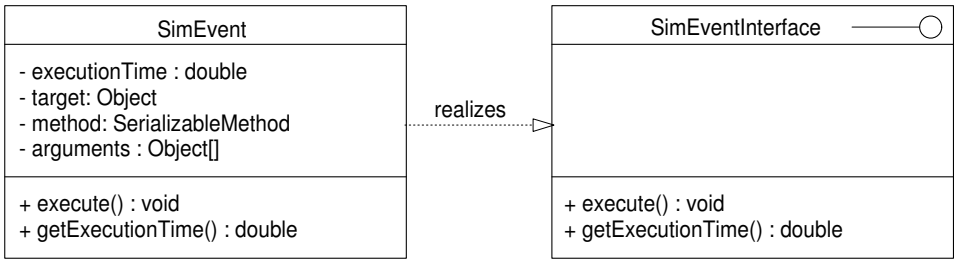

Figure 6: Reference implementation of a DSOL simulation event

Process modeling in an object oriented simulation framework

In section 1.2 we elaborated on the complexity of conceptualizing an ill structured real-world business system under investigation. We furthermore emphasize on Banks' (Banks, 1996) recommendation of involving the problem owner in this activity. Finally we conclude that this often leads to a resource based conceptualization of the real-world business system. This section focuses on ways to support the translation of these resource based concepts, e.g. resources, queues and flows, into a DEVS based specification framework like DSOL. There are several options to accomplish this translation:

The first option discussed here is the NULL option. We just do not provide any assistance in the translation of resource based conceptual models in DEVS based specification framework. Though Zeigler (2000) proved that all resource based concepts can fundamentally be explicated in DEVS concepts, this option leaves it is up to the designer to make this translation while coding the specification model.

A second option is to provide a library which enables the model builder to code the process in the entities of the specified simulation model. This is an approach followed by most Java based simulation frameworks such as Silk, SSJ, SimJava, etc. The following pseudo code reflects this approach.

```
public class Customer extends entity
{
 public void process()
 {
  server.request(1.0);
  //request a capacity of 1.0
  this.delay(1500); //suspends for t=1500;
  server.release(1.0);// resume and release
 }
}
```

Key consequences of this approach in an object oriented programming language are:

- The conceptual model is specified in long procedural code. For large models, this is in clear contrast with the required modularity and hierarchy supported both by the DEVS formalism and the Java programming language.
- The customer in the above pseudo code must be able to suspend himself for a particular time. In order to accomplish this the entity must break out a method and afterwards resume to the next line within this method. In an object oriented programming language this can only be accomplished by extending a thread (Healy 1998; Garrido 2001). This multi-threaded approach is in the Java programming no longer supported and considered deprecated (Sun 2002). The approach is inherently unsafe; arbitrary behavior can result, which may be subtle and difficult to detect, or it may be pronounced. We therefore state here that the results of simulation models based on this approach are fundamentally un-trustworthy.

The third approach presented in this paper is to design and develop a library for resource based conceptual models. This approach is implemented by most non object oriented simulation frameworks such as Arena, EM-Plant, Automod and a few object oriented frameworks such as SimKit and DSOL. In this approach one designs the model as a chain of stations. The following pseudo code reflects this approach:

```
public class SimulationModel
{
 public static void main()
 {
  //create a generator with interarrivaltime of
  //1.2 and a batchSize of 1
  StationInterface generator =
         new Generator(Customer.class,1.2,1);

  //create a delay of 1500 time units
  StationInterface delay = new Delay(1500);

  //create an exit station
```

```
  StationInterface exit = new Station();

  //now we create the flow
  generator.setDestination(delay);
  delay.setDestination(exit);
 }
}
```

Key consequences of this approach in an object oriented programming language are:

- Since a simulation model is specified by the creation of a chain of stations , this approach keeps as close as possible to resource based or process oriented, e.g. IDEF-0, conceptual modeling languages.
- Since the model is programmed in the stations it is easier to replace a station by an extension. Specifying a model is done in a service oriented approach supporting both Java's component based and DEVS modular and hierarchical design patterns .
- Since incoming entities are scheduled from station to station there is no need for the suspension of threads. An entire model may be single threaded and entities can easily be serialized and streamed from computer to computer.

DSOL has followed both the first NULL approach and the third approach. The reason why this paper mentions the first NULL approach is that though DSOL provides a library of resource based concepts, they are considered an add-on. The underlying DEVS framework remains directly accessible.

A-synchronous event model

In order to emphasize the advantages of implementing a DEVS formalism in an object oriented language this section illustrates the benefits of using an asynchronous event mechanism.

The asynchronous event mechanism consists of two sides: a listening client subscribes to a topic and an event producing publisher notifies all subscribed listeners on a particular change. The interfaces and reference implementation are illustrated in figure 7.

In figure 7 a *ListenerInterface* provides a method on which a callback is made possible and the *EventProducer* consists of a private *List* containing all subscriptions. A subscription consist of a listener and a topic. Whenever a producer invokes a *fireEvent* method, all listeners are one by one matched on the topic and if required notified.

Though the above elaboration on an asynchronous event mechanism might sound all too familiar to experienced Java programmers, it is good to understand and emphasize on its consequences:

- An asynchronous event notification mechanism supports point-to-multipoint interaction between model components and distributed (web-based) representation components.
- An asynchronous event notification mechanism supports dynamic soft coded relations between deployed (web) services.

EXAMPLE: AN *M/M/1* QUEUE

This section derscribes how to implement the traditional *M/M/1* queue: a convenient example to illustrate DSOL. It is furthermore used as an example by L'Ecuyer [L'Ecuyer, 2002] to illustrate SSJ. Since the SSJ framework is a multi threaded process oriented simulation framework, the combination of papers illustrates the difference of specifying the resource based conceptual model in entities versus stations.

In the *M/M/1* queue, the service time follows a normal distribution with μ=0.8 and σ=0.1. The system starts empty and has a runlength of one million time units. Entities are created with a batchsize=1 and an interarrival time following an exponential distribution with β=1.0.

Figure 8 illustrates the code for this system. First of all we see the creation of a simulator. The next step is to set the runlength of this particular simulator. Then we create a random stream to be used within the application. For this particular example we have used the Mersenne Twister random stream developed by Makoto Matsumoto [Matsumoto, 1998].

The first resource based building block used within figure 8 is the generator. The generator creates instances of a class by the scheduled invocation of its appropriate constructor. In order to deal with constructor overloading we submit besides the name of the class, the array of arguments for the constructor and the array of classes of which these arguments are instances. We also provide distributions describing a start time, an interarrival time and a batchsize. The fact that we generate instances of the *java.lang.Object* class emphasizes that there are no restrictions within the DSOL framework to the postponed invocation of methods or constructors.

The next steps are very straitforward. First we create a resouce with a capacity of 1.0. Then we create a seize block which claims 1.0 unit of the resource. After the resource is succesfully claimed, the entity will flow to the server which will delay the entity for around 0.8

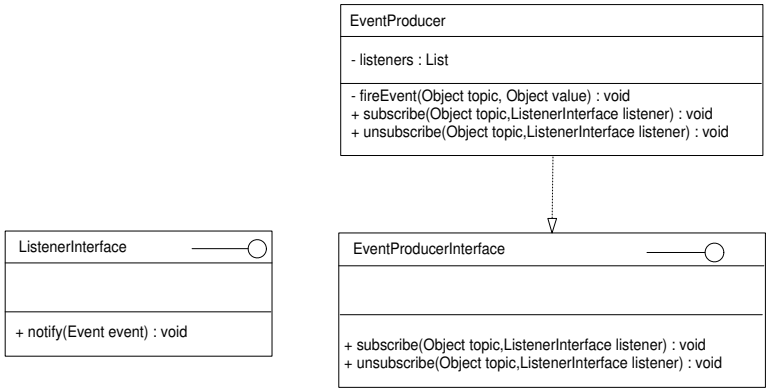

Figure 7: Reference implementation of DSOL's asynchronous messaging

time units and release the resource. Finally we initialize and start the simulator.

DISCUSSION AND CONCLUSIONS

This paper started with the need for an interaction between simulations and business systems. The problem with this integration is of course that these two worlds differ very much. Where simulations abstract and reduce reality, business systems try to fully capture that reality. The simulation clock runs faster than reality, but the business system has no choice but to operate in real time. On the other hand, there *is* a clear need for tight integration, because the most time consuming tasks in simulation studies are the mapping of the business system on the simulation and the process of gathering and preparing data for the simulation experiments. By using the asynchronous interfacing mechanisms that was introduced in section 2.3, data collection from real-life data sources is much easier than trying to create a tight interface between business systems and simulations. In addition, there does not need to be such a big difference between *delayed* method invocation and *real* method invocation. Architectures for creating the business systems can therefore also be used for simulating these systems, where the most important differences consist of the reduction of certain processes and the scheduling against artificial time. A welcome add-on is the possibility to create hybrid systems where simulation models are a component in a larger system, e.g. providing decision support functionality. Several projects have already been carried out where DSOL simulations easily exchange information with databases, spreadsheets, and external systems. Two interesting challenges will be researched in the near future: directly mapping the architecture of a business system in reduced form on a simulation architecture, and including DSOL simulation models as modules in decision support systems.

The DSOL framework as introduced offers more than a possibility for integration, it is also a basic DEVS simulator with many extensions such as statistics, animation and visualization, and process modelling. In contrast to many other approaches, though, these extensions are not tightly coupled to the DSOL core, but loosely coupled, enabling other implementations of these additions as well. The heavy use of Java interfaces in the implementation ensures that users extending or creating their own implementation of a certain functionality, do not have to change pieces of code that make use of the newly created software. It is, for instance, very easy to change the SimEvent or EventList in the core of DSOL for another implementation without making any changes to projects that have already been built on top of the DSOL core. The fact that DSOL is an open source project will hopefully stimulate specialists to indeed create better or faster implementations of some of the already implemented functionality of this Java-based simulation framework.

OBTAINING THE SOFTWARE

DSOL is published under the General Public Licence. More information on the license can be found at `http://www.gnu.org/copyleft/gpl.html`. The DSOL project description can be found at `http://www.simulation.tudelft.nl` and the software can be downloaded from `http://sourceforge.net/projects/dsol/`.

```
public class MM1Queue
{
 public static void main(String[] args)
 {
  if(args.length!=0) System.out.println("Usage: java MM1Queue");

  SimulatorInterface simulator = new Simulator();
  simulator.setRunLength(1000000);

  StreamInterface randomStream = new MersenneTwister(555);

  //The generator
  DistContinuous generatorStartTime = new DistConstant(randomStream,0.0);
  DistContinuous arrivalTime = new DistExponential(randomStream,1.0);
  DistDiscrete batchSize = new DistDiscreteConstant(randomStream,1);
  GeneratorInterface generator = new Generator(simulator,java.lang.Object.class,null,null,
                                 generatorStartTime,arrivalTime,batchSize);

  //The queue and server
  Resource resource = new Resource(simulator,1.0);
  StationInterface queue = new Seize(simulator,resource,1.0);

  //The server
  DistContinuous serviceTime = new DistNormal(randomStream,0.8,0.1);
  StationInterface server = new Delay(simulator,serviceTime);

  //The flow
  generator.setDestination(queue);
  queue.setDestination(server);

  //Starting the model
  simulator.initialize();
  simulator.start();
 }
}
```

Figure 8: M/M/1 Queue

REFERENCES

Acs J., R.K. Morck and B.Yeung. 2002 *Entrepreneurship, globalization and public policy*, in Entrepreneurship, globalization and public policy, vol. 7 (2001), pages 235 - 251.

Banks J. , J.S. Carson II, B.L. Nelson. 1996 *Discrete-Event System Simulation.* 2nd ed., Prentice Hall, Upper Saddle River, N.J.

L'Ecuyer P., L. Meliani, J. Vaucher. 2002 *SSJ: A framework for stochastic simulation in Java.* Conference paper Winter Sim'02 conference.

Garrido J.M., 2001 *Object-Oriented Discrete-Event Simulation with Java*, Kluwer Academic/Plenum Publishers, New York.

Healy K.J. and Kilgore R.A.,1998 *Introduction to silk and java-based simulation.*

Jacobs P.H.M., N.A. Lang, A. Verbraeck. 2002. *DSOL; A distributed Java based discrete event simulation architecture.* Conference paper Winter Sim'02 conference.

Law A.M., W.D. Kelton. 2000. *Simulation modeling and analysis*, 3th ed., McGraw-Hill, New York

Matsumoto M., T. Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 1998

Shannon R.E., 1975. *Systems simulation: the art and science*, Prentice-Hall

Simon H.A., 1973.*The structure of ill-structured problems* in Artificial Intelligence, vol. 4 , pages 181-202

Sol H.G., . 1982. *Simulation in information system development.*

Sun Microsystems Inc., 2002 *Why Are Thread.stop, Thread.suspend, Thread.resume and Runtime.runFinalizersOnExit Deprecated?*, `http://java.sun.com/j2se/1.4.1/docs/guide/misc/threadPrimitiveDeprecation.html`

Vangheluwe H. and J. de Lara, 2002.*Meta-models are models too* in Proceedings of the 2002 Winter Simulation Conference, E. Yücesan, C.-H. Chen, J.L. Snowdon and J.M. Charnes, ed., pages 597 - 605.

Zeigler B.P. , H. Praehofer and T.G. Kim. 2000. *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems.* 2d ed. San Diego: Academic Press.

AUTHOR BIOGRAPHIES

NIELS A. LANG is a Phd. student at Erasmus University Rotterdam. He researches simulation in logistic system design, with an emphasis on economic analysis. His e-mail address is `<nlang@fbk.eur.nl>`.

PETER H.M. JACOBS is a PhD. student at Delft University of Technology. His research focuses on the design of decision support services for the design and management of a business alliance portfolio. His e-mail address is `<p.h.m.jacobs@tbm.tudelft.nl>`.

ALEXANDER VERBRAECK is an associate professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of open and generic libraries of object oriented simulation building blocks in Java. Contact information: `<a.verbraeck@tbm.tudelft.nl>`.