Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

JAVABEAN-BASED SIMULATION WITH A DECISION MAKING BEAN

Miki Fukunari
Yu-liang Chi
Philip M. Wolfe

Department of Industrial and Management Systems Engineering
Arizona State University
PO Box 875906, Tempe, AZ, 85287-5906, U.S.A.

ABSTRACT

In this paper a methodology is introduced to ease web-based simulation development: component-based development with a decision making bean. Component-based development, which permits use of drag and drop development, can reduce development time significantly since it requires no coding. Also, a decision making bean simplifies definition of logical relationships among components. This paper introduces the JavaBeans Discrete Simulation (JBDS) system which contains a library of basic components for simulation models with a decision making bean. With JBDS, a user can easily perform 'what-if' analysis by changing component's attributes or relationships among components, since a user can visually change bean attributes in a bean builder tool.

1 INTRODUCTION

Java was introduced in late 1995. It gathered a lot of attention due to powerful features such as platform-independence and Internet-capability. Since Java is an object-oriented programming language, a Java class can be reused. Software reusability speeds development time. This reusability is further enhanced by Java component technology, JavaBeans, which was released with Java 1.1 in early 1997. Components are self-contained elements of software that can be controlled dynamically and assembled to form applications. Thus, a user does not need to write code. Component-based simulation has many advantages, such as reusability of components and simplicity of development due to visual programming. Some object-oriented or component-based simulation languages have been developed using C++ or Java. Healy and Kilgore (1997) developed JavaBeans-based Silk. Several benefits can be derived from using JavaBeans such as platform-neutral, Internet-capable, and visual programming. JavaBeans Discrete Simulation (JBDS) uses JavaBeans as simulation nodes as well as Java classes as common simulation modules. Nodes communicate with each other as a simulation progresses, and they're combined by an event mechanism. In the JBDS, graphics assists in defining relationships among nodes.

Lance and Wolfe (Sherry and Wolfe 1996) introduced the Operational Procedure Table (OPT) which combines State and Process models into a single model in tabular format. An OPT has two sections; scenario and behavior. A scenario is a combination of conditions to be satisfied for an event to occur and a behavior is a combination of actions to be invoked corresponding to the scenario. A decision making bean uses an OPT representation to define complex relationships among nodes. Since the table interpretation is easy to design and read, decision making nodes are useful for constructing a set of decision making logic and for making presentations to users. Also, this table representation simplifies checking for all possible combinations of conditions (completeness) and for only one action for one combination of conditions (consistency). It means that logic errors are easier to identify. Thus, decision making nodes can improve design efficiency and communication with users and other designers.

JBDS has a set of common libraries of simulation components. This visual programming system can reduce the complexity of simulation model development using visual representation and development. The reminder of the paper describes JavaBeans, the JBDS structure and its implementation.

2 JAVABEANS

The JavaBeans specification defines a JavaBean as a reusable software component that can be manipulated visually in a builder tool. JavaBeans are appropriate for components that can be visually manipulated while Java classes are appropriate for providing functionality to programmers. Thus, in the JBDS, nodes are JavaBeans; consequently, they can be manipulated in a builder tool to develop simulation models.

The three most important elements of a JavaBean are the set of properties it exposes, the set of methods it allows other components to call, and the set of events it fires. Properties are attributes of a bean. A user can set property values using a visual development tool. The event is comprised of three parts; an event source, an event listener, and an event object. The event model facilitates communication between beans through the mechanism of firing and receiving an event. When an event source fires an event, event listeners for the event receive the event and invokes the method associated with the event. This event model separates an event source from an event listener. An event source and an event listener can be connected at run-time. This dynamic binding supports independent development of components and applications.

A builder tool provides a mechanism to introspect beans in order to expose their properties, methods, and events. This introspection mechanism allows a user to manipulate a bean's appearance and behavior at run-time in a builder tool. A user just needs to understand each bean's capability and how to operate a builder tool. Then a user can create an application with little knowledge of Java by changing property values, and connecting beans at run-time through pre-defined events.

3 JAVABEANS DISCRETE SIMULATION (JBDS)

The JBDS system is structured into three basic elements; entity, event, and nodes.

3.1 Entity

In the JBDS, entities are Java classes. Entities move through nodes passing information. Each entity encapsulates data such as identifier, arrival/leaving times and attributes. Entities usually are created, moved through nodes following an event mechanism, and then destroyed when they leave.

3.2 Event

Events are Java classes utilized by nodes. Events in JBDS take care of communication between nodes. When an event occurs, a system state changes. Entities move through states as an event occurs. JBDS has four events that deal with communication and scheduling. For example, in basic communication, the Simulation Event is invoked which passes entity information to a next node when necessary conditions are satisfied. Events communication can be defined by connecting nodes in a builder tool since nodes communicate with each other by firing and receiving events.

3.3 Nodes

Nodes may be categorized three ways: basic nodes, controller node, and decision making nodes.

3.3.1 Basic Nodes

At present, four types of basic nodes have been developed:

1. Create Node; Entities are created with this node. A create node can specify the distribution of the time between arrivals, the maximum number of entity creations and the first time of creation.

2. Terminate Node; Entities are terminated with this node. This node gathers all sets of information associated with entities that have reached the terminate node. A statistics output associated with a terminate node can be invoked at run-time.

3. Activity Node; This node defines the distribution of the service time and number of servers.

4. Queue Node; This node manages entities waiting for a service resource. Entities wait in this node in FIFO order. Statistics associated with a queue node can be invoked at run-time.

3.3.2 Controller Node

This node manages the simulation schedule by organizing future events. Nodes which create future events, such as create and activity nodes, communicate with a controller node passing future-event information. The controller node invokes scheduled events according to current simulation time. Also, it manages global variables such as number in queue and simulation time. This node controls global variables by communicating with other nodes.

3.3.3 Decision Making Node

This node manages decision making logic. Logic can be represented in tabular format when a simulation model involves a conditional or probabilistic decision. The JBDS has several variable keywords, such as attribute, current time, number in queue, etc. Using these keywords, a user can define logical relationships among nodes using a decision making node.

The logical relationships among nodes can be represented as an OPT in two sections: scenario and behavior. A decision making node uses this table representation. Figure 1 depicts that a decision making node after defining the logical relationships. In the left column, a user can define variables to be checked, which are keywords in the JBDS. For example, QUE(1)

represents the number in queue node 1, DEST is a place to which an entity is routed, and ATR(1) is the first attribute of an entity. In the second column, values for the variable can be defined. For example, the first row in the Figure 1 defines the condition of QUE(1) ≤ 5. In the other columns (right from the leftmost two columns), the user can set possible combinations of conditions to be checked for the scenario section and possible combinations of actions to be invoked for the behavior section. For example, an entity is assigned value of 6 time units as the first attribute and then is routed to queue node 1 if QUE(1) ≤ 5. The OPT representation simplifies the design of logical relationships among nodes since this tabular format is easy to read and design. As a result, complex simulation models can developed in less time. Also, this table representation simplifies completeness and consistency checking. Decision making nodes can be a significant aid in modeling decision making logic.

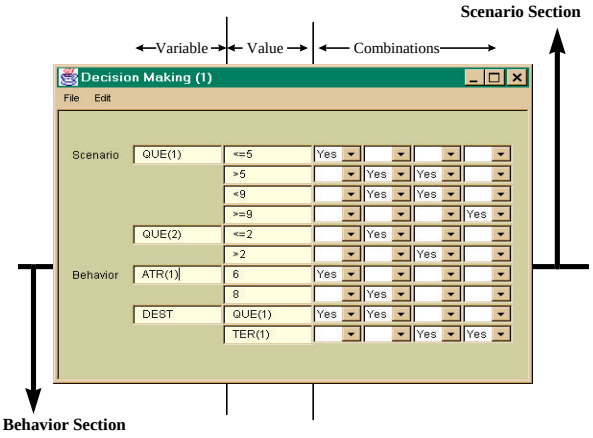

Figure 1 Decision Making Node

3.4 Trace Capability

The trace feature of JBDS lets a user examine the movement of entities through the system. After a simulation run, data associated with an entity, such as arrival time, leaving time, etc. can be shown by invoking trace output in a terminate node at run-time. Using this data, the JBDS can trace a detailed history of all entity movements.

4 JBDS IMPLEMENTATION

The JBDS can be implemented using any JavaBeans visual builder tool. This research was performed using the Beans Development Kit (BDK) by Sun Microsystems. An inspector/adjuster network model (Pritsker 1995, p.131) was simulated (see Figure 2) to illustrate the system. In this example, 70 percent of the items inspected are routed directly to packing and 30 percent of the items require adjustment. Following adjustment, the items are returned for reinspection. The inspection time is a function of the number of items waiting for inspection and the number waiting for adjustment (see Figure 1 where ATR(1) is the inspection time).

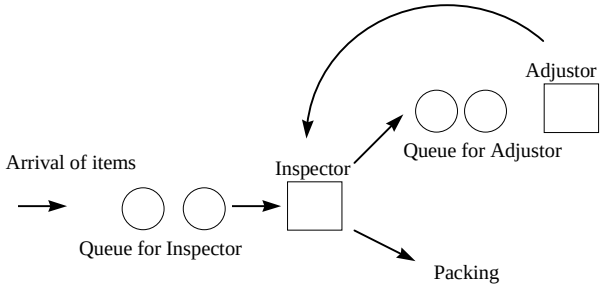

Figure 2 An Inspector/Adjuster Network Model

In a builder tool, a user can see a list of all nodes, drag and drop them into the design window, connect them so that they communicate with each other, and define their properties. This system has two decision making nodes. One decision making node occurs before the inspector queue (see Figure 1). The other decision making node occurs after the inspector activity (see Figure 3). The decision making node in the Figure 3 specifies that an entity is routed to queue node 2 with probability 0.3 and routed to the terminate node 2 with probability 0.7. After defining all relationships among nodes, the simulation model developed is shown in Figure 4. Since all development is made in a visual programming manner, it is simple and easy to design.

5 CONCLUSION

The JBDS system has demonstrated the feasibility of component-based simulation development using a table representation to define decision making logic. A user can rapidly develop models with user-friendly graphical representation instead of writing programs. The JBDS developed in this study may be accessed at (URL: http://pwolfe.eas.asu.edu/sim.html). Since Java has robust cross-platform capabilities, JBDS can be used in a distributed environment to develop complex simulation models.

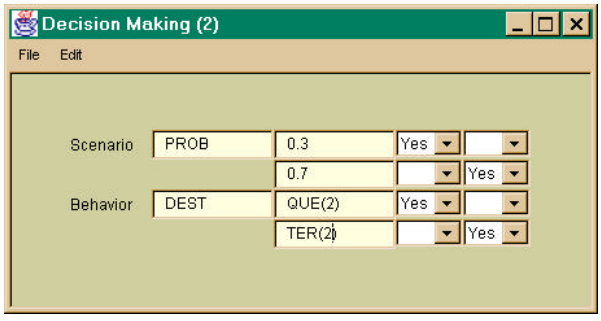

Figure 3 Decision Making Node

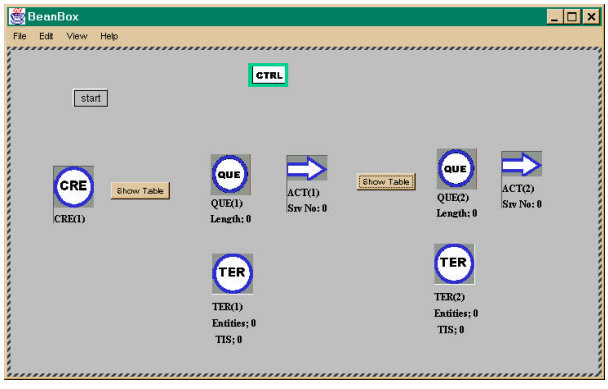

Figure 4 Simulation Application

As future work, the decision making bean will be enhanced to include logical completeness and consistency checking. In addition, more generalized and flexible components will be developed. The potential advantages, such as cross-platform and easy-to-design capability of this approach are very significant.

REFERENCES

Englander, R. 1997. *Developing Java Beans*. O'Reilly & , Inc.

Flanagan, D. 1997. *Java in a Nutshell*. 2nd ed. O'Reilly & Associates, Inc.

Healy, K. J., and R. A. Kilgore. 1997. Silk™: a Java-based process simulation language. *Proceedings of the 1997 Winter Simulation Conference*, 475-482.

Howell, F., and R. McNab. 1998. A discrete event simulation package for Java. *1998 International Conference on Web-Based Modeling & Simulation.*

Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 1998. *Simulation with Arena*. The McGraw-Hill Companies, Inc.

Khoshnevis, B. 1994. *Discrete systems simulation*. McGraw-Hill, Inc.

Kilgore, R., and K. Healy. 1998. Java, enterprise simulation and the Silk simulation language. *1998 International Conference on Web-Based Modeling & Simulation.*

Oaks, S., and H. Wong. 1997. *Java Threads*. O'Reilly & Associates, Inc.

Pritsker, A. A. B. 1995. *Introduction to simulation and SLAM II*. 4th ed. New York: John Wiley & Sons, Inc.

Pritsker, A. A. B., J. J. O'Reilly, and D. K. LaVal. 1997. *Simulation with Visual SLAM and AweSim*. New York: John Wiley & Sons, Inc.

Shlaer, S., and S. J. Mellor. 1992. *Object lifecycles: modeling the world in states*. New Jersey: Prentice-Hall, Inc.

Sherry, L., and P. M. Wolfe. 1996. The Operational Procedure Table: a formalized approach to business process specification. *IIEC Proceedings.*

Vanhelsuwe, L. 1997. *Mastering JavaBeans*. SYBEX, Inc.

AUTHOR BIOGRAPHIES

MIKI FUKUNARI received a BS in Physics from Tokyo Metropolitan University. Currently, she is a Master student in Industrial Engineering at Arizona State University. Her research field includes Component-Based Software Development, Simulation Modeling Methodology, and Distributed System.

YU-LIANG CHI received a BS and Master of Management Information System at Taiwan. He also has Master of Computer Science from Arizona State University. Currently, he is a Ph.D. student in Industrial Engineering at ASU. His research field includes Distributed System, Networking and Object-Oriented Technology.

PHILIP M WOLFE received a BS in Industrial Engineering and a BS in business administration from the University of Missouri. He also has Masters and Ph.D. degrees in Industrial Engineering from Arizona State University. Currently, He is a professor in the Dept. of I&MSE at Arizona State University. The majority of his work, teaching experiences, and research interests are related to Information Technology and Decision Support Systems.