Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

ADVANCED TUTORIAL: OVERVIEW OF SIMULATION WORLD VIEWS

C. Dennis Pegden

Simio LLC
504 Beaver Street.
Sewickley PA, 15143

ABSTRACT

Simulation models are built using one or more "world views" that provide the underlying framework for defining the system of interest. This tutorial presents an overview and brief history of the alternative modeling world views for discrete event simulation. In specific this tutorial discusses the event, process, and object worldviews, and highlights the differences and relationships between these approaches. It provides practitioners with an understanding of the advantages and disadvantages of each of these modeling views, as well as guidance on leveraging and combining the strengths of these different modeling approaches.

1 INTRODUCTION

A simulation model executes on a computer to dynamically act out the behavior of the real system over time. More formally, simulation is a set of state variables and a mechanism for changing those variables over time. A simulation modeling worldview provides the practitioner with a framework for defining a system in sufficient detail that it can be executed to simulate the behavior of the system. Unlike simple static descriptive tools such as Visio, IDEF, UML, etc., a simulation modeling worldview must precisely define the dynamic state transitions that occur over time. The worldview must provide a definitive set of rules for advancing time and changing the state of the model.

Over the 50 year history of simulation there has been three distinct world views in use: event, process, and objects. These were all developed in the 60's by the pioneers of simulation (Markowitz, Hausner, and Karr 1962; Gordon 1961; Dahl and Nygaard 1967). Although these world views have been significantly refined over the past 50 year period, the basic ideas for these three primary world views have not changed.

2 EVENT MODELING

In this modeling paradigm the system is viewed as a series of instantaneous events that change the state of the system over time. The modeler defines the events in the system and models the state changes that take place when those events occur. An important concept is that simulated time cannot advance within an event. Modeling an activity that transpires over time requires separate events to define the start and end of the activity. This approach to modeling is very flexible and efficient, but is also a relatively abstract representation of the system. As a result many people find modeling using an event orientation to be difficult.

We will illustrate the event worldview using a model of a service facility with a fixed capacity that defines the number of service operations that can be processed in parallel. When modeling a system using the event worldview we begin by asking the following questions:

1. How is the state of the system defined?
2. What events occur that can change the state of the system?
3. What is the logic within each event that defines the state transitions?

In our system example the state of the system can be defined by the number of busy servers, and the number of customers waiting to start service. This simple state description assumes that we do not distinguish between individual servers or customers (if we did, a more elaborate state description and transition logic would be required). The events that can change this state are a new customer arrival and a customer departure at the end of servicing.

The state transition logic for each of these events is summarized below in Figure 1. Note that each arrival schedules the next arrival in the sequence. We then check to see if we have available capacity for this arrival and if so, we bump the number busy and schedule the depart event. Otherwise we bump the number waiting. In the depart event we check if we have customers waiting and if so we decrement the number waiting and keep the number busy unchanged and schedule the departure event for the next customer. Otherwise we reduce the number of busy servers.

```
Arrive
{
  Schedule next Arrive event to occur at time now + inter-arrival time
  If (NumberBusy < Capacity)
  {
      NumberBusy = NumberBusy + 1
      Schedule the Depart event to occur at time now + service time
  }
  Else NumberWaiting = NumberWaiting + 1
}

Depart
{
  If (NumberWaiting > 0)
  {
      NumberWaiting = NumberWaiting – 1
      Schedule the Depart event to occur at time now + service time
  }
  Else NumberBusy = NumberBusy – 1
}
```

Figure 1: State transition logic for simple service facility.

We typically define the logic for the state transitions for each event using a simulation programming language (e.g. Simscript), or a general purpose language (e.g. Java or C++) augmented with a library (e.g. GASP) to assist with common simulation functions such as sampling from distributions or scheduling events.

When this model runs it executes the sequence of events in scheduled order. Time "jumps" ahead from event to event until the end of the simulation is reached (based on simulated time, number customers processed, etc).

The states in this simple example are all discrete states that can only change at event times. It is also possible to have continuous state variables that change on their own between events (e.g. the temperature of an ingot heating in a furnace).

In this simple example all events are scheduled to occur at specific times in the future and are typically referred to as time events. There are also other types of events that can occur that are triggered by logical conditions in the model. A change event is one that occurs as the result of a discrete change to a state. A cross event is one that occurs as the result of a continuous changing state variable reaching or crossing a threshold value.

The event world view was implemented by tools such as Simscript and GASP and these were widely used during the first 20 years of simulation. These tools were favored by many because they were very flexible and could be used to efficiently model a wide range of complex systems. However event-based models were often difficult to understand and debug and required programming skills that limited there general use. Once the process modeling tools became as flexible and as computationally efficient as the event tools users quickly migrated to the process worldview because it was easier to learn and use.

Although the event worldview is no longer used as the primary modeling approach in real applications, it remains important to understand. Many process and object based simulation tools maintain an event capability as a "backdoor" for flexibility. In addition all discrete event simulation systems implement their internal logic using this basic modeling approach, regardless of the worldview that they present to the user.

3 PROCESS MODELING

In the 80's the process orientation displaced the event orientation as the dominant approach to discrete event simulation. In the process view we describe the movement of passive entities through the system as a process flow. The process flow is described by a series of process steps (e.g. seize, delay, release) that model the state changes that take place in the system. Unlike a simple event, a process step may execute as a sequence of events over time. This approach dates back to the 1960's with the introduction of GPSS and provided a more natural way to describe the system. However because of many practical issues with the original GPSS (e.g. an integer clock and slow execution) it did not become the dominant approach until improved versions of GPSS (e.g. GPSS/H) along with newer process languages such as SLAM and SIMAN became widely used in the 80's.

When modeling a system using the process worldview we begin by asking the following questions:

1. What are the entities that move through the system?
2. What processes are executed as the entity moves from step to step in the process?

Process models are typically defined in form of a flowchart, where entities enter the process at the first step, and exit the process at the last step. The process flow chart for our simple service example is shown in Figure 2, where customers enter the process, wait to seize an available server, delay by the processing time, release the server, and then tally their time in system.

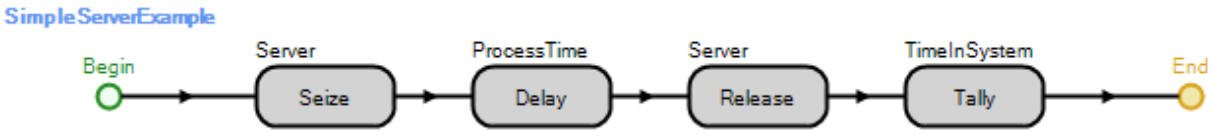

Figure 2: Process model for the simple service facility.

The service operation is modeled by a resource that has a capacity that may be seized and released by entities as they flow through the process. There are two basic types of time advances that may occur in a processing step – one is a planned delay (e.g. wait for 2 minutes). This is illustrated by the Delay step that delays the entity for the specified processing time. The other is a conditional delay (e.g. wait until Fred becomes available or a tank becomes full). This is illustrated by the Seize step that delays (if necessary) to seize a unit of the resource representing the Server.

During the 80's graphical modeling and animation also emerged as key features in simulation modeling tools. Graphical model building simplified the process of building process models, and graphical animation dramatically improved the viewing and validation of simulation results.

Another conceptual advance that occurred with process modeling during this time was the introduction of hierarchical process modeling tools that supported the notion of domain specific process libraries. The basic concept here is to allow users to create new process steps by combining existing process steps. The widely used Arena modeling system is a good example of this capability.

4 OBJECT MODELING

Although a process orientation has proven to be very effective in practice, an object orientation provides an alternative modeling paradigm that is more natural and in most cases easier to use. In an object orientation we model the system by describing the objects that make up the system. For example we model a factory by describing the workers, machines, conveyors, robots, and other objects that make up the system. The system behavior emerges from the interaction of these objects.

Objects as a formal concept were introduced by Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center in Oslo in the 1960s in Simula 67. Simula introduced the notion of classes of behavior (e.g. a Worker or Machine), and instances thereof (objects, e.g. Fred and Drill), as part of an explicit modeling paradigm. Although this was introduced as a simulation modeling concept, it completely changed the design and implementation of programming tools in general. . The ideas of Simula 67 influenced many later programming languages, including Smalltalk, LISP, C++, Java, and C#. It was not only the most significant development in simulation software, but perhaps the greatest advance in computer software in general.

Although the process worldview is still widely used in practice, a growing number of successful products have been introduced based on the object orientation. Although the object modeling paradigm is simpler and more powerful than the process orientation, some of the early products were difficult to learn and use or slow in execution. As object tools improve they will increasingly replace process tools as the primary world view for modeling.

There are some differences of opinion as to what it means to be an object-based or object-oriented simulation modeling tool. Since the object approach is generally viewed by the market as good, nearly all simulation software products claim to be object-oriented, which leads to lots of confusion about what this term really means. At a minimum an object-based simulation tool uses modeling constructs that directly relate to the physical system that they represent, as opposed to a logical process. For example an object based system would model using constructs such as machines, forklift trucks, doctors, etc., as opposed to a process description composed of time delays, seizing/releasing resources, state assignments, etc.

Some "object based simulation tools" are closed in that users cannot add new object classes (although they may be able to extend objects with event logic). Open systems allow users to add new object classes using an object oriented approach. To the purest an object-oriented simulation modeling tool must both be open and also support some key constructs related to creation of new objects. The concepts that are typically assumed to be important include the following:

- **Classes**
  The abstract characteristics of a thing (object), including its characteristics (its attributes) and behaviors (the things it can do)
- **Instances**
  The actual object created at run-time. For example Fred and Sue are instances of the Worker class. The set of values of the attributes of a particular object is called its state. The object consists of the state and the state transition behavior that's defined in the object's class.
- **Interface**
  The way objects interact in terms of passing values and messaging.

- **Sub-classing**
  Subclasses are more specialized versions of a class, which inherit attributes and behaviors from their parent class, which they can then modify and introduce their own new states and behaviors. For example a Worker class might be sub-classed into a Doctor and Nurse, where each of these two new classes have their own special states and behaviors, but also inherit some basic behavior from the Worker class.

Examples of modern object-oriented simulation tools that meet this minimal set of object-oriented features include AnyLogic, FlexSim, and Simio.

5 AGENT BASED MODELING

The term agent based modeling has become widely used in recent years. The basic concept of agent based modeling is that a system is modeled by placing agents in the system and letting the system evolve from the interaction of those agents. Example applications include crowds moving through an area, customers responding to new product introductions, or troops in combat.

Agent based modeling is typically implemented using an object-oriented simulation tool. Hence this is not a new discrete event world view, but rather a group of applications that are modeled with the object world view. Classes are used to define agent states and behaviors and instances (often large numbers) are placed in the model. The agents (objects) interact and the system state evolves over time. Some of the application areas in agent based modeling present some unique challenges, particularly when large numbers of agents are involved (e.g. modeling the evacuation of fans from a stadium).

6 MULTI-PARADIGM MODELING

Many of the simulation modeling tools support multiple modeling paradigms. By providing alternative paradigms in the same tool the user can select the modeling approach that best fits the problem at hand.

SLAM (Simulation Language for Alternative Modeling) was one of the first widely used tools to promote the idea of mixing alternative modeling approaches (processes and events and continuous) in the same model. In the case of SLAM the process/continuous modeling was used for basic modeling, and the events were used as a "back door" to provide added flexibility.

The modern object-oriented simulation tools also employ a multi-paradigm approach. These tools combine the ease and rapid modeling provided by objects with the flexibility added by incorporating user specified events and/or processes. For example an object representing a server might have selected points in the object logic where the user can insert their own event logic or process logic.

Event logic is typically incorporated into objects using either a programming language such as Java or C++, or a special scripting language that can be used in place of code. However in either case event logic has one major restriction: simulated time cannot advance within the event. This severely restricts the type of logic that can be inserted into an object instance. For example it is not possible to wait for a worker to become available within an event since this would require time to advance.

A much more powerful approach is to combine objects with processes. Since processes can span time they provide the user with considerable more power to extend the behavior of their objects. Hence processes can be embedded within an object to wait for a specified time or specified condition (e.g. Fred is available). This approach combines the full power and flexibility of processes with the ease and rapid modeling capability of objects.

REFERENCES

Dahl, O.-J. and K. Nygaard. 1967. Simula 67. *IFIP TC 2 Working Conference on Simulation Languages*, Oslo.

Gordon, G. 1961. A general purpose systems simulation program. *In Proceedings of the Eastern Joint Computer Conference*, Washington, D.C.

Markowitz, H., B. Hausner, and H. Karr. 1962. *SIMSCRIPT: A simulation programming language*. Englewood Cliffs, N. J.: Prentice Hall.

AUTHOR BIOGRAPHY

C. DENNIS PEGDEN is the CEO/Founder of Simio LLC. Prior to this position he was the founder and CEO of Systems Modeling Corporation, now part of Rockwell Software, Inc. He has held faculty positions at the University of Alabama in Huntsville and The Pennsylvania State University. He led in the development of the SLAM, SIMAN, Arena, and Simio simulation systems. He is the author/co-author of three textbooks in simulation and has published papers in a number of fields including mathematical programming, queuing, computer arithmetic, scheduling, and simulation. His email address is <cdpegden@simio.com>.