A METHODOLOGY FOR DEVELOPING DES MODELS: EVENT GRAPHS AND SHARPSIM

Arda Ceylan [(a)], Murat M.Gunal [(b)]

[(a)] Institude of Naval Science and Engineering
Turkish Naval Academy, Tuzla, Istanbul, Turkey
[(b)] Department of Industrial Engineering
Turkish Naval Academy, Tuzla, Istanbul, Turkey

[(a)] aceylan@dho.edu.tr [(b)] mgunal@dho.edu.tr

ABSTRACT
In this paper, a methodology for fast development of Discrete Event Simulation (DES) models is presented. The methodology simply works in two stages. In the first stage the modeler builds a Conceptual Model (CM) of the system to be modeled. A CM is represented as an Event Graph (EG). EGs are used to document the events and their causes in a system. In the second stage the CM is translated to an Event Based DES model. To fulfill this task we developed a DES library, SharpSim, using C# (CSharp) programming language. This paper gives an introduction to our methodology. We provide an insight into SharpSim and EGs, and illustrate a modeling example.

Keywords: Discrete Event Simulation Library, Event Scheduling, Event Graph, Simulation Software.

1. INTRODUCTION

Due to its popularity in simulation world, plenty of Discrete Event Simulation (DES) software has been developed and this trend is likely to continue in the future. There are two main types of simulation software; those which aim at non-programmers (e.g. a graphical user interface which provides drag and drop facilities to build a model by simple mouse clicks) and others which require programming skills (e.g. extending a given source code library to write a full program). First type of software is Commercials-Off-The-Shelf (COTS) such as Arena, Simul8, and Flexsim and they reach a wider user community than the other type does. It is in fact for this reason why the first type dominates the market. The obvious difference of the two types is the user friendliness; one requires the knowledge of how the software is used, and the other requires special expertise, e.g. programming. It is noteworthy that it is dangerous to strictly separate the two types since most COTS software today provides limited programming features.

In either type of simulation software, a DES is approached by a variety of worldviews. A worldview is described as a "modeling framework that a modeler uses to represent a system and its behavior" (Carson, 1993). Simulation software adopts one of these world-views. DES worldviews in the literature can be categorized into:

- Process Interaction
- Activity Scanning
- Three Phase
- Event Scheduling

Process Interaction focuses on processes which can be described as "set of events" (Roader, 2004). In this approach, entity flows play the main role where flows include all states of objects. The process is described as "a time-ordered sequence of events, activities and delays that describe the flow of a dynamic entity through a system" (Carson, 1993). Process Interaction is popular and widely used since it is easier to conceive and implement, but "deadlock problem" (Pidd, 1998) stands as the weak point. This approach is used commonly by COTS simulation software, such as Automod. Another common approach, Flow Transaction, is a derivative of Process Interaction. Arena, ProModel and Witness are some of popular software using this approach (Abu-Taieh and Sheikh, 2008).

In Activity Scanning, all activities are scanned in each time step and initiated up to their conditions. It is also called as Two Phase. Three Phase approach is a variant of Activity Scanning. It is more tedious to model, but faster since only conditional activities are scanned at each step.

Event Scheduling requires the identification of events and their impacts on system's state variables. This approach is most efficient but can be complicated to conceptually represent when the model size is big.

In this paper, we particularly focus on Event Scheduling world-view. As a first step of our interest we review the methods for conceptual modeling, such as Event Graphs (EG). Secondly, we present a new DES library developed in C#: SharpSim. Additionally we give a general idea about some basics of DES and pertinent general purpose DES software in use, and then position the SharpSim in this picture. Finally, we

provide an EG of M/M/n queuing system and a short tutorial on how a SharpSim model can be built.

2. A BRIEF REVIEW OF CONCEPTUAL MODELING METHODS AND EVENT GRAPHS

Conceptual modeling in DES is an active research area and there is still no consensus among simulation modelers on its representation, although Onggo (2009) is an attempt in which unified conceptual modeling is discussed. There are a variety of methods to conceptualize the problems in hand in terms of logical flow of objects and events in the system. We review three methods here.

The first method is the most commonly used; Process Flow Diagrams (PFD). PFDs focus on the flows of entities in a system and are used by most COTS simulation software. A PFD is created by simply placing drag-and-drop objects to represent processes and links between these processes to represent interactions between processes. The modeler, in a way, treats him or herself as an entity and follows the processes which transform an entity.

The second method is Activity Cycle Diagrams (ACD) for conceptualizing the logical flows of objects in the system. In an ACD, life cycle of entities in the system is shown. In their life time, entities changes state and interact with each other. Entity states alternate from active to dead states. Simulation time moves forward and entities of the system spend time in these states. Active states represent activities which different types of entities can cooperate. Once an entity enters an active state, its duration can be determined, generally by taking a sample from a probability distribution. However some conditions must be satisfied for an entity to be in an active state, for example, if there is a server available and there is a client waiting in a queue, a customer entity enters to a service active state. Dead state is the opposite of an active state that is when an entity is idle or waiting for something to happen. This generally means a waiting area. Unlike an active state, duration of a dead state cannot be determined in advance since the time an entity spends in dead state is bound to preceding and succeeding activities.

Finally, Event Graphs (EG) are used to conceptualize a system by focusing on its events. EGs work well with Event Scheduling approach since "Event Graphs are a way of representing the Future Event List logic for a discrete-event model" (Buss, 2001). There are two main components of EG; nodes to represent events and edges to represent transitions between events. Figure 1 shows the basic structure of EG (Roader, 2004).

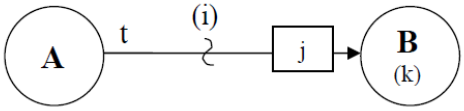

Figure 1: A Basic Event Graph

The translation of the EG in Figure 1 is as follows; "If condition (i) is true at the instant event A occurs, then event B will immediately be scheduled to occur t time units in the future with variables k assigned the values j".

3. SHARPSIM OVERVIEW

In the second stage of our methodology an Event Graph is translated into computer code to build a simulation model. SharpSim is developed for this purpose. SharpSim is an open-source Discrete Event Simulation (DES) code library developed in C# (The code can be accessed at http://sharpsim.codeplex.com). It implements Event Scheduling world-view which involves three main classes; Simulation, Event, Edge and 3 secondary abstract classes; Entity, Resource and Stats. The objects instantiated from these classes are used to implement the EG drawn, as described in the first stage of our methodology. SharpSim is appropriate for multi threading. This is particularly helpful for animation, for example a simulation model running as a thread can communicate with animation classes, e.g. updating screen objects periodically. In this section we briefly explain how SharpSim works.

3.1. Simulation Class

Simulation class is the core of SharpSim. It includes the main Event Scheduling algorithm and the thread that executes the model. Number of replications and seed number for random number generation are the parameters of this class.

There are four properties of simulation class and their descriptions are as follows;

- Future Event List (FEL): This collection involves the set of events that will be executed in the future. An instance of the event that is to be scheduled is inserted into FEL. FEL is sorted by the due time of the events, e.g. earliest event is on top of the list. Note that the event scheduling simulation algorithm scans FEL repeatedly until no events exist in FEL. After the execution of an event, it is removed from the list.
- Clock: The clock variable keeps the simulation time. It is handled in Run method and proceeds to the execution time of the next event.
- Events: This collection involves the set of events instantiated at the beginning of simulation and provides easy manipulation of events. Note that the events of a model are instantiated in the model class that is coded outside of SharpSim library.
- Edges: This collection involves the set of edges instantiated at the beginning of the simulation and provides easy manipulation of edges as in the Events list.

Simulation class includes two main and two supplementary methods. Main methods are described below. The two supplementary methods, Create Events and Create Edges, are useful for reading event and edge details direct from an Excel input file. Events and Edges are instantiated and added to Events and Edges collections.

• Run: The Event Scheduling algorithm is handled in this method. It involves a loop for each replication and another embedded loop for each event in the future event list. The first loop iterates for a number of replication times while the second embedded loop iterates till termination event is executed. In the second loop, first the clock is set to next event's execution time, then event is executed and at the end of the execution it is removed from future event list.

• Start Simulation Thread: This method is used to start simulation thread which is created when a simulation object is instantiated.

3.2. Event Class

Event is an activity which causes a state change. The set of events together with edges forming a system is created at the start of the simulation and according with interrelations among events and edges new events are cloned and added to future event list during simulation.

The constructor of this class has four arguments; event id, event name, priority, and event due time. If an instance of an event class is created with an event due time, the event is inserted to FEL directly. When more than one event has the same execution time, a second parameter is needed to decide which event will be executed first. Priority provides this secondary regulation. It is crucial to assign priorities on events particularly in complex systems. Properties of this class are explained below;

• Execution Time: Each event has an execution time. The execution time of an event is mostly set during the simulation.

• Parameter: This property is used to implement parameter passing on edges in EGs. When an event is executed, a parameter, either a single value such as an integer or an object such as a customer object, can be set into the next event. With this mechanism, for example, individual entities can be transferred from event to event.

• Queue: This property is used to keep the entities that are waiting to be scheduled into the FEL.

There is one method in the Event class, Event Executed, which is a delegate method associated with the next event. This is the point where C# event handling mechanism meets with the simulation's events. When the due time of an event comes this method is called to schedule the next event linked to the current event being executed. The event schedule occurs if the condition on the edge is true. It clones a new independent event from the following event, provides parameter passing between edge and cloned event, and sets its execution time and finally insert cloned event into the FEL.

3.3. Edge Class

Edge is a link between two events. It defines relations between events and accordingly flow of the system. Scheduling of events is decided up to edge conditions. Furthermore, execution times of newly cloned events are set according with edge's next event time value. The constructor of this class has three parameters; name of the edge, source event, and target event. Target events subscribe to source events. There are three properties of the edge class; next event time, attribute, and condition. Next event times can be deterministic or stochastic. Attribute is a variable which is set when parameters are passed between events. Condition is the condition of scheduling an event.

The modeler can create entities and resources by inheriting from the entity and resource classes. Additionally Stats class provides an easy output manipulation for the simulation.

4. M/M/N SERVICE SYSTEM SIMULATION

In this section, we aim to describe how a model of an M/M/n service system can be built using our methodology. As stated earlier, the first stage requires drawing an event graph of the system to be modeled. M/M/n queuing system's event graph is drawn in Figure 2. There are four events and six edges in this graph.

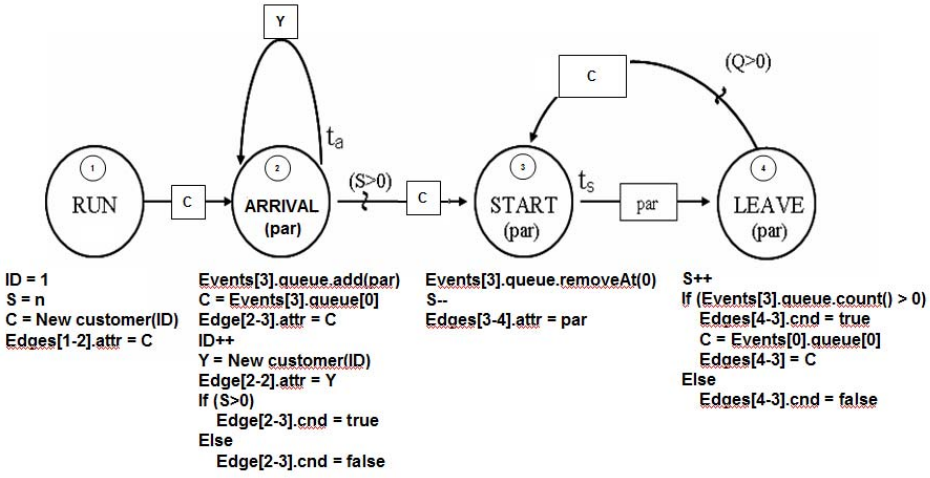

Figure 2: Event Graph of M/M/n

The Event Graph (EG) in Figure 2 has four events which represent start of simulation, arrival of customers, start of service, and end of service events in a queuing system. The variables are ID (arriving customers' ID number), S (number of available servers), and C (Customer Entity).

The explanation of this EG is as follows; When the Run event occurs, set the ID to 1 (first customer's ID), the S to n (there are n servers) and create C (an instance of Customer object). Setting the attribute of Edge [1-2] to C is required to pass the C object to the next event.

When the Arrival event occurs you first need to add the receiving Customer object to the next event's queue. Later you pull a Customer entity from the Start Event's queue and set this to the edge's attribute. Likewise, you need to create a new Customer instance and set it to the self loop attribute. Finally, you need to set the condition on edge between Arrival and Start events based on S (number of available servers).

When the Start event occurs, first customer in the Start event's queue is removed, since it is time for that customer to be served. Number of available servers decremented by one and the receiving customer entity is set as the parameter on the edge. This is to transfer the customer to the leave (end of service) event.

Final event is Leave event. Executing a leave event means that a customer finished the service and therefore number of available servers (S) must be incremented by one and a new Start event must be scheduled. Scheduling a Start event is possible if number of customers waiting in the queue (Q) is non-zero. Q is the queue count of Start event (Event[3]).

4.1. Building a SharpSim model

To build a simulation model in SharpSim, a C# project must be created. You need to create a Windows Forms Application project in a C# compiler such as Microsoft C# Express Edition 2008. The project must include the SharpSim library. SharpSim Library is a DLL file although full source code is provided and can be added to the project.

On the default form in your project, generally named as Form1, a variable of type Simulation, five variables of type Event, and five variable of type Edge are defined. You need to add a button and a richtextbox components on to the form to create a basic user interface. The model will run when the button is clicked and an output will appear in the text area.

On the button's click event, firstly you need to instantiate your simulation model by calling Simulation class's constructor. The constructor has three parameters; track list to show a default output screen, replication number, and seed number. Secondly, events are instantiated. An event has an ID, name, and priority parameters. Note that these simulation events have no due time given since all four events instantiated, and shown in Figure 2, are dynamic event. This means that their due time will be known during the simulation. On the other hand, the fifth event is the Termination event, which triggers when to stop the simulation, and has a due time. Termination event's due time is the replication length of the simulation.

After creating the events, you need to add State Change Listeners. State change listeners are related to C# event handling mechanism and help connect SharpSim events with C# form events, for example when the Run event occurs in SharpSim, Run method of the form is executed. An instance of Edge is instantiated for every edge in Figure 2. The Edge class has three parameters; name, source event, and target event. The modeller can set the time distribution and the distribution parameters on an edge by setting its ".dist" and ".mean" properties.

After the definitions, a stats collection line can be written, such as the delay time between arrival and leave events. The delay between these two events means the total time of customers in the system. And finally, the run method of the simulation instance is called which causes the simulation to start.

4.2. State Change Handlers

When a SharpSim event occurs, its corresponding method in the form is also executed. These methods are coded in the model file and inside these methods there are state change related codes, such as incrementing state variables and creating new entities.

Inside a state change handler, it is essential to write code inside "evt.EventExecuted += delegate(object obj1, EventInfoArgs e){ ...}" block. For example for the Run event in Figure 2, write a "public void Run" method and inside the method write the delegate line and then set the ID variable to 1 which means that the very first arriving customer's ID will be 1, set the S variable to 2 which means that we have initially 2 servers, create a new customer instance, and set the edge between Event 1 and 2 parameter value to this new customer.

For every event in the model, there must be a State Change handler method in the code.

4.3. Running the model

After buiding a SharpSim model as described above, the project is built and run. Clicking the button on the default form will start the simulation. This means that an instance of Simulation class, instances of events and edges will be created. Since the starting event is Run and scheduled to time 0, the model will start with this event. Execution of Run event will then cause scheduling an arrival event and in turn new arrivals will be created and so on.

In the text area, text outputs will appear as simulation runs. Note that any message, simulation related or not, can be written to the text area on the form inside the state change handlers.

5. CONCLUSION

In this paper we presented a methodology for building DES models. The methodology incorporates Event Graphs, as a conceptual modeling tool, and SharpSim, a new Discrete Event Simulation (DES) library. The DES library, SharpSim, is created using C# language and allows modelers to build DES models by programming in C#.

The SharpSim library is an implementation of event scheduling simulation approach. It aims at translating event graphs into simulation models easily. SharpSim is open source and can be downloaded at http://sharpsim.codeplex.com. The website also includes tutorials on modeling examples.

REFERENCES

Abu-Taieh, E.M.O., El Sheikh, A.A.R. (2008). Methodologies and Approaches in Discrete Event Simulation

Abu-Taieh, E.M.O., El Sheikh, A.A.R. Commercial Simulation Packages: A Comparative Study

Buss, A. (2001). Technical Notes, Basic Event Graph Modeling.

Carson, J.S. 1993. Modeling and Simulation Worldviews.

Pidd, M. (1998). Computer Simulation in Management Science. Chicester, UK: John Wiley & Sons.

Roeder, T.M.K. (2004). An Information Taxonomy for Discrete Event Simulations

Rossetti, M. D. (2008) "JSL: An Open-Source Object-Oriented Framework for Discrete-Event Simulation in Java", International Journal of Simulation and Process Modeling, vol. 4., no. 1, pp69-87, DOI: 10.1504/ IJSPM. 2008. 020614 Evans, W.A., 1994. Approaches to intelligent information retrieval. *Information Processing and Management*, 7 (2), 147–168.

Schruben, L. (1983). Simulation Modeling with Event Graphs. Communications of the ACM, Volume 23, Number 11.

Tocher, K. D. 1963. The art of simulation. London. English Universities Press.

Onggo, BSS. 2009. Towards a unified conceptual model representation: a case study in healthcare. Journal of Simulation, 2009, 3, 40-49.

AUTHORS BIOGRAPHY

Arda Ceylan has received his MSc Naval Operations Research degree at the Institute of Naval Science and Engineering in Turkish Naval Academy.

Murat Gunal is an assistant professor at the department of Industrial Engineering in Turkish Naval Academy.