SharpSim© Tutorials

Tutorial 1: M/M/n Service System Simulation
Tutorial 2: M/M/n Simulation using Excel input file
Tutorial 3: A Production/Inventory System Simulation

Ali Emre Varol, Arda Ceylan, Murat M. Günal

August - 2011

*SharpSim can be downloaded at http://sharpsim.codeplex.com/

Contents

List of Code Boxes

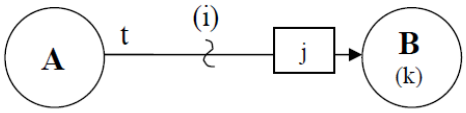
1 INTRODUCTION

SharpSim is an open-source Discrete Event Simulation (DES) code library developed in C#. You can build simulation models using SharpSim by coding in C#. You can download SharpSim at http://sharpsim.codeplex.com/. SharpSim implements event scheduling DES world-view and therefore it is essential to understand how event scheduling works. One way to learn event scheduling is to start with Event Graphs (EGs).

EGs are a way of conceptual representation of Event Scheduling world-view and there are two main components of an EG; nodes which represent events, and edges which represent transitions between events. A simple EG is given in Figure 1-1 which can be translated into the following sentence;

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

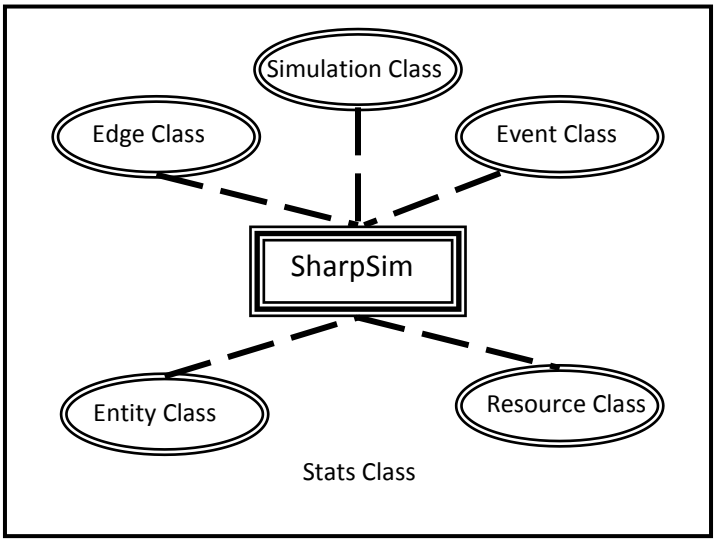
Figure 1-1 A Basic Event Graph



Note that Event A and Event B are the events which occur in the system that is to be modelled, for example arrival of a customer, starting a service etc.

SharpSim includes three main classes which are Simulation, Event, Edge and three secondary classes which are Entity, Resource and Stats, as shown in Figure 1-2. Please refer to SharpSim documentation for detailed explanation of these classes.

Figure 1-2 Class structure of SharpSim

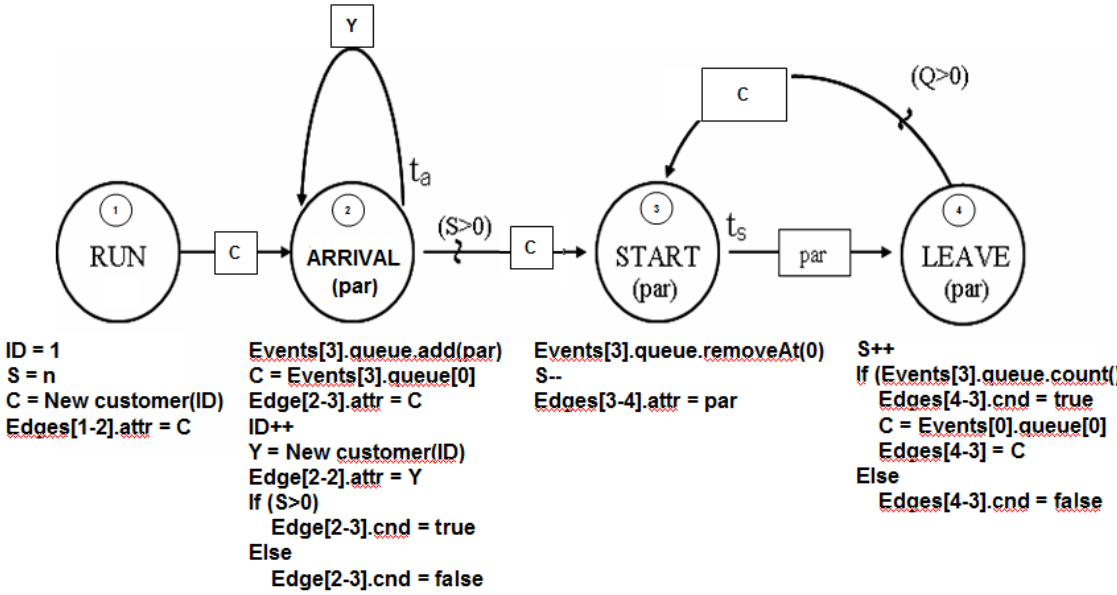


In this SharpSim tutorials set, three tutorials are given; Tutorial 1: M/M/n Service System Simulation, which is Example 2 in SharpSim pack, Tutorial 2: M/M/n Service System Simulation using Excel input file, which is Example 1 in SharpSim pack, and Tutorial 3: A Production/Inventory System Simulation, which is Example 3 in SharpSim pack.

2 TUTORIAL 1

In this tutorial a simple queuing system will be modelled using SharpSim. The Event Graph (EG) in Figure 2-1 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).

Figure 2-1 Event graph for M/M/n Queuing Model.



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.

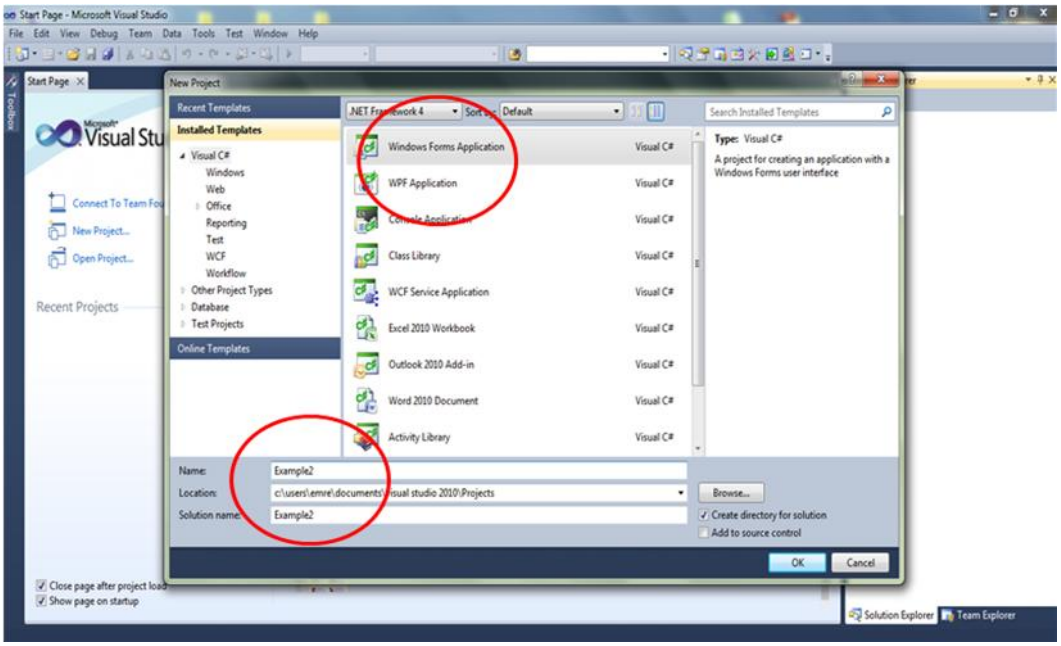
Final event is Leave event. Executing a leave event means that a customer finished the service and therefore number of available servers (S) must be incremented by one and a new Start event must be scheduled. Scheduling a Start event is possible if number of customers waiting in the queue (Q) is non-zero.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

To build this model in SharpSim, first you need to create a C# project and include SharpSim library.

2.1 Creating a C# Project

- Open a new Project (File/New Project) in Visual Studio 2010 (or any other C# compiler such as C# 2008 Express edition)
- Choose Windows Forms Application and name it as shown in Figure 2-2.

Figure 2-2 Creating a C# project

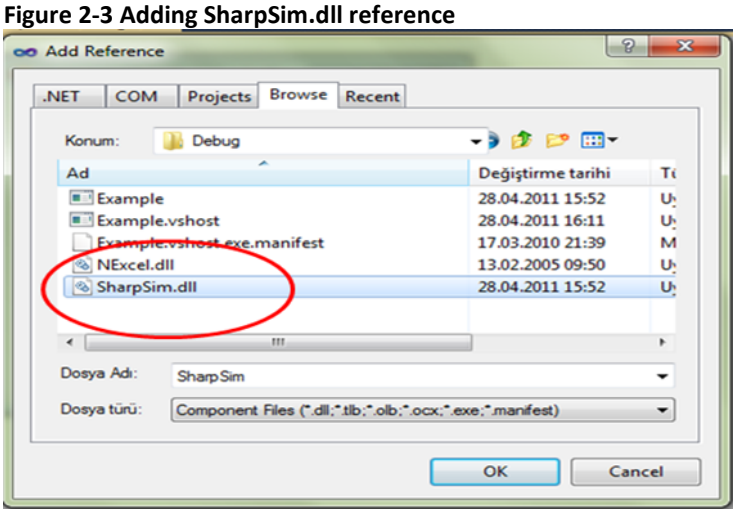


2.2 Adding SharpSim reference to your project

- After creating the project right click on "References" tree item in the project explorer and choose "Add reference …". Click on

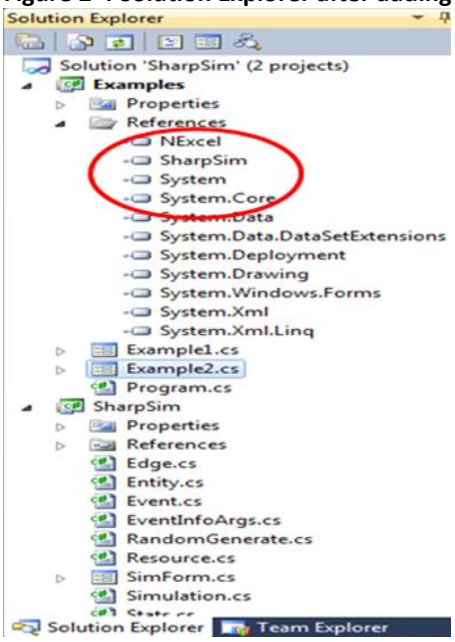
"Browse" tab and choose the "SharpSim.dll" file on your file system, as shown in Figure 2-3.

Figure 2-3 Adding SharpSim.dll reference



After adding the reference you must see "SharpSim" in your project explorer as shown in Figure 2-4. Note that if you want to use Excel inputs you also need to include the third party Excel library NExcel.

Figure 2-4 Solution Explorer after adding SharpSim reference



2.3 Setting the simulation form

- Open the form in you project and add a rich text box and a button.
- In the code editor add the using statements as shown in Code Box 2-1 in the code section of your form.

Code Box 2-1 "using" class in your example

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using NExcel;
using SharpSim;
```

2.4 Writing the model code

- Define the variables given in Code Box 2-2 in the global declarations section of your form. Note you must define every event and edge as seen in Figure 2-1.

Code Box 2-2 Defining your simulation model objects

```
public partial class Example2 : Form
{
    Simulation sim;

    Event eRun;
    Event eArrival;
    Event eStart;
    Event eEndService;
    Event eTerminate;

    Edge edge1_2;
    Edge edge2_2;
    Edge edge2_3;
    Edge edge3_4;
    Edge edge4_3;
}
```

Simulation provides a simulation platform where all the elements of the model can interact with each other. Simulation class includes the main properties of the simulation such as future event list and simulation time and the mechanism which handles event scheduling algorithm.

An Event represents a node in an Event Graph. An event is an occurrence which causes a state change in the system.

An Edge represents an arc in an Event Graph. An edge is a connection line between two events.

- On the button click event, instantiate the SharpSim simulation object as in Code Box 2-3. Simulation class has two overloaded constructors.

Code Box 2-3 Instantiating the simulation object

```
private void button1_Click(object sender, EventArgs e)
{
    sim = new Simulation(true, 10, false);
```

▲ 1 of 2 ▼ Simulation.Simulation(**bool trackList**, int replication, bool randomizeSeed)
Constructs a new Simulation with the given arguments
trackList: *If true, exhibits executed events on the Simulation form.*

▲ 2 of 2 ▼ Simulation.Simulation(**bool trackList**, int replication, bool randomizeSeed, int seedNo)
Constructs a new Simulation with the given arguments
trackList: *If true, exhibits executed events on the Simulation form.*

Tracklist : If true, executed events are shown on the Simulation form of SharpSim.
Replication : Number of times the model will be replicated with the given configuration.
Randomize Seed: If true, produces a new seed for each replication.
Seed No : Is used as a seed for Random type variable.

- Instantiate the Events as shown in Code Box 2-4. The Event class has two overloaded constructors. You need to create every event seen in Figure 2-1. Note that the Terminate event does not exist in the EG but since an event to stop the model is required you explicitly need to add this event to your model code.

Code Box 2-4 Creating the events

```
    eRun = new Event("1", "Run", 1, 0);
    eArrival = new Event("2", "Arrival", 4);
    eStart = new Event("3", "Start", 2);
    eEndService = new Event("4", "EndService", 3);
    eTerminate = new Event("5", "Terminate", 5, 50);
```

▲ 1 of 2 ▼ Event.Event(**string no**, string name, int priority)
Constructs a new Event with the given arguments
no: *Identification number of the event.*

▲ 2 of 2 ▼ Event.Event(**string no**, string name, int priority, double executionTime)
Constructs a new Event with the given arguments
no: *Identification number of the event.*

No : Identification string of the event.
Name : Name of the event.
Priority : Priority of the event.
Execution Time : Simulation time which the event will be executed

- 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 eRun event occurs in SharpSim, Run method of

this form (which will be explained later) will be executed.

```
//State change listeners
Run(eRun);
Arrival(eArrival);
Start(eStart);
EndService(eEndService);
Terminate(eTerminate);
```

- Instantiate the Edges. You need to create every edge, the arrows between events, in Figure 2-1. The Edge class has three parameters; name, source event, and target event.

```
edge1_2 = new Edge("1-2", eRun, eArrival);
edge2_2 = new Edge("2-2", eArrival, eArrival);
edge2_2.dist = "exponential";
edge2_2.mean = 5.0;
edge2_3 = new Edge("2-3", eArrival, eStart);
edge3_4 = new Edge("3-4", eStart, eEndService);
edge3_4.dist = "exponential";
edge3_4.mean = 5.0;
edge4_3 = new Edge("4-3", eEndService, eStart);
```

Edge.Edge(**string name**, Event sourceEvent, Event targetEvent)
Constructs a new Edge with the given arguments
name: *Name of the edge.*

Name : Name of the edge.
SourceEvent : Arc's departing event.
TargetEvent : Arc's arriving event

The modeller can set the time distribution and the distribution parameters on an edge by setting its ".dist" and ".mean" properties.

- Complete source code for button click event is shown in Code Box 2-5.
- The penultimate line "sim.CreateStats("2-4");" is written to collect statistics for the delay time between events 2 and 4. The delay between these two events means the total time of customers in the system.
- Final line "sim.Run()" calls the main simulation run method and the simulation starts.

Code Box 2-5 Complete source code for Button Click event

```
private void button1_Click(object sender, EventArgs e)
{
    sim = new Simulation(true, 10, false);

    eRun = new Event("1", "Run", 1, 0);
    eArrival = new Event("2", "Arrival", 4);
    eStart = new Event("3", "Start", 2);
    eEndService = new Event("4", "EndService", 3);
    eTerminate = new Event("5", "Terminate", 5, 50);

    //State change listener
    // First do state changes and later event schedules (creating edge)
    Run(eRun);
    Arrival(eArrival);
    Start(eStart);
    EndService(eEndService);
    Terminate(eTerminate);

    //Create edge (event schedule)
    edge1_2 = new Edge("1-2", eRun, eArrival);
    edge2_2 = new Edge("2-2", eArrival, eArrival);
    edge2_2.dist = "exponential";
    edge2_2.mean = 5.0;
    edge2_3 = new Edge("2-3", eArrival, eStart);
    edge3_4 = new Edge("3-4", eStart, eEndService);
    edge3_4.dist = "exponential";
    edge3_4.mean = 5.0;
    edge4_3 = new Edge("4-3", eEndService, eStart);

    //collecting statistics
    sim.CreateStats("2-4");

    //RUN THE SIMULATION
    sim.Run();
}
```

2.5 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.

2.5.1 Run Event state change

Translation of the pseudo code in Figure 2-1 is;

- 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 as shown in Code Box 2-6.

Code Box 2-6 Run Event state change handler

```
public void Run(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        //State changes will be written this section as shown below.
        ID = 1;
        S = 2;
        Customer customer = new Customer(ID);
        edge1_2.attribute = customer;
    };
}
```

2.5.2 Arrival Event state change

On an Arrival event (Code Box 2-7);

- Add the arriving Customer to the Start event's queue.
- Set the attribute of the edge between events 2 and 3 to the first customer waiting in the Start event's queue.
- Increment the global variable ID by one.
- Create a new customer
- Set the self loop creating arrival event's edge attribute to this new customer
- Set the edge between events 2 and 3 condition to True, if the number of servers available is greater than 0, and False otherwise.

Code Box 2-7 Arrival Event state change handler

```
public void Arrival(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
//To implement parameter passing, first you need to push the parameter to the queue
//and then pull the first customer in the queue (FIFO queue discipline)

        eStart.queue.Add(e.evnt.parameter);
        edge2_3.attribute = eStart.queue[0];

        ID++;
        Customer cust = new Customer(ID);
        edge2_2.attribute = cust;

        if (S > 0)
               edge2_3.condition = true;
        else
               edge2_3.condition = false;
    };
}
```

2.5.3 Start Event state change

On the Start event (Code Box 2-8);

- Remove the first customer in the Start Event's queue
- Decrement the number of servers available by one
- Set the attribute value of the edge between events 3 and 4.

Code Box 2-8 Start Event state change handler

```
public void Start(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        eStart.queue.RemoveAt(0);
        S--;
        edge3_4.attribute = e.evnt.parameter;
    };
}
```

2.5.4 EndService Event state change

On the EndService event (Code Box 2-9);

- Increment the number of available servers by one
- If the number of customers waiting in the Start Event's queue then set the condition on edge between events 4 and 3 to false. This means that there is no one waiting to be served.
- Otherwise, set the condition to True and set the first customer in the Start Event's queue to the attribute of the edge between events 4 and 3.
- Add a statistics collection line

Code Box 2-9 EndService Event state change handler

```
public void EndService(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        S++;
        if (eStart.queue.Count() == 0)
               edge4_3.condition = false;
        else
        {
               edge4_3.condition = true;
               edge4_3.attribute = eStart.queue[0];
        }

        Stats.CollectStats("2-4", e.evnt.parameter.ReturnInterval("2", "4"));
    };
}
```

2.5.5 Terminate Event state change

On the Terminate event () which is not in the event graph;

- Write a message to indicate that the replication is ended.
- Add the mean statistics to the global statistics dictionary.

Code Box 2-10 Terminate Event state change handler

```
public void Terminate(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        richTextBox1.Text += "Replication No : " + Simulation.replicationNow + "
               ended." + "\n";
        Stats.AddDataToStatsGlobalDictionary("2-4", Stats.Dictionary["2-4"].mean);
    };
}
```

2.6 Creating an Entity: Customer Class

SharpSim provides an abstract Entity class which can be extended to create model related entities. In our example the entities are customers and therefore a Customer class is created as shown in Code Box 2-11. It is a simple class with only an id property.

Code Box 2-11 Customer Class

```
class Customer : Entity
{
   public Customer(int id)
          : base (id)
   {
      this.identifier = id;
   }
}
```

2.7 Running the application

Select Program.cs on the right-side and double click it and be sure that the name of the form is written at Application.Run row as shown in Figure 2-5. Now, the model is ready to run. Build your application and run it. You will see two windows as seen in Figure 2-6.

Figure 2-5 Program code

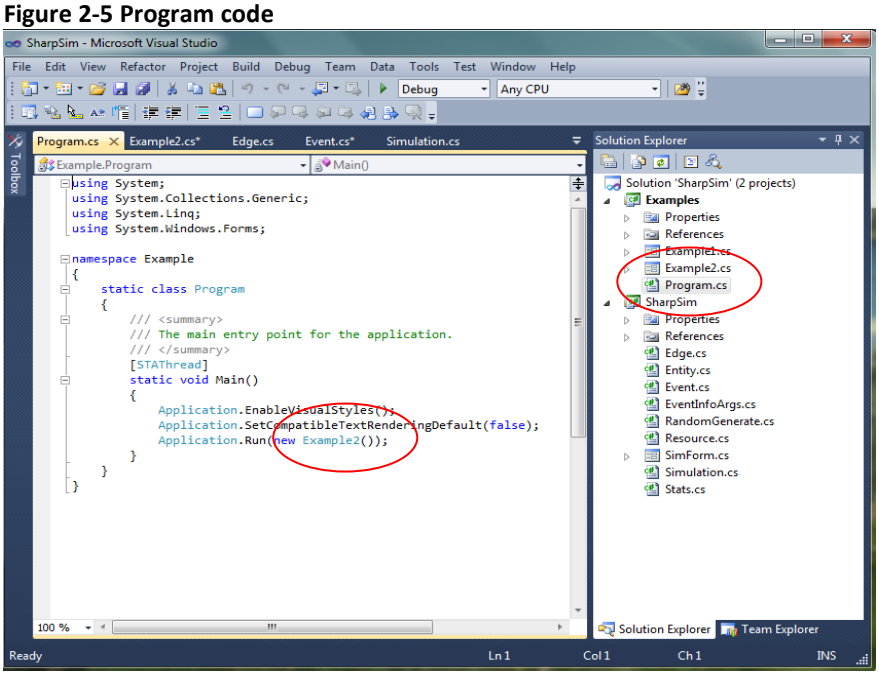
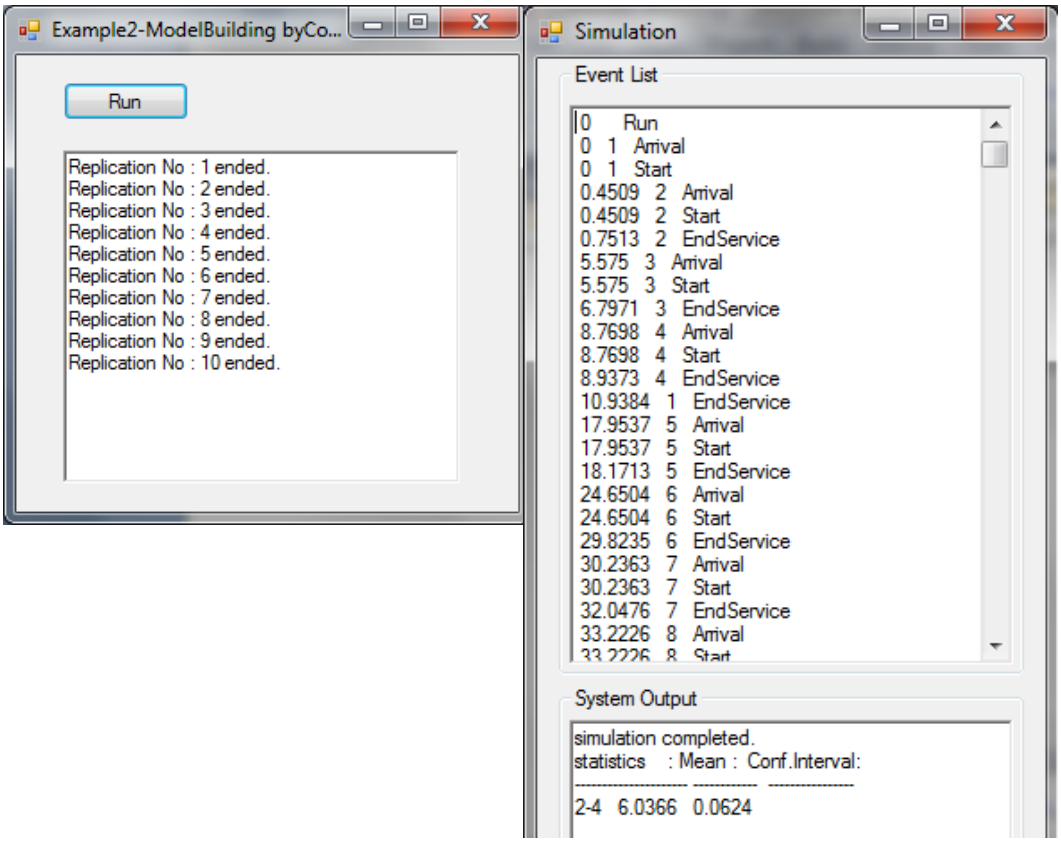


Figure 2-6 Output windows

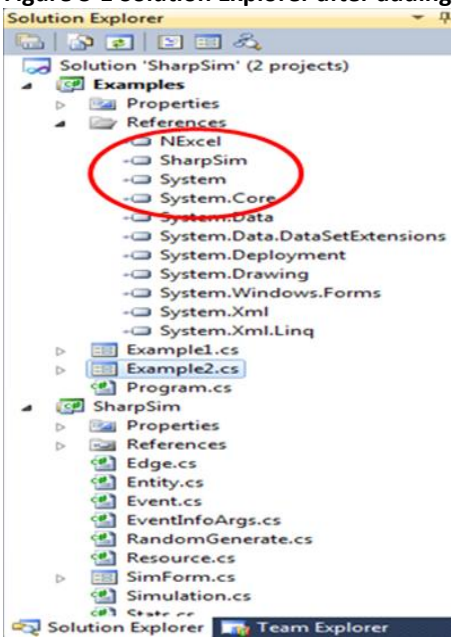


3 TUTORIAL 2

This tutorial is the second part of SharpSim tutorial series. As presented in Tutorial 1, you can build your models by instantiating your events and edges inside the code. This may be tedious when you have many events. SharpSim also provides methods to read event and edge information from an Excel file. This tutorial describes how a model of the EG given in Figure 2-1 is built using the Excel input file.

As stated before, this tutorial is only different than tutorial 1 in terms of the inputs being read from an Excel file. For this reason you must also include a third party Excel library NExcel in your C# project. The DLL for NExcel is supplied with the SharpSim.

Figure 3-1 Solution Explorer after adding SharpSim reference



3.1 Setting the simulation form

- Open the form in you project and add a rich text box, a button, and a OpenFileDialogbox.
- In the code editor add the using statements as shown in Code Box 2-1 in the code section of your form.

Code Box 3-1 "using" class in your example

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using NExcel;
using SharpSim;
```

3.2 Writing the model code

- Define the variables given in Code Box 2-2 in the global declarations section of your form. This time you do not need to define all events and edges here, since these will be read from Excel file.

Code Box 3-2 Defining your simulation model objects

```
public partial class Example1 : Form
{
        Simulation sim;
        int ID;
        int S;

        public Example1()
        {
            InitializeComponent();
        }
}
```

The constructor of the form includes a call to initialize components.

- On the button click event, insert the code in Code Box 2-3.

Code Box 3-3 Instantiating the simulation object

```
private void button1_Click(object sender, EventArgs e)
{
      this.openFileDialog1.FileName = "*.xls";
      this.openFileDialog1.InitialDirectory = Application.StartupPath;
      if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
      {
      //read the excel file
            string filename = openFileDialog1.FileName;
            Workbook workbook = Workbook.getWorkbook(filename);

            sim = new Simulation(true, 3, true);

            sim.CreateEvents(workbook.getSheet("Events"));
      //State change listeners
            Run(sim.events["1"]);
            Arrival(sim.events["2"]);
            Start(sim.events["3"]);
            EndService(sim.events["4"]);
            Terminate(sim.events["5"]);

            sim.CreateEdges(workbook.getSheet("Edges"));
            sim.CreateStats(workbook.getSheet("Stats"));
      }
      sim.StartSimulationThread();
}
```

First two statements and the “if” statement are about opening the Excel file. You specify the file extension and the start-up path. The “if” statement checks whether the file chosen by the Open File Dialog exists. If the file opening is successful then a Workbook instance is created. This part comes from NExcel.

As in tutorial 1, a simulation instance is created by specifying tracklist on/off, number of replications, and randomizing seed on/off.

Sim.createEvents method is a SharpSim method. It reads the “Events” (or any other worksheet with the desired format) worksheet of the Excel file. The format of this worksheet is given below. The first row must include the labels; “Event No”, “Name”, “Priority”, and “Exec.Time” respectively. The first column is the integer identifier of events. The second column is a text area to name events. The third column is the priority (an event with bigger priority value is executed first). The last column is the time scheduled time of an event. If a value is given, at the beginning of the simulation, these events are added to the Future Event List (FEL). Others which have no Exec.Time value are the events that are scheduled during the simulation run.

	A	B	C	D	E
1	Event No	Name	Priority	Exec.Time	
2	1	Run	1	0	
3	2	Arrival	4		
4	3	StartService	2		
5	4	EndService	3		
6	5	Terminate	5	1000	

As in the first tutorial, you need to write state change listeners for every event. An event listener takes one argument of event type. You can directly specify an event from the simulation and using IDs.

CreateEdges method reads the edges data from “Edges” worksheet. The first column is a text label for the edge. It includes IDs of the events as text. Following two columns are the names of source and the target events. “Inter.Time” is the time on the edge. Remember that you need to specify a time delay on edges. This means that when executing the source event after a delay of time the target event must be executed (in other words scheduled in the FEL). This column is to specify a deterministic time delay. Whereas in the following three columns you enter some values to specify “stochastic” time delays. In the “Dist”, you enter a distribution name. In “Mean” and “Std.Dev” you enter first two parameters of the distribution (if the distribution is 1-parameter-dist then you

enter the parameter value into the “mean” cell). Following distributions are included in version 1.1:
"exponential", "normal", “uniform", "gamma", "chiSquare", "inverseGamma", "weibull", "cauchy", "studentT", "laplace", "logNormal", "beta".

	A	B	C	E	F	G	H	I	J
1	Edges	Source	Target	Inter.Time	Dist	Mean	Std.Dev.	src	trg
2	1-2	Run	Arrival	0				1	2
3	2-2	Arrival	Arrival		exponential	5		2	2
4	2-3	Arrival	StartService	0				2	3
5	3-4	StartService	EndService		exponential	5		3	4
6	4-3	EndService	StartService	0				4	3

- Explanations of “State Change Listeners” are done in the previous tutorial. Note that 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 this form will be executed. The links are done by “sim.events["1"]” lines in the code. The sim variable has an events collection and ID number as string is the key in this collection.

3.3 State Change handlers and Running the application

All the state change handlers, the methods which are linked to simulation events, are the same as in the first tutorial.

Creating the Entity class and running the application are explained in the first tutorial.

4 TUTORIAL 3

In this tutorial we present a generic model of a production/inventory system which is presented by Altiok and Melamed, 1997. The system to be modeled is a production facility which is subject to failures. There is one type of product. The products produced at this facility are stored in a warehouse. This generic model illustrates how an inventory control policy regulates flow of products between production and inventory facilities.

First two tutorials used SharpSim v.1.0 however this tutorial is using SharpSim v.1.1. The main addition to this version is the addition of "Cancelling Edge". Some other improvements as published at the official website are also made

4.1 Problem Statement

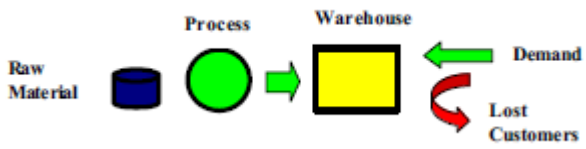
The production system is composed of three stages;

1. Filling each container unit (e.g., bottles)
2. Sealing each unit
3. Placing labels on each unit.

For modeling purposes, the processing times of individual stages are combined into a single-stage processing time.

Figure 4-1 depicts a schematic diagram of the system (taken from Altiok and Melamed, 2007). Raw-materials are stored in a storage area and are used to produce the products in the production process. Finished products are sent to the warehouse. Customers arrive at the warehouse with product requests (demands), and if a request cannot be fully satisfied by on-hand inventory, the unsatisfied portion represents lost product order.

Figure 4-1: A production/inventory system (taken Altiok and Melamed 2007)



The following assumptions are made:

• There is always sufficient raw material in storage, so the process never starves.

• Product processing is carried out in lots of 5 units, and finished lots are placed in the warehouse. Lot processing time is uniformly distributed between 10 and 20 minutes.

• The production process experiences random failures, which may occur at any point in time. Times between failures are exponentially distributed with a mean of 200 minutes, while repair times are normally distributed, with a mean of 90 minutes and a standard deviation of 45 minutes.

• Warehouse operations implement (r, R) inventory control policy. The warehouse has a capacity (target level) of R=500 units, so that the production process stops when the inventory in the warehouse reaches the target level. From this point and on, the production process becomes blocked and remains inactive until the inventory level drops to or below the reorder point r=150 units (the production process resumes as soon as the reorder level is down-crossed.) Note that this is a convenient policy when a resource needs to be shared among different types of products. For instance, when the production process becomes blocked, it may actually be assigned to another task or product that is not part of this system.

• The inter-arrival times between successive customers are uniformly distributed between 3 to 7 hours, and individual demand sizes are distributed uniformly between 50 and 100 units. For programming simplicity, demand sizes are allowed to be any real number in this range. On customer arrival, the inventory is immediately checked. If there is sufficient stock on hand, that demand is promptly satisfied. Otherwise, the unsatisfied portion of the demand is lost.

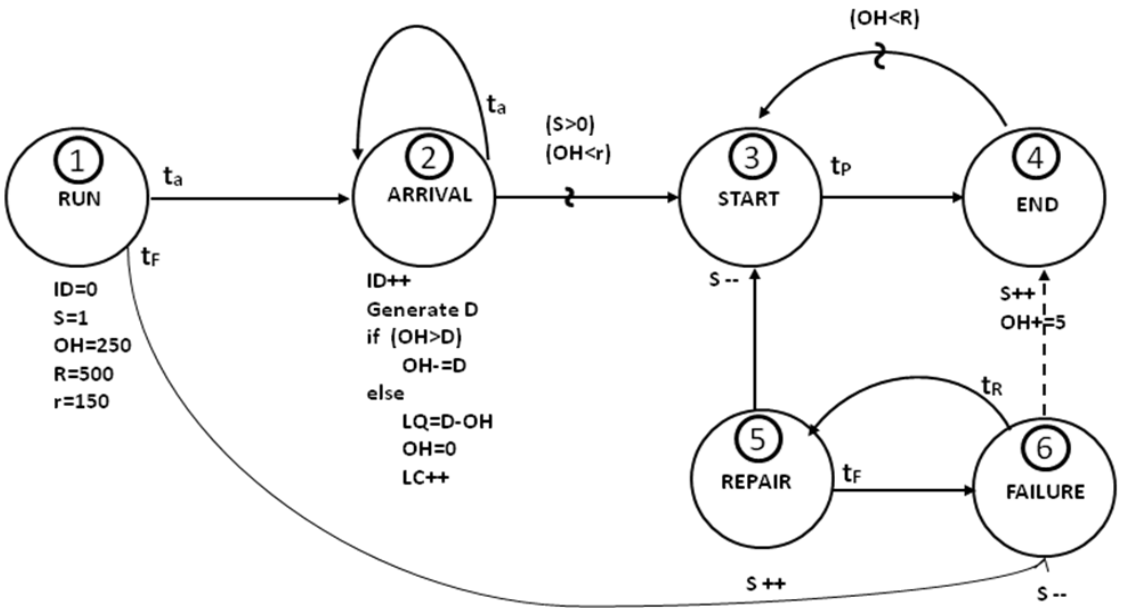
Some typical performance measures of interest are:

- Average inventory level
- Percentage of customers whose demand is not completely satisfied
- Average lost demand size given that it is not completely satisfied

4.2 Event Graph of the System

The event graph of the system described in the previous section is given in **Figure 4-2**, as it is crucial before building a SharpSim model.

Figure 4-2: Event Graph for production/inventory model



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 1, the OH (On-hand inventory) to 250, the R (inventory position) to 500 and the r (reorder point) to 150. This is the initialization phase and Run event is a triggering event for the rest of the events. We need to set the conditions of Edge [1-2] and Edge [1-6] to true.

When the Arrival event occurs you first need to increase ID by one and generate random D (demand) according to uniform distribution with parameters 50 and 100. Later you check on-hand inventory. If it is more than D, you satisfy the customer. Otherwise, LC (Lost Customer) incremented by one, calculate LQ (Lost Quantity) and set OH to 0. Finally, you need to set the condition on edge between Arrival and Start events based on S (number of available servers) and on-hand inventory. Note that <R, r> inventory model policy is: when on-hand inventory level hits r order up to R.

When the StartProduction event occurs, number of available servers is decremented by one.

When the EndProduction event occurs, number of available servers is incremented by one and on-hand incremented by five. Scheduling a Start event is possible if on-hand is less than R.

When the Failure event occurs, scheduled first EndProduction event in Future Event List (FEL) is deleted. Edge [6-4] is a **cancelling edge** and this is one of the new features of SharpSim. Also, number of available servers is decremented by one to cancel Start event condition. Schedule next Repair event.

When the Repair event occurs, number of available servers is incremented by one.

We mentioned creating C# Project, adding SharpSim as a reference and writing the model code using SharpSim in Tutorial 1. We will explore new properties of SharpSim, learn how to use them and some recommendations for users by creating production/inventory model at the following sections. Note that in this tutorial, as in Tutorial 1, Excel input is not used.

4.3 Creating Production/Inventory Model

4.3.1 Setting the simulation form

- Add a form in your project and add a rich text box and a button.
- In the code editor add the using statements as shown in Code Box 2-1 in the code section of your form.

Code Box 4-1 “using” class in your example

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using NExcel;
using SharpSim;
```

4.3.2 Creating the model

- Define the variables given in Code Box 2-2 in the global declarations section of your form. Note that you must define every event and edge as seen in Figure 4-2.

Code Box 4-2 Defining your simulation model objects

```
public partial class ProductionExample : Form
{
    Simulation sim;

    Event eRun;
    Event eArrival;
    Event eStartProduction;
    Event eEndProduction;
    Event eRepair;
    Event eFailure;
    Event eTerminate;

    Edge edge1_2;
    Edge edge2_2;
    Edge edge2_3;
    Edge edge3_4;
    Edge edge4_3;
    Edge edge1_6;
    Edge edge5_3;
    Edge edge5_6;
    Edge edge6_5;
    Edge cedge6_4; //canceling edge
}
```

- On the button click event, instantiate the SharpSim simulation object as shown in Code Box 2-3.

  Code Box 4-3 Instantiating the simulation object

```
private void button1_Click(object sender, EventArgs e)
{
   sim = new Simulation(true, 10, false);
```

- Instantiate the Events as shown in Code Box 2-4. You need to create every event seen in Figure 4-2. Note that the Terminate event does not exist in the EG but since an event to stop the model is required, you explicitly need to add this event to your model code.

  Code Box 4-4 Creating the events

```
eRun = new Event("1", "Run", 1, 0);
eArrival = new Event("2", "Arrival", 6);
eStartProduction = new Event("3", "StartProduction", 4);
eEndProduction = new Event("4", "EndProduction", 5);
eRepair = new Event("5", "Repair", 2);
eFailure = new Event("6", "Failure", 3);
eTerminate = new Event("7", "Terminate", 7, 1000000);
```

- 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 eRun event occurs in SharpSim, Run method of this form (which will be explained later) will be executed.

Code Box 4-5 Adding State Change Listeners

```
//State change listeners
Run(eRun);
Arrival(eArrival);
StartProduction(eStartProduction);
EndProduction(eEndProduction);
Repair(eRepair);
Failure(eFailure);
Terminate(eTerminate);
```

- Instantiate the Edges. You need to create every edge, the arrows between events, in Figure 4-2.

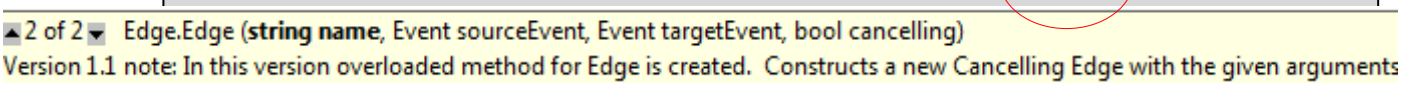
  Code Box 4-6 Instantiating Edges

```
edge1_2 = new Edge("1-2", eRun, eArrival);

//Version 1.1 property
List<object> edge1_2Dist = new List<object>();
edge1_2Dist.Add("uniform");
edge1_2Dist.Add(180.0);
edge1_2Dist.Add(420.0);
edge1_2.distribution = edge1_2Dist;

//Version 1.0 edge type
//edge1_2.dist = "uniform";
//edge1_2.param1 = 180.0;
//edge1_2.param2 = 420.0;
```

```
edge2_2 = new Edge("2-2", eArrival, eArrival);
edge2_2.dist = "uniform";
edge2_2.param1 = 180.0;
edge2_2.param2 = 420.0;
edge2_3 = new Edge("2-3", eArrival, eStartProduction);
edge3_4 = new Edge("3-4", eStartProduction, eEndProduction);
edge3_4.dist = "uniform";
edge3_4.param1 = 10;
edge3_4.param2 = 20;
edge4_3 = new Edge("4-3", eEndProduction, eStartProduction);
edge1_6 = new Edge("1-6", eRun, eFailure);
edge1_6.dist = "exponential";
edge1_6.param1 = 200.0;
edge5_3 = new Edge("5-3", eRepair, eStartProduction);
edge5_6 = new Edge("5-6", eRepair, eFailure);
edge5_6.dist = "exponential";
edge5_6.param1 = 200.0;
edge6_5 = new Edge("6-5", eFailure, eRepair);
edge6_5.dist = "normal";
edge6_5.param1 = 90.0;
edge6_5.param2 = 45.0;
//Version 1.1 property
//Cancelling edge last parameter of edge indicates cancelling edge
cedge6_4 = new Edge("6-4", eFailure, eEndProduction, true);
```



- As it's shown in Code Box 4-6, version 1.1 has an overloaded method for ComputeValue (in Edge.cs). Compute Value takes a List in which the first element is the distribution name. The modeler can set the time distribution and the distribution parameters on an edge by using these two overloaded methods.
- At version 1.1, following distributions are added to SharpSim library.
    - Exponential
    - Normal
    - Uniform
    - Gamma
    - Chi Square
    - Inverse Gamma
    - Weibull
    - Cauchy
    - Student T
    - Laplace
    - LogNormal
    - Beta
- At version 1.1, also, the cancelling edge mechanism is added to SharpSim. There is a boolean parameter in the Edge class as shown with a red circle in **Code Box 4-6**. This parameter is set to true for a cancelling edge.

- Complete source code for button click event is shown in Code Box 4-7.

Code Box 4-7 Complete Source Code for Button Click

```
private void button1_Click(object sender, EventArgs e)
{
sim = new Simulation(false, 1000, false);

eRun = new Event("1", "Run", 1, 0);
eArrival = new Event("2", "Arrival", 6);
eStartProduction = new Event("3", "StartProduction", 4);
eEndProduction = new Event("4", "EndProduction", 5);
eRepair = new Event("5", "Repair", 2);
eFailure = new Event("6", "Failure", 3);
eTerminate = new Event("7", "Terminate", 7, 1000000);

//State change listener
Run(eRun);
Arrival(eArrival);
StartProduction(eStartProduction);
EndProduction(eEndProduction);
Repair(eRepair);
Failure(eFailure);
Terminate(eTerminate);
//Create edge (event schedule)
edge1_2 = new Edge("1-2", eRun, eArrival);
//Version 1.1 property
List<object> edge1_2Dist = new List<object>();
edge1_2Dist.Add("uniform");
edge1_2Dist.Add(180.0);
edge1_2Dist.Add(420.0);
edge1_2.distribution = edge1_2Dist;
//Version 1.0 edge type
//edge1_2.dist = "uniform";
//edge1_2.param1 = 180.0;
//edge1_2.param2 = 420.0;
edge2_2 = new Edge("2-2", eArrival, eArrival);
edge2_2.dist = "uniform";
edge2_2.param1 = 180.0;
edge2_2.param2 = 420.0;
edge2_3 = new Edge("2-3", eArrival, eStartProduction);
edge3_4 = new Edge("3-4", eStartProduction, eEndProduction);
edge3_4.dist = "uniform";
edge3_4.param1 = 10;
edge3_4.param2 = 20;
edge4_3 = new Edge("4-3", eEndProduction, eStartProduction);
edge1_6 = new Edge("1-6", eRun, eFailure);
edge1_6.dist = "exponential";
edge1_6.param1 = 200.0;
edge5_3 = new Edge("5-3", eRepair, eStartProduction);
edge5_6 = new Edge("5-6", eRepair, eFailure);
edge5_6.dist = "exponential";
edge5_6.param1 = 200.0;
edge6_5 = new Edge("6-5", eFailure, eRepair);
edge6_5.dist = "normal";
edge6_5.param1 = 90.0;
edge6_5.param2 = 45.0;
//Version 1.1 property
//Cancelling edge last parameter of edge indicates cancelling edge
cedge6_4 = new Edge("6-4", eFailure, eEndProduction, true);
sim.CreateStats("SOHtimeAverage");
sim.CreateStats("Unsatisfied Customer");
sim.CreateStats("Lost Demand Size");
sim.CreateStats("Total Customer");
sim.CreateStats("Unsatisfied Ratio");
sim.Run();
}
```

- “sim.CreateStats("SOHtimeAverage");” is written to collect statistics for the average on-hand inventory. Average on-hand inventory is a time weighted average. This type of statistics is also a new feature in version 1.1.

- Final line “sim.Run()” calls the main simulation run method and the simulation starts.

4.4 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.

4.4.1 Run Event state change

Translation of the event graph in Figure 4-2 is as follows;

- Set the initial statistical data values to 0.
- Set variable S to 1, which means that we have initially 1 production machine.
- Set OnHand to 250, which means that initial value of on-hand inventory is 250. Also add this value to Stats dictionary.
- Set BatchSize to 5, which means that every production event will produce 5 products (in batches).
- Set r to 150, which means that reorder point is equal to 150.
- Set bigR to 500 which means that the system will produce up to 500.

Code Box 4-8 Run Event state change handler

```
public void Run(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
     ID = 0;
     UnsatisfiedCustomer = 0;
     LostDemandSize = 0;
     S = 1;
     OnHand = 250;
     Stats.CollectStats("SOHtimeAverage", new double[2]{ Simulation.clock, OnHand });
     BatchSize = 5;
     r = 150;
     bigR = 500;
    };
}
```

4.4.2 Arrival Event state change

On an Arrival event (Code Box 2-7);

- Increase Customer ID by one.
- Generate random demand.

> If the modeler requires to generate random numbers inside the model, "rnd' static variable in RandomGenerate class must be used, such as "**`RandomGenerate.rnd.Next(50, 100);`**". C#'s random number generator must NOT be used.

- Check the condition of OnHand inventory according to Demand, if it's available then sell them, otherwise;
    - Sell all OnHand then set it to 0.
    - Compute LostDemandSize
    - Increase UnsatisfiedCustomer by one.
    - Collect statistical data.
- Set the edge between events 2 and 3 condition to True, if OnHand is less than or equal to r and the number of servers available is greater than 0, and False otherwise.

Code Box 4-9 Arrival Event state change handler

```
public void Arrival(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
     ID++;
     Demand = RandomGenerate.rnd.Next(50, 100);

     if (OnHand > Demand)
     {
         OnHand -= Demand;
         Stats.CollectStats("SOHtimeAverage", new double[2] { Simulation.clock, OnHand });
     }
     else
     {
         LostDemandSize += Demand - OnHand;
         Stats.CollectStats("Lost Demand Size", LostDemandSize);
         OnHand = 0;
         Stats.CollectStats("SOHtimeAverage", new double[2] { Simulation.clock, OnHand });
         UnsatisfiedCustomer++;
     }
     if ((OnHand<r) && (S > 0))
         edge2_3.condition = true;
     else
         edge2_3.condition = false;
    };
}
```

4.4.3 StartProduction Event state change

On the Start event (Code Box 2-8);

- Decrease the number of server available by one

Code Box 4-10 Start Production Event state change handler

```
public void StartProduction(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        S--;
    };
}
```

4.4.4 EndProduction Event state change

On the EndProduction event (Code Box 2-9);

- Increase the number of available servers by one.
- Add BatchSize to OnHand.
- Add a statistics collection line.
- If OnHand is less than bigR, set the condition on edge between events 4 and 3 to true. This means that the system will produce more products.
- Otherwise, set the condition to false

Code Box 4-11 EndProduction Event state change handler

```
public void EndProduction(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
     S++;
     OnHand += BatchSize;
     Stats.CollectStats("SOHtimeAverage", new double[2] { Simulation.clock,
                                                          OnHand });
     if (OnHand < bigR)
         edge4_3.condition = true;
     else
         edge4_3.condition = false;
    };
```

4.4.5 Repair Event state change

On the Repair event (Code Box 4-12);

- Increase the number of available server.
- If OnHand is less than r, set the condition on edge between events 5 and 3 to true. This means that the system will start production.
- To schedule next failure, set the condition on edge between events 5 and 6 to true.

Code Box 4-12 Repair Event state change handler

```
public void Repair(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        S++;
        if (OnHand < r)
        {
            edge5_3.condition = true;
        }
        else
        {
            edge5_3.condition = false;
        }
        edge5_6.condition = true;
    };
}
```

4.4.6 Failure Event state change

On the Failure event (Code Box 4-13);

- Decrease the number of available server.
- Scheduled first EndProduction event in FEL is deleted. Edge [6-4] is a **cancelling edge** and this is one of the new features of SharpSim.
- To schedule next Repair event, set the condition on edge between events 6 and 5 to true.

Code Box 4-13 Failure Event state change handler

```
public void Failure(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        S--;
        cedge6_4.cancellingEdge = true;
        edge6_5.condition = true;
    };
}
```

4.4.7 Terminate Event state change

On the Terminate event (Code Box 4-14) which is not in the event graph;

- Write a message to indicate that the replication is ended.
- Add collected statistics to the global statistics dictionary.

Code Box 4-14 Terminate Event state change handler

```
public void Terminate(Event evt)
{
 evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
 {
  unsatisfiedRatio = (double) UnsatisfiedCustomer / ID;
  richTextBox1.Text += "Replication No : " + Simulation.replicationNow + " ended." + "\n";
  Stats.AddDataToStatsGlobalDictionary("SOHtimeAverage",
                                       Stats.Dictionary["SOHtimeAverage"].timeWeightedAverage);
  Stats.AddDataToStatsGlobalDictionary("Unsatisfied Customer", UnsatisfiedCustomer);
  Stats.AddDataToStatsGlobalDictionary("Total Customer", ID);
  Stats.AddDataToStatsGlobalDictionary("Lost Demand Size", Stats.Dictionary["Lost Demand Size"].mean);
  Stats.AddDataToStatsGlobalDictionary("Unsatisfied Ratio", unsatisfiedRatio);
 };
}
```

4.5 Running the application

Select Program.cs on the right-side and double click it and be sure that the name of the form is written at Application.Run row as shown in Figure 2-5. Now, the model is ready to run. Build your application and run it. You will see two windows as seen in Figure 2-6.

Figure 4-3 Program code

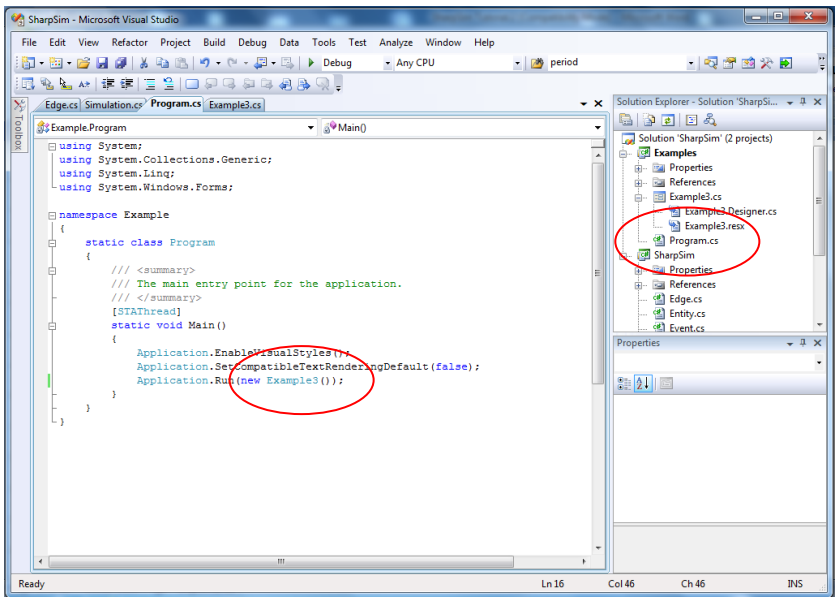
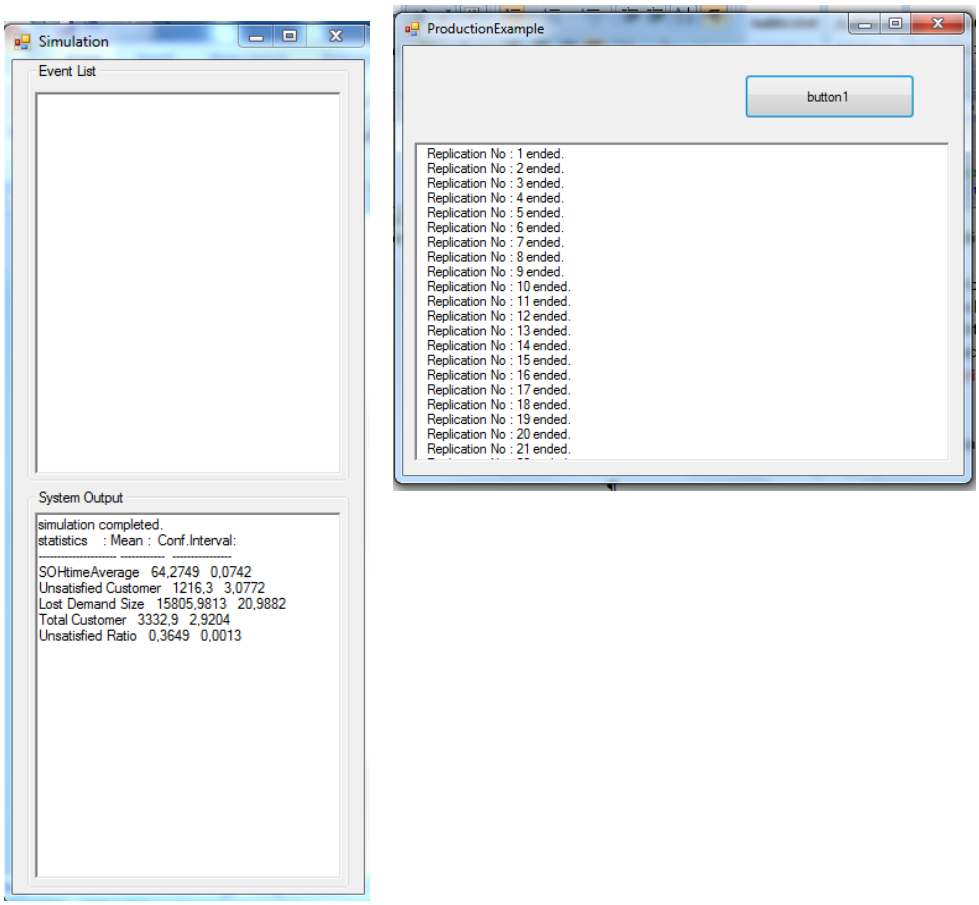


Figure 4-4 Output windows



REFERENCES

Altiok, T., B.Melamed, 2007, Simulation Modeling and Analysis with Arena, Elsevier.