Visualization of the Process Interaction Worldview in Discrete Event Simulation

Thomas Tracey, Brian Dilinila, James F. Leathrum, Jr., Roland R. Mielke
Department of Modeling, Simulation and Visualization Engineering
Old Dominion University
Norfolk, Virginia
ttrac005@odu.edu, bdili002@odu.edu, jleathru@odu.edu, rmielke@odu.edu

ABSTRACT

A process interaction model is a specific form of modeling paradigm representing the interactions involved in an entity moving through some specific process. While there exist many ways of viewing and implementing process interaction models and diagrams, there does not exist a visualization method to denote scheduling and execution of events during runtime of a simulation. During runtime, the scheduling of events, execution of events, and interactions between processes are all hidden and unobservable throughout the course of the simulation. This lack of knowledge about the underlying processes in action, as well as the events that affect the system state, often hinders the understanding of the model at hand. This creates problems in teaching and introducing discrete event simulation to students because there is no visual component of the program that represents the underlying processes occurring during runtime.

The authors propose a visualization approach to displaying the scheduling and execution of events in process interaction models. This visualization approach highlights event scheduling and execution, as well as interactions between processes. The authors' methodology for this approach is focused on building upon examples of process interaction models by highlighting triggers, event scheduling, and event execution during process interaction simulations. The authors found that the development of this visualization method is very useful for highlighting key factors in process interaction simulations during runtime.

ABOUT THE AUTHORS

Thomas Tracey is a Senior in the undergraduate Modeling and Simulation Engineering program at Old Dominion University. He serves as an undergraduate teaching assistant for Modeling and Simulation courses.

Brian Dilinila is a Senior in the undergraduate Modeling and Simulation Engineering program at Old Dominion University. He is currently an intern under Combat Direction Systems Activity Dam Neck representing NAVSEA at and providing assistance to Joint Staff J7.

James Leathrum is an Associate Professor in the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He earned the Ph.D. in Electrical Engineering from Duke University. His research interests include simulation software design, distributed simulation, and simulation education. His e-mail address is jleathru@odu.edu.

Roland Mielke is a University Professor in the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He earned the Ph.D. in Electrical and Computer Engineering from the University of Wisconsin-Madison. His research interests include system theory, the theory and application of simulation, and simulation education. His e-mail address is rmielke@odu.edu.

Visualization of the Process Interaction Worldview in Discrete Event Simulation

Thomas Tracey, Brian Dilinila, James F. Leathrum, Jr., Roland R. Mielke
Department of Modeling, Simulation and Visualization Engineering
Old Dominion University
Norfolk, Virginia
ttrac005@odu.edu, bdili002@odu.edu, jleathru@odu.edu, rmielke@odu.edu

INTRODUCTION

It is well understood that modelers and simulation developers need to understand the underlying mechanisms that simulation software uses to manage the simulation, such as how simultaneous events are handled (Schriber, Brunner, and Smith, 2016). Currently, the standard method for supporting simulation development in observing the low-level effect of these mechanisms is through the use of event traces (Schriber, Brunner, and Smith, 2016). Understanding this relationship is particularly important during the education process of teaching software development of simulation models. Students have difficulty relating the events identified at the modeling level to the scheduling and execution of events as controlled by a simulation executive during software development. Collins, Dumaliang, Gonda, and Leathrum (2017) developed a visualization for the Event Scheduling Worldview; however, Leathrum, Mielke, Collins, and Audette (2017) identified the need to include the Process Interaction Worldview in the educational process due to the prevalence of simulation tools based on this worldview. This paper presents a new software tool that provides a visualization of relationship between the process interaction worldview and the underlying execution of events in the simulation executive as visualized via an event list and an event graph.

The modeling and simulation development process typically progresses from developing a system model based on the system under study, to developing an event or process model that implements the system model, and finally to implementing the simulation, as illustrated in Figure 1 for the Process Interaction Worldview. The simulation implementation commences with the development of a software model that is then implemented using a simulation tool, a simulation or general purpose programming language, or even an alternative environment such as a spreadsheet. Visualizations driven by the simulation usually tend to visualize either the system under study or the system model. It is suggested here that a visualization of the process interaction in parallel with a representation of the associated event model is advantageous for use in modeling and simulation education. The authors believe that a process flow graph representation of the process interaction coordinated with the associated event graph is an appropriate approach to show how the simulation implements and manages events, thus filling this need.

The approach taken to visualize the behavior of events is to develop a separate visualization tool with the necessary hooks to drive the visualization from a variety of simulation environments. The visualization includes a process flow graph for each process in the process interaction development with interactions between processes clearly identified. The activities associated to scheduling and execution of events within the process flow are highlighted during execution. In conjunction, an event graph is highlighted to indicate the current state of execution from the event perspective, and a linear event trajectory highlights the history of event execution and the known future events. The work in (Collins et. al. 2017) found the event graph and event trajectory a highly effective representation of the underlying event execution. The visualization tool provides the capability to control the advancement of the simulation execution, allowing the developer to step through the simulation or run it at a selected pace.

The visualization tool was developed for educational purposes to support a course in discrete-event simulation that utilizes simulation tools and spreadsheets, and a course on software design for discrete-event simulations where simulations are developed in a general purpose programming language. As a result, the initial implementation of the visualization tool is scaled for the class of problems usually studied in an undergraduate academic environment allowing students to place minimal calls to the visualization tool to notify it of all event schedules and executions and then observe how their code behaves relative to the model. However, if the use of the visualization tool proves successful, there are many ways it might be enhanced for the development and verification of larger scale simulations.

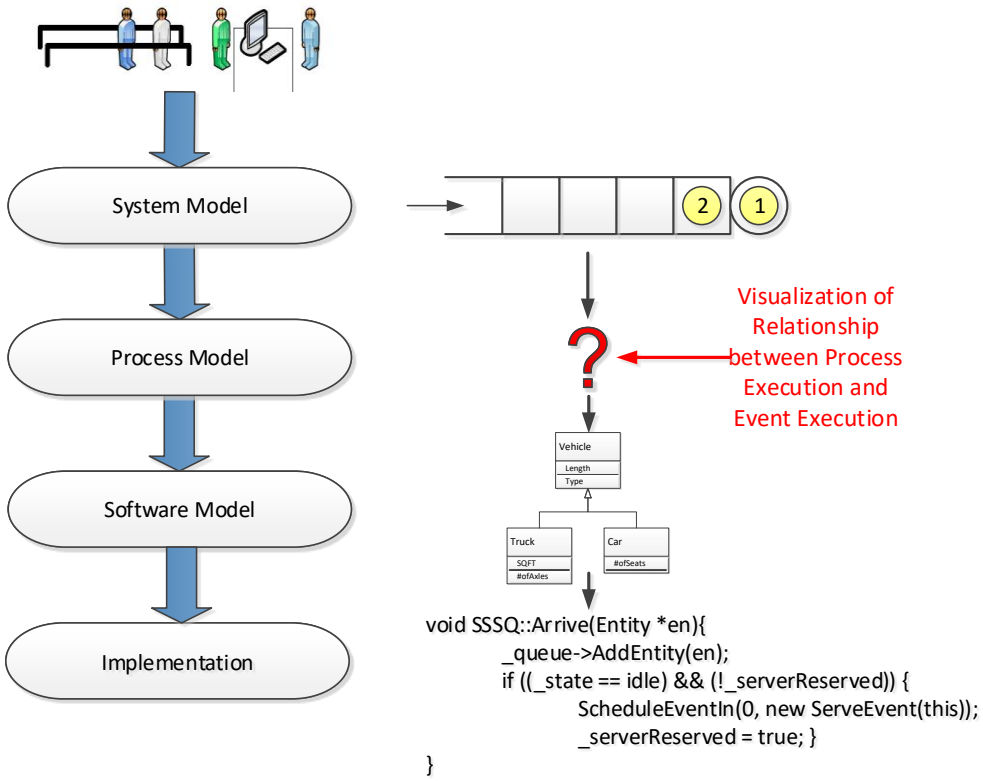

Figure 1. High Level View of Discrete Event Simulation Development.

The remainder of the paper is organized in six sections. First, *Related Works* are presented that concern the process interaction worldview, visualization of event scheduling and execution, and visual approaches for representing the relationship between software models and their implementation. Then, the topic of understanding underlying execution in programming and simulating process interaction models is discussed in *Difficulties in Comprehending the Execution of Process Interaction Software*. In the third section, the authors present their *Approach to Visualization of Process Interaction Worldview*. This section includes a brief introduction to process interaction modeling, and it includes the author's approach for visualizing event scheduling and execution of process interaction models. Next, an *Example Process Interaction Model* is introduced. The authors walk through the example by using figures to provide visual snapshots of the visualization approach in action. After the example, the *Visualization Tool Architecture* is presented. The architecture section explains how developers can integrate their code with the visualization approach. Finally, the authors present their *Conclusion*. In this section, the authors' present and future plans for adapting this visualization software for educational purposes are introduced.

RELATED WORKS

The concept of having access to the underlying event behavior is not new. Schriber, Brunner, and Smith (2016) discuss the concept of interactive model verification. This involves introducing breakpoints into the software or running an event trace so that event information is output to the developer to assist in verifying that the simulation behaves as the model specifies. These capabilities are common in discrete-event simulation tools. When combined with a visualization of the system model, the difficulty is that there often is a layer of abstraction between the model and the underlying event model that generates the timing of event activity. Therefore, the developer is directly comparing the simulation's event actions to the high-level model. This paper proposes introducing a low-level event model visualization to support process interaction.

Attempts at better displaying the way events are scheduled and executed in different modeling paradigms has also been explored. Collins, Gonda, Dumaliang, Leathrum, and Mielke (2017) introduced a visual representation for

scheduling and execution of events in the discrete event simulation of an event graph. By visually displaying the event set and event graph while simulating the graph, event scheduling and execution could be highlighted as the simulation advances through each event. The addition of including the event set visualization synchronized with event execution allowed users to better understand scheduling and execution of events in a discrete event simulation. The purpose of this paper is the development of a similar approach to displaying event scheduling and execution in discrete event simulation for educational purposes.

Process flow modeling is a popular modeling paradigm for describing process-oriented system behavior. It is commonly used in simulation software for process-oriented simulation languages such as Arena, where a process can be described as a time-ordered sequence of events that may encompass several activities (Choi and Kang, 2013). It is favored as a modeling paradigm for its capability as an entity-based modeling formalism, where studying only the activities of entities in the system is of interest.

IBM Rational Rose Modeler is an example of a tool that provides insight into the relationship between the software model and the software implementation. It is an object-oriented UML software design tool used for visual modeling and component construction of large-scale software applications. A user creates UML diagrams which are documented and used to generate code. The software was designed to enhance software design and development by emphasizing the importance of conceptual models and modular software architectures. It also provides a visual representation of the execution of the model in conjunction with the software execution. The purpose of this paper is the development of a similar capability in discrete event simulation.

Similar research on visualizing a scheduled sequence of activities, past, present, and future, has been conducted on the capability to visualize CPU schedulers. Suranauwarat (2007) presents a simulator with graphical animation intended to increase understanding on CPU scheduling algorithms. The simulator's animation contains a graph showing status and a separate view of corresponding color-coded timelines. These timelines consist of labeled blocks with time indicated above. This simple yet effective representation demonstrates the utility of the authors' approach to other problem domains.

DIFFICULTIES IN COMPREHENDING THE EXECUTION OF PROCESS INTERACTION SOFTWARE

The difficulty students have with understanding the execution of software implemented using a process interaction world view is that the software does not behave according to their previous experience with procedural and object-oriented programming. Rather than a logical order of execution following a single thread of execution, there are now multiple processes executing with control switching back and forth. This is generally implemented through the use of either multithreading or coroutines. The selection of which process to activate next is based on an underlying event trajectory from which a simulation executive selects the next event based on scheduled time and activates that process. Students are classically exposed to the event scheduling world view and develop an understanding of the execution of events; this extra layer of indirection results in confusion.

To support this, the authors utilize a two-fold process carried out between a course in discrete event simulation and a course in simulation software design. In discrete event simulation, students are exposed to both the process interaction world view and the event scheduling world view. In simulation software design, students learn how to develop a simulation executive to select and execute events as well as how to implement applications using both an event-based and a process-based approach. The first step in the process is to show the students how to identify events in a process-based model. The second is to let them observe the behavior of the process interaction world view both from the process based implementation and the underlying event scheduling/execution process. This paper provides a visualization to support the second phase of this process. This project focuses on the learner centeredness learning theory lens, as the approach is tailored on students' previous knowledge and simplified for their ease of use.

APPROACH TO VISUALIZATION OF PROCESS INTERACTION WORLDVIEW

To produce a visualization of the relationship between a process interaction model and the scheduling and execution of events in a discrete event simulation, an ability to represent both models is developed. Then each representation can be animated in parallel to illustrate the relationship. As an activity is performed on the process-based model, the

corresponding activities in the event model are presented. This approach involves displaying and synchronously animating the process interaction model with its corresponding event graph and event trajectory (a combined representation of the event history, currently executing event, and set of scheduled future events).

Process Interaction

Multiple approaches can be used to represent and define the management of events. Process interaction is a prominent worldview where the system is represented as a set of processes capable of performing logic to enact system state changes and passing time by either delaying for an amount of time or waiting for an interaction from another process. This worldview is popular in discrete event simulation tools such as Arena (Choi and Kang, 2013), where processes are visually represented as process blocks. It is utilized when it is necessary to observe and understand how different processes interact over time.

The authors have opted to represent the process interaction model as a set of process flowcharts with associated interactions identified. The reason for this is there is a clear relationship between the process flowchart and the software implementing it. In a process flowchart, blocks represent either a process to perform or a decision block (e.g. conditions) to control the logical flow of execution. Each block has incoming arcs and an outgoing arc which facilitate signal flow through the process. Special cases to support process interaction include "Wait for" blocks and trigger arcs. "Wait for" blocks schedule an event in the future after a given time duration to handle the passage of simulation time. Trigger arcs are arcs from one process to another, activating the triggered process by scheduling an associated event to occur immediately. Multiple blocks within a process flowchart can belong to a single event.

Students are required to understand the relationship between the simulation execution and the flowchart. Students create a graphical model (the process flowchart) to understand the step-by-step process of individual processes and their interaction. This model generally can be mapped easily to a software implementation. Figure 2 shows an example process flowchart model for a simple server/queue model. Note the special gray blocks indicating the passage of time and the dashed arcs indicating one process triggering another. The example system involves a process to model a source to create entities with an interarrival time between creations, and a process to model each entity's logic moving through the server/queue. Thus, the source periodically creates a new entity, initiating its process. The entities execute the server logic, waiting for a server to become available, when triggered that a server is available, delaying for a service time, and then prior to departure, triggering the next entity in the queue that a server is available.

The next step the student performs is to identify the events associated with the processes. This is done by finding the points in the processes where the process is activated after waiting for an outside influence, either the passage of time or a trigger from another process. Thus, the output arc from each gray block indicates the beginning of the logic for a new event, and all logic reachable from that point until entering a new gray block may be logically organized as a single event. In Figure 2, one event is defined for the source process (Create Entity) and three events are defined for each entity process (Arrive, Start Serve, and Done Serve). Students then identify the scheduling activities in the processes. These occur on gray blocks where a predefined time delay is provided (the blocks that wait for the interarrival time and the service time in the example), or a trigger between events (the arc from the source to the entity activating the entity process and the arc from one entity back to the next in line activating it for service).

When visualizing the execution of events, the process blocks must be highlighted synchronously with their corresponding events. This is done to show which processes belong to which events as well as which processes schedule which events. State changes are visualized with the execution and the scheduling of new events. This project follows the route of visualizing groups of blocks so the student may better understand the relationship between the event graph and the process flowchart. Another reason to do this, rather than visualize the sequence of process blocks, is to reduce the amount of effort required on the part of the developer. This is done to get the point across rather than to overwork the student developer. Figure 3 shows the difference between visualizing the sequence of process blocks (left) and visualizing groups of process blocks with their corresponding event.

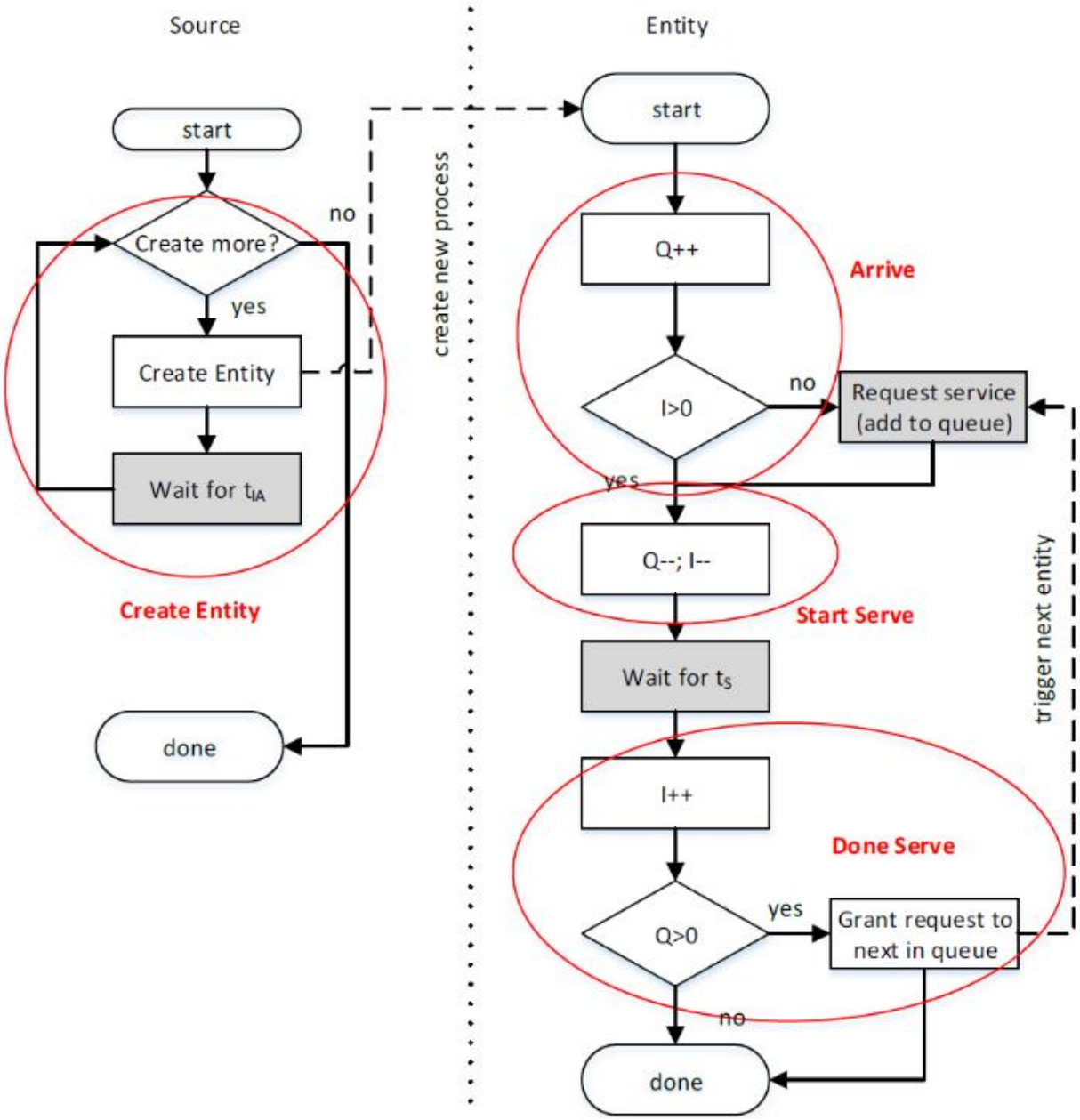

Figure 2. Example Process Flowchart. Adapted from "Proposed Unified Discrete Event Simulation Content Roadmap for M&S Curricula" by J. F, Leathrum, Jr. et. al., 2017. 2017 Winter Simulation Conference.

Visualization

The approach for visualization is to individually represent the process flowchart and the event scheduling/execution. To represent the event scheduling/execution, the authors utilized past successful experience with event graphs and event trajectories (Collins et. al. 2017). The event graph provides a graphical representation of events. Each event is represented by a node in the graph with associated state change logic. Arcs between nodes represent an event's ability to schedule another event at a scheduled future time. Arcs also can be labeled as conditional to represent logic within the scheduling event. The event graph representing the process flowchart in Figure 2 is provided in Figure 3.

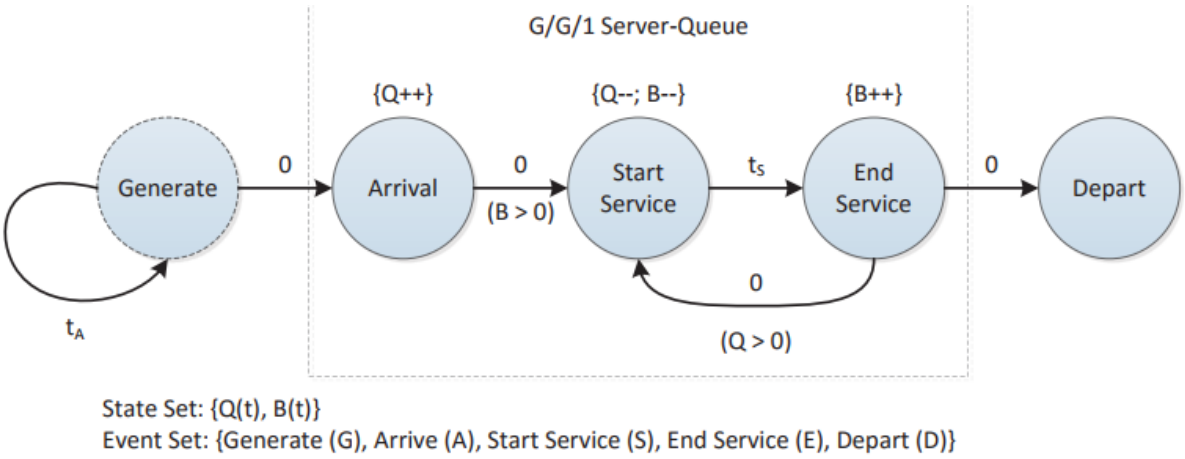

Figure 3. Event Graph Representation of the Process Flowchart in Figure 2.

With the provided visuals, the technique used to visualize behavior is to have the software driving the visual send a signal to the visual each time an event activity occurs, either starting the execution of an event or the scheduling of a new event. When a new event is executed, the associated blocks in the process flow are highlighted green as are the associated node in the event graph and event in the event trajectory. When an event is scheduled, the associated arc in the process flow is highlighted red as is the associated arc in the event graph. In addition, the process blocks for the scheduled event are highlighted blue and a new blue event is inserted in the event trajectory and outlined red.

Figure 4 shows the developed GUI provided by the project team demonstrating the server/queue system. A user-defined netlist file defines the graphics for the process flow on the left and the event graph on the bottom right. The GUI also provides functionality to allow the user to control the execution of the software driving the visualization. It allows the user to single step through the event activities one at a time. Alternatively, the user can define a delay applied to each step to allow continual execution at a rate that allows the student to observe behavior over time.

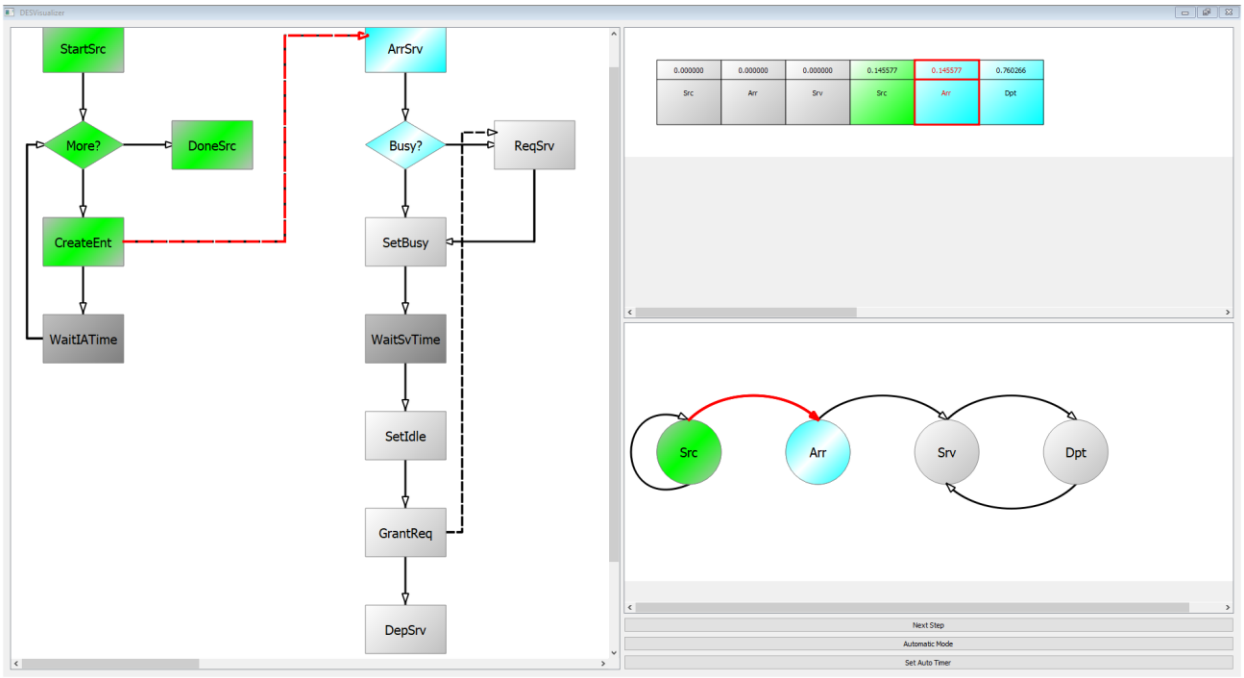

Figure 4. Visualization Tool Illustrating the Server/Queue Model from Figures 2 and 3.

EXAMPLE PROCESS INTERACTION MODEL

In this section, an example of the GUI visualization is provided. To help students better understand process interaction, the GUI advances through each step to show where one entity can make triggers to activate the process of another. In this example, the simulation has advanced to time t = 5.99 units. Multiple entities have already been created, arrived to the server, and completed service. At least one entity, entity *N*, is currently waiting in queue and is requesting service. Meanwhile, entity *M* completes its service. In Figure 5 the Depart event for entity *M* executes. The state of the system changes with the server becoming idle. As entity *M* departs, it grants a request to the next entity in line to execute its process. Figure 6 demonstrates the scheduling of a Start Service event that occurs as a result of entity *M* triggering entity *N*'s progress. The outward arc from "GrantReq" is dashed, signaling a trigger. Therefore, the scheduled Start Service event is scheduled to happen in zero time.

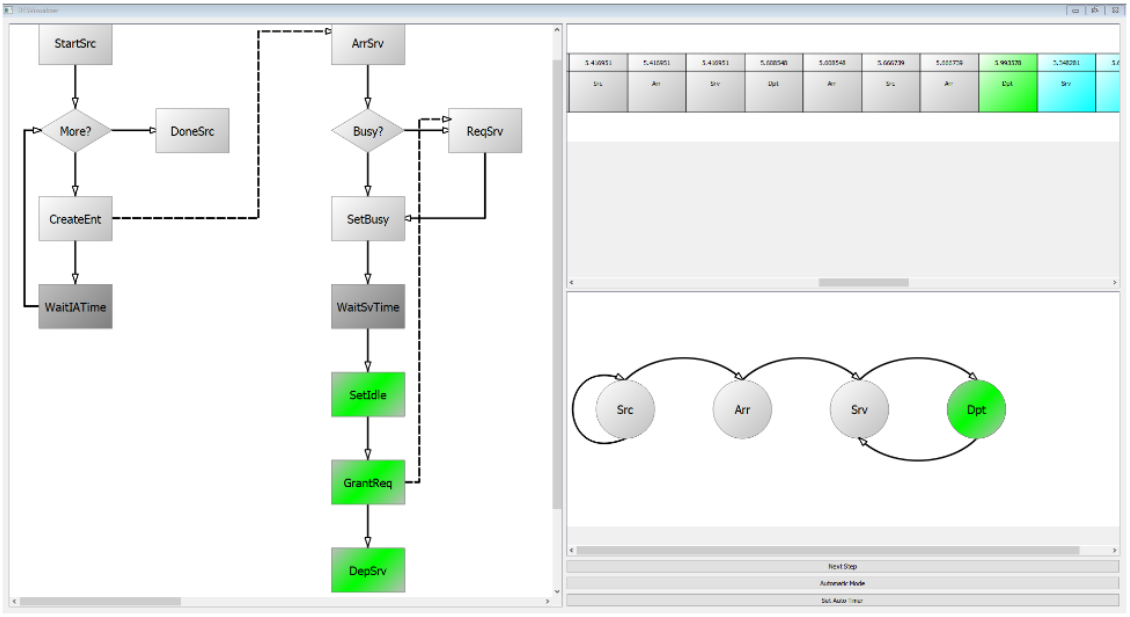

Figure 5. At t = 5.99 Units, Entity *M* Departs.

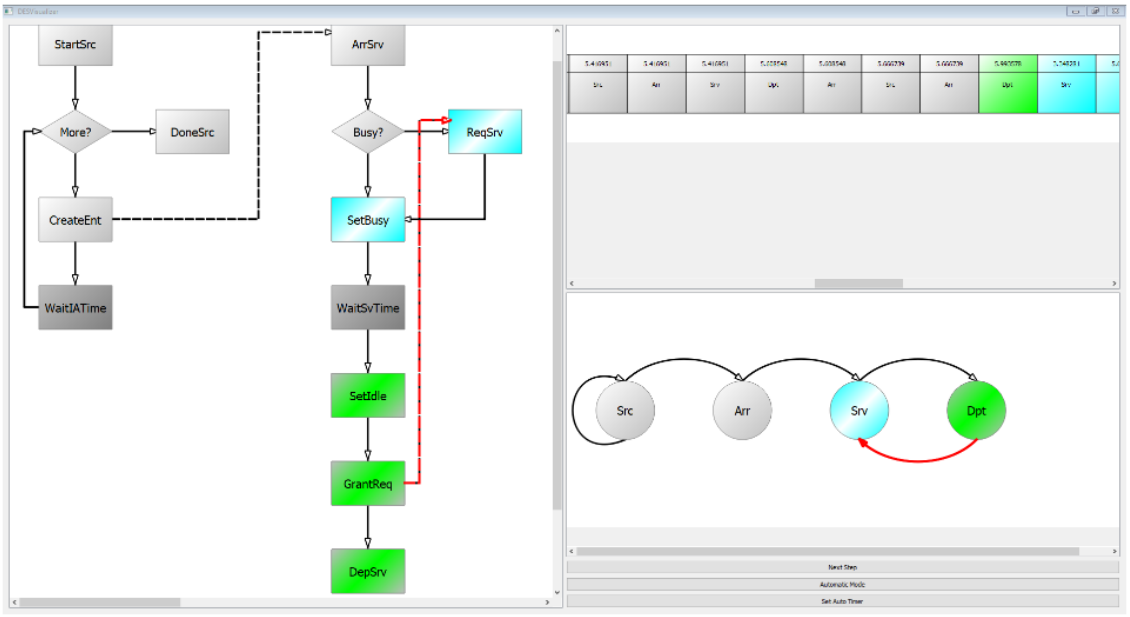

Figure 6. At t = 5.99 Units, the Departing Entity *M*'s Process Triggers Next Entity in Line to Begin Service.

Figure 7 shows that entity *N* now continues its process at the Start Service event execution. The server returns to being busy. Entity *N* continues its service for a given time distribution.

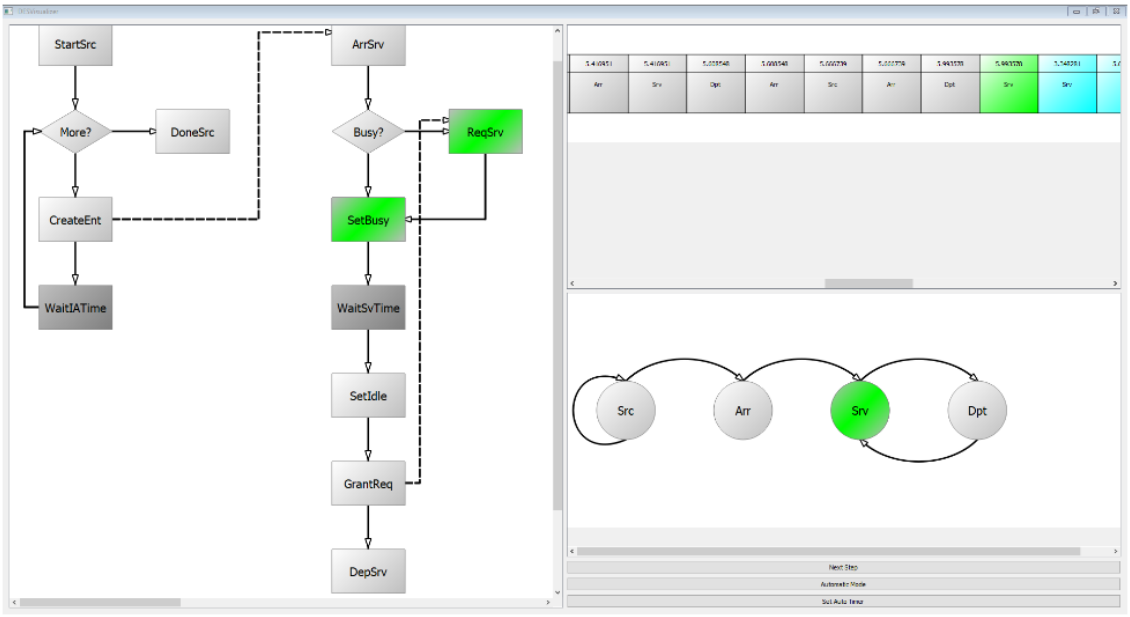

Figure 7. At t = 5.99 Units, the Triggered Entity *N* Continues its Process and Begins Service.

Figure 8 shows that the Start Service event also schedules a Depart event. The block "WaitSvTime" is a "WaitFor" block; therefore, it schedules the event Depart to occur at 8.39 time units.

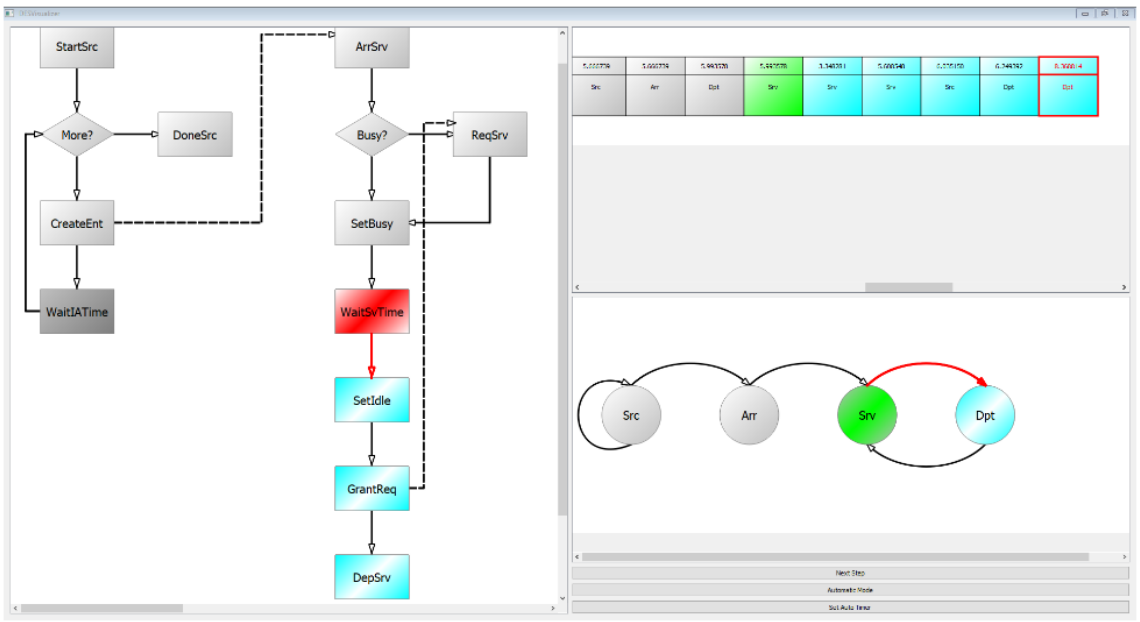

Figure 8. At t = 5.99 Units, the Event Start Service Schedules a Depart Event at t = 8.39 Units.

VISUALIZATION TOOL ARCHITECTURE

Three main components are needed to allow the GUI to communicate with the simulation: the parser file (netlist), the executable for the DESVisualizer GUI (built using Qt), and preparation to be completed for the developer's application

to send updates to the GUI. The system architecture is shown in Figure 9. Requests are sent from the developer's application software to the visualization. Separately defined requests for event execution and event scheduling are utilized to notify the visual of the activity being performed, and the API provided to the developer waits for a response from the visual before allowing the application to continue. On update completion, and the ensuing predefined delay or response time of the user, a reply is sent back to the API, allowing the application program to continue execution.

From the developer's application, the simulation follows the graphical model that is visualized by the GUI. Note that should the developer's application not behave correctly, a resulting abhorrent behavior will be observable in the visual. Thus, not only can the developer observe the relationship between his code and the model on which it was developed, he can utilize it to debug the behavior of his code.

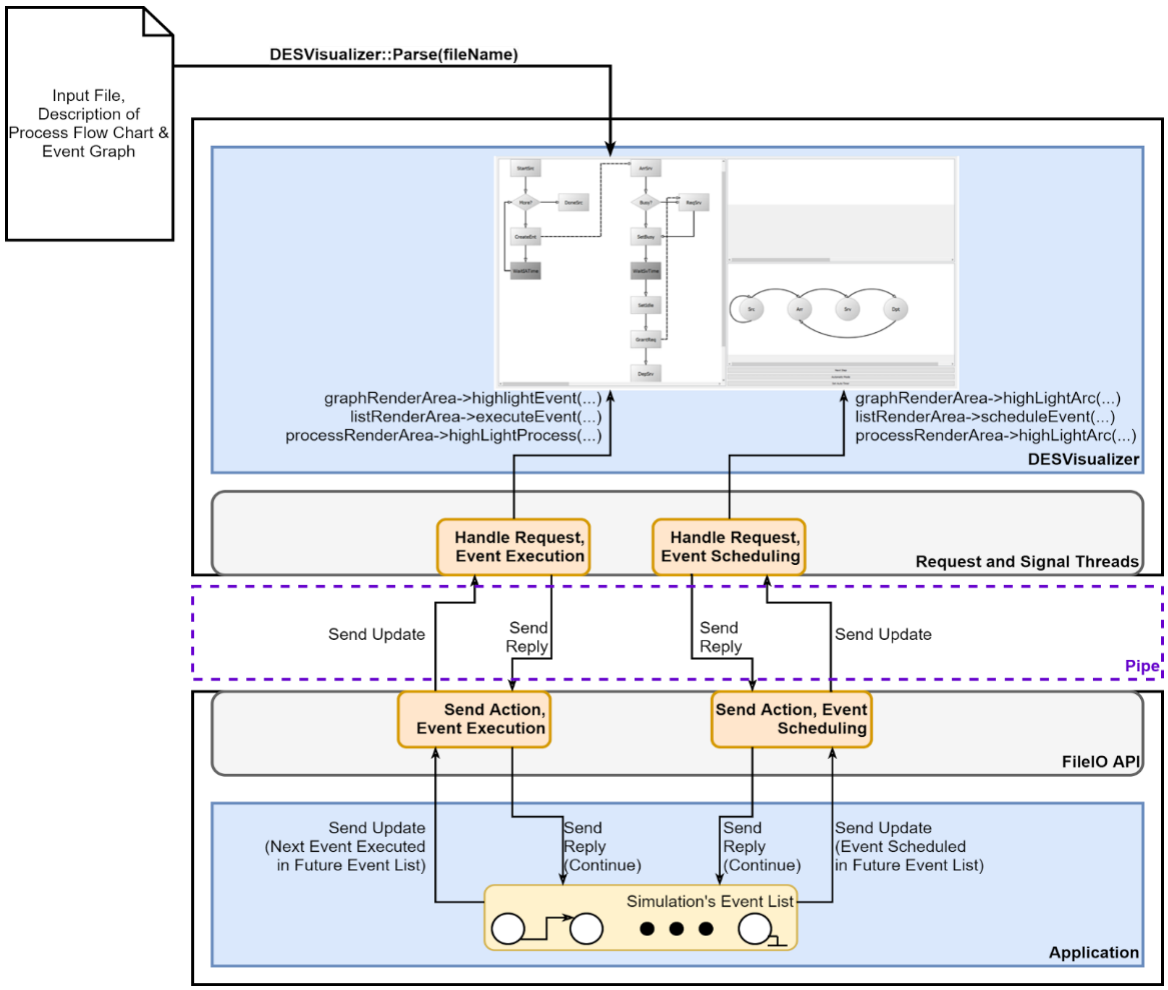

Figure 9. System Architecture of DES Visualizer Interacting with a Developer's Application and Netlist File.

To enable visualization of students' simulation software, the visualization must be integrated with the actual code. This requires the student to add the update calls to the visualization as denoted in Figure 9. This involves identifying the start execution of each event and the scheduling of each event. Example code is provided in Figure 10 for the process involved to implement the entity from Figure 2. The code is highlighted to clearly identify the relationship to Figure 2. The events are color coded as Arrive event – turquoise, Start Serve event – green, and Done Service event – magenta. The gray "WaitFor" blocks and triggers correspond with the gray highlighted code in Figure 10. Finally, the update calls to the visualization tool are highlighted yellow in the code.

The takeaway from the example code is the simplicity of adding the updates to the code. The student must clearly identify the corresponding events in the processes as annotated in Figure 2. With the corresponding code identified, a call to OutputExecuteEvent is made to notify the visualization that the next event is being executed, one for each of

the three events in the entity's processes. This results in the visualization indicating the associated event being executed by updating the highlighted portion of the process flow, event graph, and event trajectory. In addition, each time a new event is scheduled, either by waiting for a specified time (a wait for action) or by a process being triggered by another (a grant request action), an OutputScheduleEvent call is made to alert the visualization to highlight the edges in the process flow and event graph corresponding with the scheduling and to add the newly scheduled event to the event trajectory.

```
//begin Arrive Event
OutputExecuteEvent(1, GetCurrentSimTime());
Q++;
//end Arrive Event
if (I > 0) {
        _queue.Request(entity->GetTrigger()); }
OutputScheduleEvent(2, GetCurrentSimTime() + 0);
//begin Start Service Event
OutputExecuteEvent(2, GetCurrentSimTime());
Q--; I--;
Time tService = _serviceTime->GetRV();
OutputScheduleEvent(3, GetCurrentSimTime() + tService);
//end Start Service Event
entity->WaitFor(tService);
//begin Done Service Event
OutputExecuteEvent(3, GetCurrentSimTime());
I++;
if (Q > 0) {
        _queue.GrantRequest(); }
//end Done Service Event
```

Figure 10. Entity Process Source Code.

CONCLUSION

The visualization presented in this paper is currently being fielded within ODU's Modeling and Simulation courses. In the current academic year, the visualization is being used at both the graduate and undergraduate level as a demonstration tool in class to assist students' understanding of the concepts but will not be placed in their hands until further testing is completed. The demonstration will consist of the visualization running together with an already completed DES simulation. In the next academic year, students will utilize the GUI in conjunction with their own code to better understand their programs as well as the relationship between their process interaction model and the event graph. The authors are confident that students will benefit from the use of this visualization tool based on experiences with the work in (Collins et. al. 2017).

REFERENCES

Leathrum, J. F., Mielke, R. R., Collins, A. J. & Audette, M. A., (2017). "Proposed Unified Discrete Event Simulation Content Roadmap for M&S Curricula." *2017 Winter Simulation Conference (WSC)*.

Collins, S. C., Dumaliang, L. C., Gonda, N. D., Leathrum, J. F. Jr., & Mielke, R. R (*2017*). "Visualization of Event Execution in a Discrete Event System." *50th Annual Simulation Symposium (ANSS 2017)*, 2017.

Choi, B. K., & Kang, D. (2013). *Modeling and Simulation of Discrete-Event Systems*, Hoboken, NJ: John Wiley & Sons.

IBM, "*IBM Rational Rose Modeler*", retrieved Dec. 12, 2016, from http://www-03.ibm.com/software/products/en/rosemod.

Schriber, T., Brunner, D. & Smith, J., "Inside Discrete-Event Simulation Software: How it Works and Why it Matters," in *Proceedings of the 2016 Winter Simulation Conference*, Dec. 2016.

Suranauwarat, S. (2007). "A CPU Scheduling Algorithm Simulator". *2007 37th Annual Frontiers In Education Conference – Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 2007.