3-22-2019

The Effect of Modeling Simultaneous Events on Simulation Results

John M. Carboni

THE EFFECT OF MODELING SIMULTANEOUS EVENTS ON SIMULATION RESULTS

THESIS

John M. Carboni, 2nd Lieutenant, USAF

AFIT-ENG-MS-19-M-014

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-M-014

THE EFFECT OF MODELING SIMULTANEOUS EVENTS ON SIMULATION RESULTS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Software Engineering

John M. Carboni, BS

2nd Lieutenant, USAF

March 2019

AFIT-ENG-MS-19-M-014

THE EFFECT OF MODELING SIMULTANEOUS EVENTS ON SIMULATION RESULTS

John M. Carboni, BS

2nd Lieutenant, USAF

Committee Membership:

Dr. Douglas Hodson, PhD
Chair

Maj Jeremy R. Millar
Member

Dr. John O. Miller, PhD
Member

AFIT-ENG-MS-19-M-014

Abstract

This thesis explores the method that governs the prioritizing process for simultaneous events in relation to simulation results for discrete-event simulations. Specifically, it contrasts typical discrete-event simulation (DES) execution algorithms with how events are selected and ordered by the discrete-event system specification (DEVS) formalism. The motivation for this research stems from a desire to understand how the selection of events affects simulation output (i.e., response). As a particular use case, we briefly investigate the processing of simultaneous events by the Advanced Framework for Simulation, Integration and Modeling (AFSIM), a military discrete-event combat modeling and simulation package. To facilitate the building of classic DEVS-based models, the python software package PythonPDEVS is used. Initial results indicate that the explicit modeling of how simultaneous events are selected as promoted by the DEVS formalism plays a significant role on simulation results.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Douglas Hodson, for his guidance and support throughout the course of this thesis effort. The insight and experience gained through this experience is appreciated. I would, also, like to thank my sponsor for both the support and latitude provided to me in this endeavor.

John M. Carboni

Table of Contents

Page

List of Figures

Page

List of Tables

Page

THE EFFECT OF MODELING SIMULTANEOUS EVENTS ON SIMULATION RESULTS

I. Introduction

The DoD defines a simulation as "A method of implementing a model over time. Also a technique for testing, analysis, or training in which real-world and conceptual systems are reproduced by a model." (Battilega and Grange)

General Issue

The Advanced Framework for Simulation, Integration and Modeling (AFSIM) is a military discrete-event-based combat modeling and simulation package. AFSIM, like other DES packages, sorts scheduled events using a priority queue, but also like other packages, prioritizes the execution of events (scheduled at the same time) in an undetermined manner. In other words, simultaneous events are scheduled for execution based on their order of insertion (as determined by software execution flow). It is believed that this ordering and execution influences simulation output and therefore results. Additionally, AFSIM (like other discrete-event-based simulations) provides no explicit method to model or determine the processing order of scheduled simultaneous events. It is believed that the execution order associated with simultaneous events is an important aspect to any simulation; by knowing how a model behaves when simultaneous events are scheduled allows one to know their experiment's bias and adjust accordingly when analyzing results. Two, or more, models may have events scheduled to occur at the same time but there is always an order to which model executes first in a computer simulation. This research seeks to understand the effect the execution order on simulation results - specifically, it illuminates this issue as a modeling concern.

The Discrete Event System Specification (DEVS) formalism, which lays out well-defined parameters for how to create one's models, model systems, and handle simultaneous events, will be used to evaluate this issue. There are many variations of the DEVS formalism; this research considers the classic formalism. In DEVS the atomic model is the fundamental building block for defining a system behavior. Atomic models are connected to create so-called coupled models which lead to assembling a more complex hierarchy. DEVS defines a "select function" which provides an avenue to explicitly define how simultaneous events are to be processed. This research leverages a python-based DEVS software package called PythonPDEVS, which of course allows a programmer to explicitly define the "select function". Because DEVS, forces a modeler (i.e., programmer) to explicitly define this aspect of behavior, more insight is gained in understanding simulation outputs.

Problem Statement

AFSIM is a popular simulation package that is used to conduct combat-oriented simulation studies. Because of this, and the prevalent use of other discrete-event simulations, an understanding of how the processing of simultaneous events might affect simulation results is of high interest. AFSIM simply provided a motivation for this research; the results should be interpreted more generally.

Research Objectives/Questions

The objective of this research is to design simple simulation examples that explore the effect of modeling of simultaneous events on simulation results. To address the problem statement, this research answers the following questions:

1. Illuminate how AFSIM (in particular) processes simultaneous events?
2. How can the importance of simultaneous event handling be exemplified, and is it important?

Hypothesis

Does the modeling of simultaneous events matter or affect simulation results? If yes, then AFSIM, as well as other DES, could benefit from implementing a method of defining priority in simultaneous events.

Methodology

Once the classic DEVS framework was decided to be the best choice to make the desired examples, a tool implementing the formalism was determined. Upon reviewing the pros and cons of multiple different DEVS-based tools, PythonPDEVS was the tool most applicable to the purposes of this research. As PythonPDEVS is written in Python, it allows fast prototyping of models (as it is implemented in the Python language) and lets the user explicitly define simultaneous event handling as the classic DEVS formalism requires. Simple examples focusing on the effect of the simultaneous event handling were built. These examples were evaluated by observing how the processing of simultaneous events affects simulation results.

Implications

This research will provide decision makers an understanding and examples of how including a method to explicitly define simultaneous event handling can affect simulation results. Also, this thesis will provide straightforward, easily understood examples of the effect of modeling simultaneous events on simulation results.

Preview

Chapter 2 will provide background knowledge for the research in this thesis. Chapter 3 expands the methodology of this research regarding the experiments performed. Chapter 4 will give more detailed results and findings regarding the experiments. Finally, chapter 5 will have conclusions and recommendations for future work.

II. Literature Review

Chapter Overview

The purpose of this chapter is to provide the reader with the requisite background knowledge for this topic. To start, a brief introduction to what Modeling and Simulation, as well as Discrete Event Simulation (DES) is, and why it is useful in a wide range of fields. Next, an overview of the Air Force's AFSIM system and how it has provided the motivation for this research. Then, an explanation of what DEVS (specifically classic DEVS) is, how it works, and why it's being used. Finally, a review of the different DEVS tools and why PythonPDEVS was the one chosen to perform this research.

Modeling and Simulation

Modeling and Simulation is the use of computers to approximate real-life scenarios, it is a substitution for physical experimentation. Using modeling and simulation in lieu of a physical experiment reduces costs and avoids safety issues. A computer simulation of a combat situation is cheaper and obviously safer than a real alternative.

The act of "modeling" is one that produce a model – a representation of a real or imagined system of interest. "A model is a representation of the construction and working of some system of interest. A model is similar to but simpler than the system it represents" (Maria). One's model should walk the line between representing functions of the real system and staying simple enough to experiment with, "a good model is a judicious tradeoff between realism and simplicity" (Maria). It is recommended to gradually increase a model's complexity over time. However, before experimenting one

must validate their model. This means to verify that the model performs as the real system does. “Model validation techniques include simulating the model under known input conditions and comparing model output with system output” (Maria). An invalid model will yield useless experimentation results. Model type classifications include deterministic, stochastic, static, and dynamic. “Typically, simulation models are stochastic and dynamic” (Maria), which means that they have at least one variable (input or output) that is probabilistic and time-varying interactions among variables are a factor.

A simulation is an experiment with a model to test how that model will behave under different circumstances, to do the same in the system represented by the model is usually too expensive or impractical. “Simulation is a tool to evaluate the performance of a system, existing or proposed, under different configurations of interest and over long periods of real time” (Maria). It’s important to be able to test a model without changing the actual system it represents because rash “act first, think later” actions will yield poor results. “Simulation is used before an existing system is altered or a new system built, to reduce the chances of failure to meet specifications, to eliminate unforeseen bottlenecks, to prevent under or over-utilization of resources, and to optimize system performance” (Maria). This research uses a discrete event simulator.

A Discrete Event Simulation (DES) models changes in system state when specific events occur at different times. “Discrete event simulation is less detailed (coarser in its smallest time unit) than continuous simulation but it is much simpler to implement, and hence, is used in a wide variety of situations” (Maria). While the actual processing of the simulation is done by a computer, the most important part of a simulation is the human(s) involved in crafting it. Just because the simulation runs does not mean it’s good or useful,

"human decision making is required at all stages, namely, model development, experiment design, output analysis, conclusion formulation, and making decisions to alter the system under study" (Maria).

Modeling and Simulation is a frugal experimentation option. Often is the case where it is impossible/extremely expensive to test systems in the real world. Mathematical models also lend themselves nicely to simulation when they "can be formulated but analytic solutions are either impossible or too complicated" (Maria). Many different fields use modeling and simulation to their advantage, it is one of the most used research techniques. "Applications of simulation abound in the areas of government, defense, computer and communication systems, manufacturing, transportation, health care, ecology and environment, sociological and behavioral studies, biosciences, epidemiology, services, economics and business analysis" (Maria).

An important part of DES is the method of resolving simultaneous events in one's model. Handling this issue has been done differently in different software, but there hasn't really been a clear example of the effect of explicitly defining model priority in simultaneous events. My thesis provides some examples of this and evaluates how different an outcome can be depending on which event occurs first.

Discrete Event Simulation

In Discrete Event Simulation, the "central assumption is that the system changes instantaneously in response to certain discrete events" (Maria). The only thing that matters in a discrete simulation is the event, nothing between events is of any consequence. Computer networks are an example of discrete event systems. Some events

for this example can be the start of a packet transmission, end of a packet transmission, and expiry of a retransmission timeout. “This implies that between two events such as start of a packet transmission and end of a packet transmission, nothing interesting happens” (Varga). This means that it’s important for the person modeling to include any and all states of interest according to their intentions. Discrete event simulations run on a loop of future events, this loop continues until the simulation ends. “The subsequent loop consumes events from the [event loop] and processes them. Events are processed in strict timestamp order in order to maintain causality, that is, to ensure that no event may have an effect on earlier events.” (Varga). Figure 1 shows pseudocode of a typical future event set (FES).

```
initialize -- this includes building the model and
              inserting initial events to FES

while (FES not empty and simulation not yet complete)
{
    retrieve first event from FES
    t:= timestamp of this event
    process event
    (processing may insert new events in FES or delete existing ones)
}
finish simulation (write statistical results, etc.)
```

Figure 1: Pseudocode of a FES

“Initialization step usually builds the data structures representing the simulation model, calls any user-defined initialization code, and inserts initial events into the FES to ensure that the simulation can start” (Varga). Events are processed in timestamp order, the processing of events is where the code supplied by the modeler comes into play. This is the code that defines what each event means regarding the simulation. “Processing an event involves calls to user-supplied code. For example, using the computer network

simulation example, processing a timeout expire event may consist of re-sending a copy of the network packet, updating the retry count, scheduling another timeout event, and so on" (Varga). A discrete simulation ends when a specific termination condition has been met, "The simulation stops when there are no events left (this happens rarely in practice), or when it isn't necessary for the simulation to run further because the model time or the CPU time has reached a given limit, or because the statistics have reached the desired accuracy" (Varga).

A traffic light is an example of a discrete-event system. A traffic light has three states (red, green, and yellow) and it does nothing interesting between them. Each state of the traffic light elicits a response from its surroundings, but there is no change in response while the state advances in time. When the traffic light transitions into a new state, that's when the response changes. This thesis expands on this example, discretely modeling an intersection with car, policeman, and traffic light models to examine how the simulation's results are affected by explicitly modeling simultaneous events.

Using discrete-event simulation has been very useful in many fields of study. Medical studies have used DES to improve radiation therapy scheduling (Werker et al.) and to examine ways to better clinical environments (Swisher, Jacobson, Jun, et al.). In addition, DES has been used by studies improving optimization principles (Swisher, Jacobson, and Yücesan) as well as assembly manufacturing practices (Detty and Yingling).

AFSIM

AFSIM is a combat modeling simulation framework that leverages DES to process events of interest. “AFSIM was developed to address analysis capability shortcomings in existing legacy simulation environments as well as to provide an environment built with more modern programming paradigms in mind” (Clive et al.). AFSIM was originally called Analytic Framework for Network-Enabled Systems (AFNES) and was developed by Boeing. In February of 2013 Boeing, under contract, delivered AFNES to the Air Force with unlimited government rights. The Air Force Research Lab “rebranded AFNES as AFSIM and has begun to distribute AFSIM within the Air Force and DoD, including DoD contractors” (Clive et al.).

AFSIM is a modeling and simulation tool used by the Air force to simulate the warfighter in various scenarios. It is a government developed C++ framework (framework shown in figure 2) used for “constructing engagement and mission-level analytic simulations for the operations Analysis community, as well as virtual experimentation” (Clive et al.). AFSIM can simulate conditions from sea-level to space and evaluate the efficiency of a military mission in those domains. AFSIM can simulate a myriad of military systems and weapons using its agent-based software. To name a few, it can model “weapon kinematics, sensor systems, electronic warfare systems, communication networks, advanced tracking, correlation, and fusion algorithms, and automated tactics and battle management software” (Clive et al.). AFSIM has three main parts; the framework itself, the Integrated Development Environment (IDE), and the Visualization Environment for Scenario, Preparation and Analysis (VESPA).

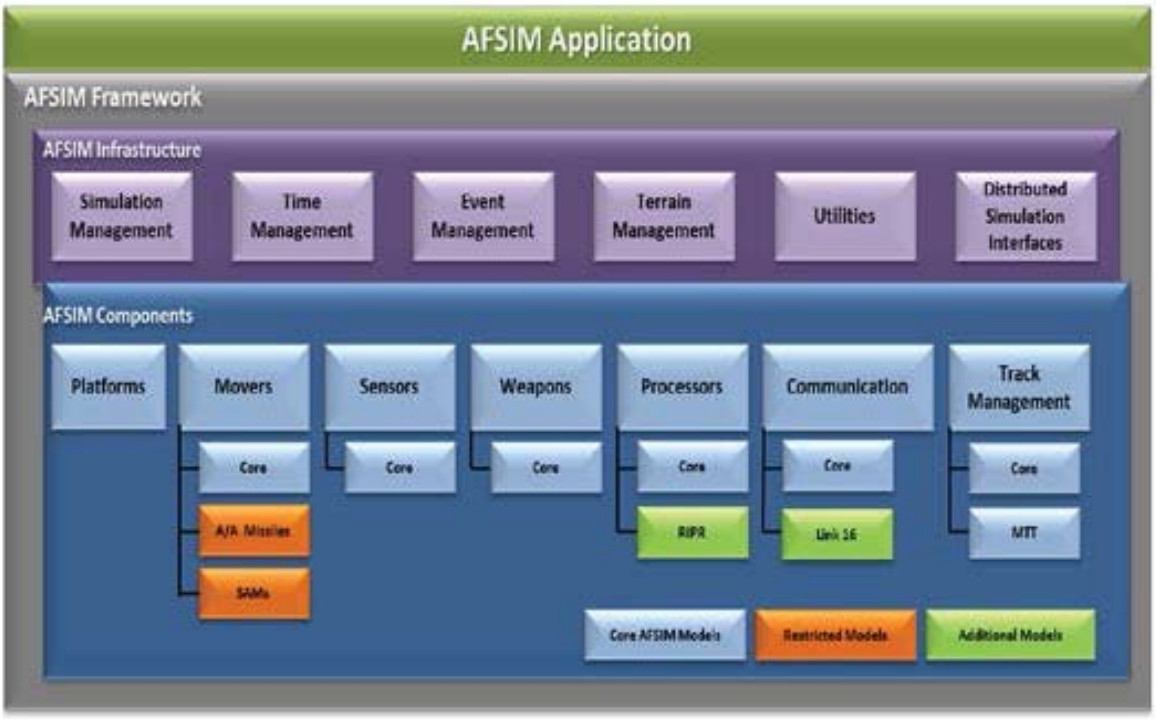

Figure 2: AFSIM Framework

The framework of AFSIM is the actual code libraries that run the simulations. These libraries house routines for:

"the top-level control and management of the simulation; management of time and events within the simulation; management of terrain databases; general purpose math and coordinate transformation utilities; and support of standard simulation interfaces, such as those supporting the Distributed Interactive Simulation (DIS) protocol" (Clive et al.).

The AFSIM software also allows for the creation of entities (models) to populate scenarios, these include "user-defined movers, sensors, weapons, processors for defining system behavior and information flow, communications and track management" (Clive et al.).

A central question of concern in this thesis is how the order of processing simultaneous events affect simulation results. Since AFSIM provided the motivation for this research, a quick investigation of how it handles this conflict was checked. As seen

in Figure 3, there is no other criteria in which events are sorted on other than time. It leverages a priority queue ordered by simulation time.

```
37      //! Add an event to the event queue.
38      //!
39      //! @param aEventPtr Pointer to the event to be added to the queue.
40      //! The WsfEvent::GetTime method will be used to determine when the event should be dispatched.
41      //! The event manager becomes the owner of the event.
42      //!
43      //! If multiple events are queued with the same time, the order in which
44      //! those events are dispatched is not guaranteed to be the same as the
45      //! order of insertion.  The only guarantee is that events with a smaller
46      //! time will be dispatched before those with a greater time.
47      virtual void AddEvent(WsfEvent* aEventPtr);
```

Figure 3: Comment in AFSIM Header File

An undetermined priority governs the handling of simultaneous events in AFSIM. This comment led to the inspiration that motivated this research. It should be noted, it is unknown as to whether AFSIM has simultaneous events happen often or at all. This research utilizes a contrived example in which simultaneous events happen often.

Classic DEVS

DEVS has been around for over forty years and is a widely accepted and applied framework in the modeling and simulation community. The DEVS formalism was developed in the 1970s by Bernard P. Zeigler. “As a mathematical basis for discrete event modeling, DEVS provides not only a formal representation of discrete event dynamic systems that is independent of any computer realization, but also a guideline for how to build abstract DEVS simulation engines to simulate the models” (Song). This research uses the classic DEVS formalism. Classic DEVS is the original version of DEVS, supporting sequential DES modeling and simulation. “Main advantages of DEVS

compared to other discrete event formalisms are its rigorous formal definition, and its support for modular composition" (Van Tendeloo and Vangheluwe, "Classic Devs Modelling and Simulation").

In classic DEVS, the atomic model is "the indivisible building block of a model" (Van Tendeloo and Vangheluwe, "Classic Devs Modelling and Simulation"). The atomic model, whatever it may be modeling, needs to have a multitude of things for it to be complete. The atomic model is described as an 8-tuple:

$$M = \langle X, Y, S, q_{init}, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

- X is the set of input events
- Y is the set of output events
- S is the set of states
- q_{init} is the initial state of the model
- ta is the time advance function
- δ_{ext} is the external transition function
- δ_{int} is the internal transition function
- λ is the output function

An input/output event in DEVS is something that triggers the model to transition from one valid state to another. A transition is what defines the movement between states. Internal transitions are due to the passage of time in the simulation, the time advance. External transitions are caused by an input from another model. "When there is no external event, the time interval the model stays on its current state is determined by

applying the ta function to the current state. And the next state of the model is determined by $\delta_{int}(s)$, where s is the current state" (Song). Only internal transitions trigger a model's output function,

> "the output function is defined on the state, and deterministically returns an event (or no event). the event is generated before the new state is reached. This means that instead of the new state, the output function still uses the old state (i.e., the one that is being left). For this reason, the output function needs to be invoked right before the internal transition function" (Van Tendeloo and Vangheluwe, "Classic Devs Modelling and Simulation").

Multiple atomic models can be coupled together to create a coupled model. Atomic models are coupled together via their input/output ports (shown below as C_{xx}, C_{yx}, C_{yy}). A coupled model is also described as a tuple:

$$M = \langle X, Y, D, \{M_i\}, C_{xx}, C_{yx}, C_{yy,} Select \rangle$$

- X, Y are input/output event sets
- D is the component model name set
- $\{M_i\}$ is the set of component models
- C_{xx}, C_{yx}, C_{yy} are the input, internal, output couplings
- $Select$ is a tie-breaker function

Coupled models can also be treated as atomic models to create more complex scenarios, this is called closure under coupling. "Coupled models are not distinguishable from atomic models when they are coupled with atomic models. Based on the feature of closure under coupling of DEVS, complex system can be hierarchically constructed by model coupling" (Song). The select function in a coupled model is what is of most

interested for this research. This is what determines priority in a simultaneous event, two or more events occurring at the same time. “This function takes all conflicting models and returns the one that gets priority over the others. After the execution of that internal transition, and possibly the external transitions that it caused elsewhere, it might be that the set of imminent models has changed” (Van Tendeloo and Vangheluwe, “Classic Devs Modelling and Simulation”).

Classic DEVS works well for this thesis because of its very strict formalism. If simultaneous events can affect the outcome of a simulation in classic DEVS, where every aspect of the atomic and coupled models have such strict requirements to adhere to in order to ensure validity, then it should affect results of simulations in other frameworks. Classic DEVS can be implemented via various software tools, each have their share of pros and cons.

PythonPDEVS

There are many different software tools that implement the DEVS formalisms: ADEVS, CD++, DEVS-Suite, MS4 Me, PowerDEVS, PythonPDEVS, VLE, and X-S-Y. Each tool supports specific formalisms and have their own features, “these tools have distinct design goals and a specific programming language implementation” (Van Tendeloo and Vangheluwe, “An Evaluation of DEVS Simulation Tools”). Three of these tools were more suited to this research than the rest; ADEVS, CD++, and PythonPDEVS. Using ADEVS wasn’t an option because it does not have the required select function for simultaneous events, which this thesis uses (as shown in Figure 4). CD++ is a very popular tool and it supports classic DEVS, but not as well as required. CD++ does not

allow for the user to define their own select function, "CD++ and X-S-Y implicitly use a hard-coded select function, such as selecting the first model after alphabetic sorting on model name" (Van Tendeloo and Vangheluwe, "An Evaluation of DEVS Simulation Tools").

		ADEVS	*CD++*	*DEVS-Suite*	*MS4 Me*	*PowerDEVS*	*PythonPDEVS*	*VLE*	*X-S-Y*
Formalisms	Parallel DEVS	Y	M	Y	Y	N	Y	Y	N
	Classic DEVS	N	Y	N	N	Y	Y	N	Y
	Dynamic Structure	Y	M	N	N	N	Y	Y	N
	Cell DEVS	N	Y	N	N	N	N	Y	N
Compliance	Translation functions	M	N	N	N	N	Y	N	N
	Event modularity	N	M	N	N	N	M	N	N
	Positive time	Y	N	N	M	N	Y	Y	N
	Select function	-	N	-	-	M	Y	-	N
	Confluent	Y	-	Y	N	-	Y	Y	-

Figure 4: DEVS Sim Tools Comparison (Van Tendeloo and Vangheluwe, "An Evaluation of DEVS Simulation Tools")

Of the tools supporting classic DEVS, "only PythonPDEVS allows users to define the select function explicitly" (Van Tendeloo and Vangheluwe, "An Evaluation of DEVS Simulation Tools"). PythonPDEVS supports the functionality required for this thesis while also providing much documentation, good examples, and accessibility to its creators. It is also, arguably, quicker to learn and use since it utilizes the Python programming language. "Due to its implementation in Python, an interpreted, dynamically typed language, fast prototyping of models becomes possible. Despite its interpretation-based nature, PythonPDEVS attempts to achieve high performance. Both

atomic and coupled models are written in Python, making (re)compilation unnecessary" (Van Tendeloo and Vangheluwe, "An Evaluation of DEVS Simulation Tools"). PythonPDEVS will be the tool used by this research to create examples that test the effect of modeling simultaneous events on simulation results.

Summary

Modeling and Simulation is the use of computers to approximate real-life scenarios. This is typically the preferred method of experimenting due to the low cost as opposed to live experimentation. Modeling and simulation are used in many fields, the military, especially, is no stranger to it.

The structure of a discrete-event simulation execution algorithm motivated interest in DEVS, a discrete event formalism that explicitly forces a modeler to define all aspects of a "model". There are many DEVS formalisms, but for the purposes of this research, classic DEVS was chosen as it defines this aspect for a model. Finally, the PythonPDEVS tool was selected to implement the models.

III. Methodology

Chapter Overview

The purpose of this chapter is to present this research's process into how investigative questions will be answered. First, the objective of our DEVS model examples are highlighted. Next, system boundaries and various variables (response, control, constant factors) are identified. Then, the actual test examples will be explained. Finally, a description of the testing matrix will be presented.

Experiment Objective

The experiment objective is to understand how the modeling of simultaneous events can affect the results of one's simulation. The experiment will focus on simple examples modeling a traffic system. The first scenario is a rudimentary model containing one observer and two traffic light models. Which light will the observer see change first? The purpose being to provide a very straightforward example of the select function at work. The second scenario is slightly more complex. Models for a traffic light, policeman, and car will interact in expected ways as in real life, but the important part is how they interact at that moment when the light changes from yellow to red and the car crosses the intersection at the same time. Does the car get ticketed by the policeman or succeed in crossing the intersection? How the select function is defined for this scenario will directly impact the results of this experiment.

System Boundaries

There are seven different ways to define the select function for scheduling priority to take place: Policeman (P)-Light (L)-Car (C), P-C-L, L-P-C, L-C-P, C-L-P, C-P-L, and

a random draw. Expectations are for the variants giving priority to the policeman to be modeling a strictly enforced intersection, one with a cop diligently watching and/or a traffic light camera. This would represent the high end of the spectrum, many calls to the select function and multiple tickets given. Giving scheduling priority to the car would represent the low end of the spectrum, modeling an intersection that is much less strict. The random draw and the light would represent the middle of the spectrum and probably be closer to the average intersection.

The main purpose of this experiment is observing the effect of modeling simultaneous events on a simulation's results. Even if these predictions are incorrect, expectations are still that each priority scheme will provide significantly different results from the last.

Assumptions

The following are assumptions made for this experiment:

- Experiment utilizes the classic DEVS formalism and the PythonPDEVS DEVS tool
- Experiment is performed on windows machine, operating on Windows 10, with Python 3.6 installed

Response Variables

Response variables will be the number of "ticket" state occurrences, times the select function is called during the simulation, and the number of models' states occurrence:

- Number of "ticket" state occurrence: The total amount of times a ticket was issued from a simultaneous event is the primary response variable. This can be affected by model transition times, the total simulation run time, and the select function.
- Number of select function calls: This will be the total amount of times the simulation has a simultaneous event occur and calls the select function to resolve it. This can be affected by model internal transition times, the total simulation run time, and the definition of the select function.
- Number of Models' states occurrence: This will be the total number of times a model's state occurred in the simulation, for all models' states. This can be affected by model internal transition times, the total simulation run time, and the definition of the select function.

Control Variables

The control variable will be the definition of the select function, the seven different definitions for this will provide different results:

- Select function definition: The experiment will use seven different select function definitions to test how the modeling of simultaneous events can affect the results of a simulation

Constant Factors

The following are all the constraints I am not going to change throughout the experiment's different phases:

- Simulation run time: The simulation will be the same amount of time at every part of the experiment.
- Model transition times: The internal transition times of the car, traffic light, and policeman models will remain constant. Changing these wouldn't serve the purpose of this experiment other than to unnecessarily complicate it.
- Number of models: Adding more models to this simulation would bring unnecessary difficulty for this experiment.
- Number of Trials: Each phase of the experiment will be the same amount of trials.

Example Models

The first model made is a very simple contrived example of the select function at work, there are two traffic light models changing from red to green to yellow in a cycle at the same internal transition times. There is also an observer model that just looks at the lights and outputs which light it sees change first. This allows for a very easy defining of the select function and understanding of its effect on the simulation. Defining the select function to prioritize Light_1's internal transition over Light_2's means the external transition for the observer will always notice Light_1 first, and vice versa for prioritizing Light_2. We can already see, even in this simple example, the kind of impact the select function has on the simulation's results. Expanding this example could have the observer behaving differently for seeing Light_2 first as opposed to Light_1.

The second contrived example is slightly more complex with three models. There is a traffic light, a car driving through the intersection, and a policeman watching the intersection. All internal transition times match up with that of the traffic light. The traffic

light initializes in the “red” state for 60 seconds, transitions to its “green” state for 50 seconds, transitions to its “yellow” state for 10 seconds, and the cycle repeats. The car initializes in the “stopped” state for 60 seconds and then has the internal transition to the “continue_through” state for 60 seconds. The policeman initializes in the “alert” state for 60 seconds and then transitions to its “idle” state for 60 seconds. The policeman also has a one second “give_ticket” state that it can externally transition to if it is already in its “alert” state, the traffic light is in its “red” state, and the care is in its “continue_through” state. Every transition is based off the traffic light to focus on the changes that result with the differing select function definitions. At every transition three models are trying to execute at the same time, results of this model depend even more on the definition of the select function. If there is no defined select function then the priority of simultaneous events is based on the alphabetical order of model names (i.e “car”, “policeman”, then “trafficLight”). The testing process is explained in the next section.

Test Matrix

Testing will be done based on the different ways to define the select function. The random definition testing will consist of 10000 trials of 1000 second simulations to collect a large amount of data. My experiment is done using a computer, 10000 trials should be easy enough to get and this will also let the data points show a sharper difference between definitions of the select function. All the other defined select function tests are deterministic and need only be ran once to gather enough data. The table below illustrates the data collection:

Table 1: Test Matrix for Select Function Calls

Select Definition	P-L-C	P-C-L	L-P-C	L-C-P	C-L-P	C-P-L	Random (10,000 Trials)
Calls to Select Function	# Select Calls	# Select Calls	# Select Calls	# Select Calls	# Select Calls	# Select Calls	# Select Calls (AVG)

Each trial is one run of the simulation, the random definition will run the simulation 10000 times. I expect the number of times the select function is called to be predictable; policeman phases will have the most, car phases will have the least, and light/random phases will be somewhere in the middle. The non-random select function definitions are deterministic and will only be run once. Table 2 is for the models' states of the simulation for each select function definition:

Table 2: Test Matrix for Model State Occurrences

(Select Definition)	Alert (AVG for random trials)	Idle (AVG for random trials)	Red (AVG for random trials)	Green (AVG for random trials)	Yellow (AVG for random trials)	Continue (AVG for random trials)	Stopped (AVG for random trials)	Ticket (AVG for random trials)
State occurrences	#	#	#	#	#	#	#	#

Table 2 will display the number of times each state occurs in each definition of the select function (average number for random definition). This data will help further determine the effect explicit model priority has on simulation results.

Summary

This Chapter identified the objective, variables, and testing process of this thesis's experiment. A description of the different model examples used was also provided for better understanding.

IV. Analysis and Results

Chapter Overview

This chapter will remind the reader of this research's investigative questions and how they were answered. Then, the results from testing the examples described in chapter 3 will be discussed. The simple two-traffic-light example will be briefly reviewed before going into the more complex example with the traffic light, car, and policeman. Finally, summary statistics and the testing matrices will be used to explain results.

Investigative Questions Answered

At the beginning of this research, the first question was: how does AFSIM handle simultaneous events? This was answered by reviewing the source code for this software.

The second question: How can the importance of simultaneous event handling be exemplified, and is it important? DEVS, specifically classic DEVS with the PythonPDEVS tool, was used to construct simple discrete-event simulation examples focusing on what happens when different models' events conflict during the simulation. It was found that how one explicitly defines simultaneous events, via the select function in this case, does affect the outcome of a simulation. The results of testing these examples are detailed next.

Results of Simulation Scenarios

The primary focus of this section will be on the testing results of the three-model example, it is seen here how big a factor explicit model priority can be in a simulation's results. The example with two traffic lights and one observer was just as straightforward as one would think, the light with priority was always seen first by the observer. It serves

the purpose to provide a simple example of how the select function works and what it does.

Test Matrix Results

Table 3: Average Select Function Calls

Select Definition	P-L-C	P-C-L	L-P-C	L-C-P	C-L-P	C-P-L	Random
Trials (1-10000)	48	48	24	24	40	40	37.3296

Table 4: Random Phase Statistics

Max	45
Min	29
Mean	37.3296
Standard Deviation	2.29507382
Variance	5.26736384

Table 5: Average Model State Occurrences

(Select Definition)	Alert	Idle	Red	Green	Yellow	Continue	Stopped	Ticket
P-L-C	**33**	**24**	**9**	**8**	**8**	**40**	**17**	**0**
P-C-L	**41**	**32**	**9**	**8**	**8**	**40**	**33**	**0**
L-P-C	**17**	**16**	**9**	**8**	**8**	**24**	**9**	**0**
L-C-P	**17**	**16**	**9**	**8**	**8**	**24**	**9**	**0**
C-L-P	**9**	**32**	**9**	**8**	**8**	**24**	**25**	**8**
C-P-L	**25**	**32**	**9**	**8**	**8**	**24**	**41**	**8**
Random	**23.6687**	**25.2813**	**9**	**8**	**8**	**29.354**	**22.2778**	**2.6818**

Table 6: Model State Occurrence Statistics for Random Phase

Random	give_ticket	stopped	yellow	red	idle	green	continue_through	alert
mean	2.6818	22.2778	8	9	25.2813	8	29.354	23.6687
max	8	35	8	9	37	8	40	34
min	0	11	8	9	16	8	20	13
stdev	1.330544535	3.215467	0	0	2.857826	0	2.754066811	2.887653
var	1.77034876	10.33923	0	0	8.16717	0	7.584884	8.33854

Results Overview

The predictions were almost correct. The expected results for the car and the light were switched, but the random and policeman results were as expected. Table 3 displays the number of calls to the select function for each explicitly defined and the average

number of calls for the random definition. We can see that results differed significantly depending on which model had highest priority, but the second highest priority doesn't seem to have much of an effect on the results here at the outset of analysis. Perhaps most interesting are the results for the random definition and how it compares to the other, explicitly defined, phases.

Highest Priority: Policeman

When the policeman has the highest priority, it results in more simultaneous events (48 select function calls) than any of the other definitions tested. This makes sense with how our models were defined, the policeman model's internal transition doesn't cause external transitions in the other models. The policeman switches states from "alert" to "idle", and vice versa, acting as an observer. Due to this, the simulation must play out the next simultaneous event between the other two models every single time.

The second highest priority in these phases cause a slight difference in the actual trace results of the simulation. If the traffic light model has the second highest priority then the simulation results in a smooth intersection, no tickets given. However, if instead the car model is given the second highest priority then the resulting trace contains more data, but still no tickets given. With highest priority, the policeman model changes state to "idle" first before the traffic light model changes state to "green". When the car model has the second highest priority it can run the red light every time because its state changes to "continue_through" before the traffic light transitions to "green". There is more data in the trace because the traffic light model has the most influence over the other two models. Every state change in the traffic light causes external transitions in the other two models,

so when it finally changes state with its last priority it causes external transitions that output redundant data. It can be seen in Table 5 the increase in state occurrences between the P-L-C and P-C-L scenarios.

Highest Priority: Traffic Light

The traffic light resulted in the fewest simultaneous events when it had highest priority, only 24 calls to the select function. This is because the traffic light model is the main influencer in this simulation. Every time the traffic light changes state it will cause an external transition in the other two models and eliminate their next internal transition. This also eliminates the next simultaneous event that would have happened between the car and policeman models, the light has already dictated their next state. With the priority of the light diffusing any further scheduling conflicts, it follows that the select function would be utilized less.

These scenarios are the only ones where the second highest priority doesn't matter. As mentioned earlier, the traffic light model transitions first every time and dictates the states of the other two models. That leaves no further priority required, the other models will only act in external transitions via the traffic light. We see this further exemplified in the trace results for both the L-P-C and L-C-P scenarios, Table 5 shows identical results for state occurrences.

Highest Priority: Car

In the explicit definitions with highest priority given to the car model, the select function was called 40 times. This is the closest average to that of the random scenario. The car model, like the policeman model, doesn't have much influence. The only external

transition the car model causes is in the policeman model. When the car is in state "continue_through" and the traffic light is in state "red", the policeman transitions to a "give_ticket" state. Giving the car priority results in the most tickets issued. The policeman is still in its "alert" state and the traffic light is still in its "red" state when the car changes state from "stopped" to "continue_through". The policeman model's external transition occurs when this happens and gives the car a ticket. Like in the traffic light priority definitions, this external event will eliminate some subsequent simultaneous events from occurring (hence 40 calls) but not all.

The C-P-L and C-L-P scenarios are the only ones that yield any tickets given by the policeman model. The trace results for both definitions show eight tickets given over the course of the 1000 second simulation. Much like when the policeman had highest priority, there is only a slight difference caused by the second highest priority. The C-P-L trace has more data due to the same reason as the P-C-L trace. When the traffic light model has last priority, it will cause external transitions in the other models that, at that point, are redundant.

Highest Priority: Random

For 10000 trials of the 1000 second simulation with random priority, results yielded an average of about 37.3 calls of the select function with a range from 29 to 45 calls. The random priority never reached as high as 48 calls nor as low as 24 calls like we saw on the explicit definitions. Giving priority at random each time a simultaneous event occurs provides a wider range of possibilities in the simulation, but we can see it doesn't account for the scenarios that we explicitly defined. I also noticed that the average of the

other three highest priorities (48, 24, 40) also comes out to 37.3 calls. As expected, the random priority results came out to be the average of the other results.

The difference in traces for the random priority show a range of values for each state that excludes some values from the explicit definitions. For example, in Table 5 we can see the "stopped" state has occurrences as high as 41 and as low as 9, but in Table 6 we see the random results for "stopped" range from 11 to 35. Only the "red", "green", and "yellow" states occur at a constant rate throughout the random trials. These results aren't surprising, no other model has any influence over the traffic light model's transitions. Traffic light states will always occur no matter the priority. Unlike the traffic light model, the policeman and car models' states show much greater variation. The average number of occurrences for their states for the random definition falls in the middle range of the explicit scenario data, except for the "give_ticket" state. The random average for "give_ticket" is 2.6818 with a range from 0 to 8. This is the only state with an average that doesn't land somewhere near the middle of the range. However, 2.6818 is very close to a third of eight and tickets are only given when the car model has priority. Having the average number of occurrences for this state land around the one third mark in the range makes sense since only one of three models contributes to the occurrences of this state. I don't think that would have been too obvious if we didn't have the data from the other explicit definitions.

Analysis

Just as predicted, the simulation varied significantly depending on how the select function was defined. If neglected, the handling of simultaneous events could cause

issues in one's simulation. If left to random, or some arbitrary definition, data could be lost. We look at how the random priority never accounts, as far as number of calls, for the P-L-C/P-C-L and L-P-C/L-C-P scenarios. At a glance, it can be construed as outliers representing unlikely or obscure scenarios, but those scenarios make sense for priority. Simulations are meant to mimic real-world systems, in an actual traffic light intersection it is the traffic light that dictates all the actions of those around it. Also, there are intersections where police officers sit at and/or direct traffic, they would have highest priority in those real-world systems.

We also see redundant data in these results. With this simple three-model example, the second highest priority didn't matter. If the trace wasn't identical then it was providing extra, redundant information. This is another reason why the handling of simultaneous events should be accounted for in any simulation. One can easily streamline the testing of their simulation and reduce noise in their data with proper thought and utilization of the handling of simultaneous events.

If it is known how the simulation handles simultaneous events, then there can be better analysis of why it is behaving a certain way. If said handling can be explicitly defined in the simulation, then there is more well-rounded gathering of data. Knowledge and handling of simultaneous events in one's simulation can be very impactful when trouble shooting, analyzing, and gathering data.

Summary

This chapter covered the results from testing the three-model example. A discussion was had regarding what the results mean and how they relate the bigger

question of this thesis: how important is the handling of simultaneous events? Separate testing scenarios were covered in depth, all leading to the final conclusions drawn in the next chapter.

V. Conclusions and Recommendations

Chapter Overview

This chapter will discuss the conclusions drawn from this research, whether the handling of simultaneous events is important. Then, Recommendations for actions to be taken by DES developers. Finally, Recommendations for future work will end this chapter.

Conclusions of Research

The contrived example in this research has shown that the handling of simultaneous events in modeling and simulation can be an important concept to have properly fleshed out in one's simulation. Depending on the simulation (whether it has multiple simultaneous events), devising priority for models in a simultaneous event can make a difference in one's results. Proper consideration of this concept of modeling and simulation can be impactful when trouble shooting ones models, gathering data, and analyzing data.

Coding the models and simulation conditions is no easy task. Often is the case that the simulation isn't acting as intended and one must figure out why and fix it. Just the knowledge alone of how the simulation handles simultaneous events is a boon to trouble shooting. It makes it easier to walk through what is happening in the simulation and pin point the spot errors begin to occur.

Testing a simulation can take an extraordinary amount of computing time, especially if one is testing for every possible scenario. Accounting for simultaneous

events by explicitly defining model priority can shorten testing time and reduce noise in results by eliminating the gathering of redundant/useless data.

There is also a benefit to the analysis of data when simultaneous event handling is properly accounted for. It adds an extra layer in results analysis between "what happened?" and "what does it mean?". Now, there is also "why it happened that way?" which adds extra insight to the other layers for a more complete analysis.

We know this is true for DEVS, as it was the DES used in this research, but it would be interesting to repeat this experiment with other DES and see if the results are similar.

Recommendations for Action

Motivation for this research came from studying how AFSIM works. The question of the importance of simultaneous event handling stemmed from AFSIM's own undetermined event priority in simultaneous event handling. Due to the findings in my research, I would recommend further investigation to more fully understand its influence on simulation results. If there is some way to update the code framework with something like the select function in DEVS, then it should be considered by decision makers.

Recommendations for Future Research

A great continuation of this research would come in the form of a similar experiment, but in Parallel DEVS using the PythonPDEVS tool. Like the select function in classic DEVS, Parallel DEVS has the confluent function. The Confluent function basically serves the same purpose as the select function, but it is not restricted to internal transition priority between models. It also determines priority for external transitions. It

would be interesting to replicate this research using Parallel DEVS and see how results change. (Van Tendeloo and Vangheluwe, “Introduction to Parallel DEVS Modelling and Simulation ”)

Other good future work would be to replicate this research using other DES tools. It would be interesting to see if other DES tools behaved similarly according to their respective versions of the select function, if they even have one. A survey paper of other DES tools regarding which have implicit handling of simultaneous events, which let you explicitly define model priority, and which neglect it altogether would be good future work. An investigation of AFSIM to see if the explicit modeling of simultaneous events makes a difference (given what it models) would be interesting future work as well.

Summary

This chapter discussed conclusions drawn from the research. Then, recommendations for action were given based on the conclusions. Finally, ideas for future work regarding this thesis were offered.

Bibliography

Clive, Peter D., et al. “Advanced Framework for Simulation, Integration and Modeling (AFSIM).” *International Conference of Scientific Computing*, 2015, pp. 73–77, http://worldcomp-proceedings.com/proc/p2015/CSC7058.pdf.

Detty, Richard B., and Jon C. Yingling. “Quantifying Benefits of Conversion to Lean Manufacturing with Discrete Event Simulation: A Case Study.” *International Journal of Production Research*, 2000, doi:10.1080/00221686.2010.491688.

Maria, Anu. “Introduction to Modelling and Simulation.” *Winter Simulation Conference*, 1997, doi:10.1109/WSC.1997.640371.

Song, Hongyan. “Infrastructure for DEVS Modelling and Experimentation.” *McGill Master Thesis*, School of Computer Science McGill University, Montr´eal, Canada, 2006.

Swisher, James R., Sheldon H. Jacobson, and Enver Yücesan. “Discrete-Event Simulation Optimization Using Ranking, Selection, and Multiple Comparison Procedures.” *ACM Transactions on Modeling and Computer Simulation*, 2003, doi:10.1145/858481.858484.

Swisher, James R., Sheldon H. Jacobson, J. Brian Jun, et al. “Modeling and Analyzing a Physician Clinic Environment Using Discrete-Event (Visual) Simulation.” *Computers and Operations Research*, 2000, doi:10.1016/S0305-0548(99)00093-3.

Van Tendeloo, Yentl, and Hans Vangheluwe. “An Evaluation of DEVS Simulation Tools.” *Simulation*, vol. 93, no. 2, 2017, pp. 103–21, doi:10.1177/0037549716678330.

---. “Classic Devs Modelling and Simulation.” *2017 Winter Simulation Conference*, 2017.

---. “Introduction to Parallel DEVS Modelling and Simulation .” *Proceedings of the Spring Simulation Conference 2018, April 15-18, 2018, Baltimore, VS*, 2018, pp. 613–24, doi:10.1002/bltj.

Varga, Andras. “Discrete Event Simulation System.” *OMNeT++ User Manual*, Version 3., 2005.

Werker, Greg, et al. “The Use of Discrete-Event Simulation Modelling to Improve Radiation Therapy Planning Processes.” *Radiotherapy and Oncology*, 2009, doi:10.1016/j.radonc.2009.03.012.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
21-03-2019	Master's Thesis	Sept 2017 – March 2019

TITLE AND SUBTITLE

The Effect of Modeling Simultaneous Events on Simulation Results

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Carboni, John M., 2nd LT, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-19-M-014

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Headquarters US Air Force
1570 Air Force Pentagon, Washington, DC 20330-1570
571-256-2011, Mark.A.Gallagher16.civ@mail.mil
ATTN: Dr. Mark A. Gallagher

10. SPONSOR/MONITOR'S ACRONYM(S)

USAF HQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This thesis explores the method that governs the prioritizing process for simultaneous events in relation to simulation results for discrete-event simulations. Specifically, it contrasts typical discrete-event simulation (DES) execution algorithms with how events are selected and ordered by the discrete-event system specification (DEVS) formalism. The motivation for this research stems from a desire to understand how the selection of events affects simulation output (or response). As a particular use case, we briefly investigate the processing of simultaneous events by the Advanced Framework for Simulation, Integration and Modeling (AFSIM), a military discrete-event combat modeling and simulation package.

15. SUBJECT TERMS
DEVS, PythonPDEVS, AFSIM, Simultaneous Event, DES

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Douglas Hodson, AFIT/ENG
U	U	U	UU	49	19b. TELEPHONE NUMBER *(Include area code)* 786-371-3350 john.carboni@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18