Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

THE IMPACT OF MODELING PARADIGMS ON THE OUTCOME OF SIMULATION STUDIES: AN EXPERIMENTAL CASE STUDY

Saikou Y. Diallo
Christopher J. Lynch
Jose J. Padilla
Ross Gore

Virginia Modeling, Analysis and Simulation Center
1030 University Blvd.
Suffolk, VA 23435, USA

ABSTRACT

We explore the impact of using different modeling paradigms on the outcome of simulation studies. Modeling paradigms, once implemented, follow different computational rules regarding how calculations are made and are sequenced during runtime. To test the effects of computational differences on a simulation's outcomes, we implement a simple queuing system as a Discrete Event Simulation model, a System Dynamics model, an Agent-based Model, and a Multi-paradigm Model. Our findings show that paradigm selection does play a role in the generation of outcomes, as the System Dynamics model produces a significantly different set of outcomes than the other models for the selected output variables. This paper serves as a first step in examining how the selection of a paradigm affects the outcome of the simulation.

1 INTRODUCTION

This paper explores differences in simulation results when designing a model using different modeling paradigms and then implementing a simulation using different modeling methods. We use a scenario comprising a conceptual model of a generic restaurant during its five hour lunch period to design and implement four simulations using the modeling paradigms of Discrete Event Simulation (DES), System Dynamics (SD), Agent-based Modeling (ABM), and Multi-paradigm Modeling (MPM). We implement the first three models using DES, SD, and ABM individually and we construct the MPM using a combination of all three of these simulation methods simultaneously. The purpose of this paper is to show that the selection of a modeling method for constructing a simulation can have a significant impact on the calculation of the simulation's results; thereby showing the importance of selecting appropriate modeling methods based on the description of the system. We aim to support the foundation of MPM research by examining how different implementations of the same conceptual model yield different results.

We present four simulation models constructed using different modeling paradigms. Modeling paradigms "encompass various ways to think about representing systems" (Lynch and Diallo 2015, pg. 1622), serve as the "mindset for modeling" that drives model design (Hardebolle and Boulanger 2009, pg. 689), and contains the assumptions that generally accompany each way of thinking (Lorenz and Jost 2006). From a more general perspective, the term 'paradigm' reflects the commonly shared set of "assumptions, concepts, values, and practices that constitutes a way of viewing reality" within a community (McGregor and Murnane 2010, pg. 1) and "sets down the intent, motivation and expectations" that drive the research (Mackenzie and Knipe 2006, pg. 2). A paradigm does not require a standard interpretation or a full set of rules in order to guide research (Kuhn 1970).

In contrast, 'Modeling methods' link modeling paradigms with their implementations. A method is the technique, procedure, or tool used to collect data, conduct research, or analyze data and is based upon a selected methodology (Mackenzie and Knipe 2006; McGregor and Murnane 2010). A methodology is an approach linking a paradigm to research (Mackenzie and Knipe 2006) and deals with the "philosophical assumptions that underlie any natural, social or human science" (McGregor and Murnane 2010 pg. 2). A common approach for transitioning from a MPM model or any single modeling paradigm into an executable simulation is through the use of formalisms (Vangheluwe, de Laura, and Mosterman 2002; Zeigler 1984). Formalisms provide explicit, unambiguous representations of a model necessary for a model's implementation on a digital computer (Zeigler 1984). The Discrete Event System Specification (DEVS) formalism, developed by Zeigler (1984), is an example of a commonly used formalism for implementing DES models.

Multi-paradigm Modeling involves using multiple paradigms to facilitate the conceptualization and design of a model (Lynch et al. 2014; Villa and Costanza 2000), but uses one or more modeling methods to implement the simulation (Hardebolle and Boulanger 2009; Mosterman and Vangheluwe 2004; Vangheluwe, de Laura, and Mosterman 2002). Vangheluwe, de Lara, and Mosterman (2002) define MPM as a combination of multi-formalism modeling (the coupling together of models specified in different formalisms or converting a model specified in one formalism into a different formalism), abstraction (the relationships between models constructed at different levels of abstraction), and meta-modeling (describing classes of models which allows for the specification of formalisms). From this definition, abstraction plays the role of connecting paradigms together when transitioning to implementation, as any mechanisms needed within the combined model for maintaining consistency of units, time, and interactions should be identified as part of this process. Collectively, these concepts emphasize the importance of being able to build the simulation when engaging in MPM. Lynch et al. (2014) present a framework for constructing multi-paradigm models that is intended to handle multiple stakeholders' perspectives, account for different levels of abstraction, and provide traceability from reference model, to conceptual model, to simulation model. Hardebolle and Boulanger (2009) provide a survey of techniques used for conducting MPM.

Other Modeling and Simulation (M&S) terms contain similar meanings to that of MPM, such as multimodeling as a more general term referring to models comprised of multiple models independent of paradigm (Fishwick and Zeigler 1992; Fishwick 1995) or multifaceted modeling when using multiple models to address different portions of a question (Zeigler 1984). Lynch and Diallo (2015) and Balaban, Hester, and Diallo (2014) provide further discussions on the various M&S terminologies used with respect to constructing models that are comprised of multiple types of models. In this paper, we focus on paradigms and not methods or formalisms.

The remainder of this paper continues as follows: Section 2 provides our methodology for constructing the four simulation models and for conducting our comparison of results. Section 3 provides a description of a conceptual restaurant scenario and each of the four simulation implementations. Section 4 presents the results from each simulation and a discussion. We conclude in Section 5 and provide avenues for future work.

2 METHODOLOGY

Our methodology consists of three basic steps: (1) establish the background information for the base model used for all four simulation implementations; (2) construct the four simulation implementations; and (3) compare the results of each implementation to discern the effects that computational differences cause when running the same experiments for each simulation. The beginning of Section 3 presents the necessary information about the system that holds across all of the models. Step 3 requires running the simulations for the same length of time and collecting equivalent output types. For example, the number of customers that exit the restaurant is represented in each simulation, but in various forms (i.e. entities, flows, and agents). When we construct the simulations (Step 2), we ensure that the same rates of arrivals to the system, processing time within the system, and base time units are maintained across all four implementations. For comparing each of the simulations, we collect the minimum, maximum, average, and mean confidence values for the total number of individuals that complete their business and exit the system as well as for the total number of individuals that are still inside of the system at the end of each run. Additionally, we collect the total real time that each run takes to execute in order to compare differences in execution lengths across the modeling methods.

Using the DES paradigm, we break the system into a sequence of discrete steps, based on the points within the system that require any length of time for the entity to act, or points at which the entities depend upon the availability of a resource. This sequence is then represented as a sequence of nodes connected together along a mono-directed path that is structurally reflective of the modeled system. Our example uses a single server and a single queue which means that the number of resources directly restricts the maximum number of entities that can move through a process at any given time. For queuing simulations, the times between arrivals are statistically similar to number produced by an exponential distribution using the mean value of the flow rate (McKenzie 2010); therefore, equivalent arrival rates can be used in each implementation.

Within the SD paradigm, the entities are observed based on how they flow into and out of the system as an aggregated population. Entities are no longer discernable as individuals moving through the system; instead, the cumulative outflow from the system reveals the total number of entities that exited during the course of a run. Conceptually, this means that resources do not directly interact with entities in this paradigm. Instead, the effect of interacting with entities and keeping them in the system for some period of time is now captured as a dampener on the outflow rate of the model.

Similar to the DES conceptualization, in the ABM paradigm the system is broken down into a discrete set of sequential states. Instead of capturing these states as a sequence of system nodes, the states are represented as a sequence of states that are internal to each agent; therefore, the ABM implementation does not visually reflect a direct structural representation of the modeled system. The individuals entering the system represent one type of agent (of which a new agent is generated as needed during execution) and the system represents a second type of agent. This gives each individual the same set of behaviors and goals, but allows the system to respond to requests in order to maintain order and prevent conflicts from the arriving agents.

The MPM implementation combines components from all three of these implementations. SD components capture the arrivals of individuals into the system using flow rates; however, these arrival rates are then converted into discrete individuals. ABM components represent the individuals within the system as agents and portray their current internal states as they move through the system. DES components represent the sequence of states that the individuals move through within the system as well as the resources that process the individuals. The interactions that cause the individuals to ultimately complete their goals within the system and then exit the system take place between the agents within the ABM component and the resources within the DES components. The following section presents the conceptual model driving each implementation and provides a description of each model.

3 THE MODELS

We use a scenario of a restaurant during its lunch period as a starting point from which to implement simulation models using DES, SD, and ABM individually, as well as a combined DES, SD, and ABM model. A customer arrives on average every 45 seconds over a five hour lunch period from 10:00 am to 3:00 pm and waits in a single line to be served. The restaurant opens for lunch at 10:00 am and the restaurant initially starts free of customers. The restaurant has a single server that takes orders and gives the customers their food one at a time on a first-come first-served basis. The restaurant does not have a seating area; therefore, customers leave the restaurant as soon as they obtain their food. The server spends an average of 45 seconds serving each customer; however, depending upon the order, the transaction can be completed in as quickly as 25 seconds or take as long as 90 seconds per customer. Figure 1 provides a conceptual model for this scenario. Figure 2 parts A-D provide views of the restaurant scenario implemented in each of the simulation methods.

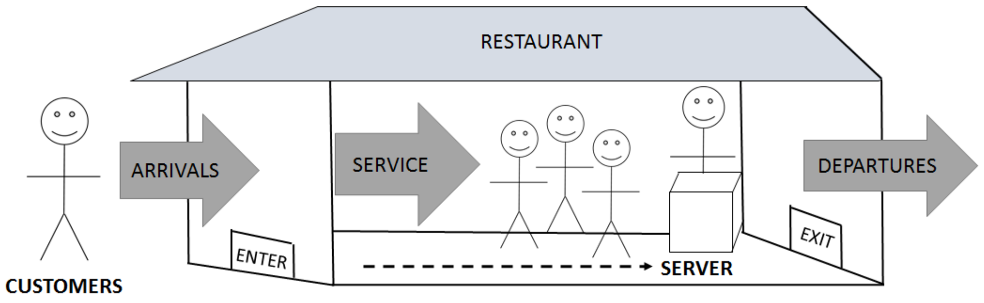

Figure 1: Conceptual model depicting the arrival of customers to a restaurant followed by potential queuing while people receive service one at a time on a first-come first-served bases and concludes with the departure of the customers.

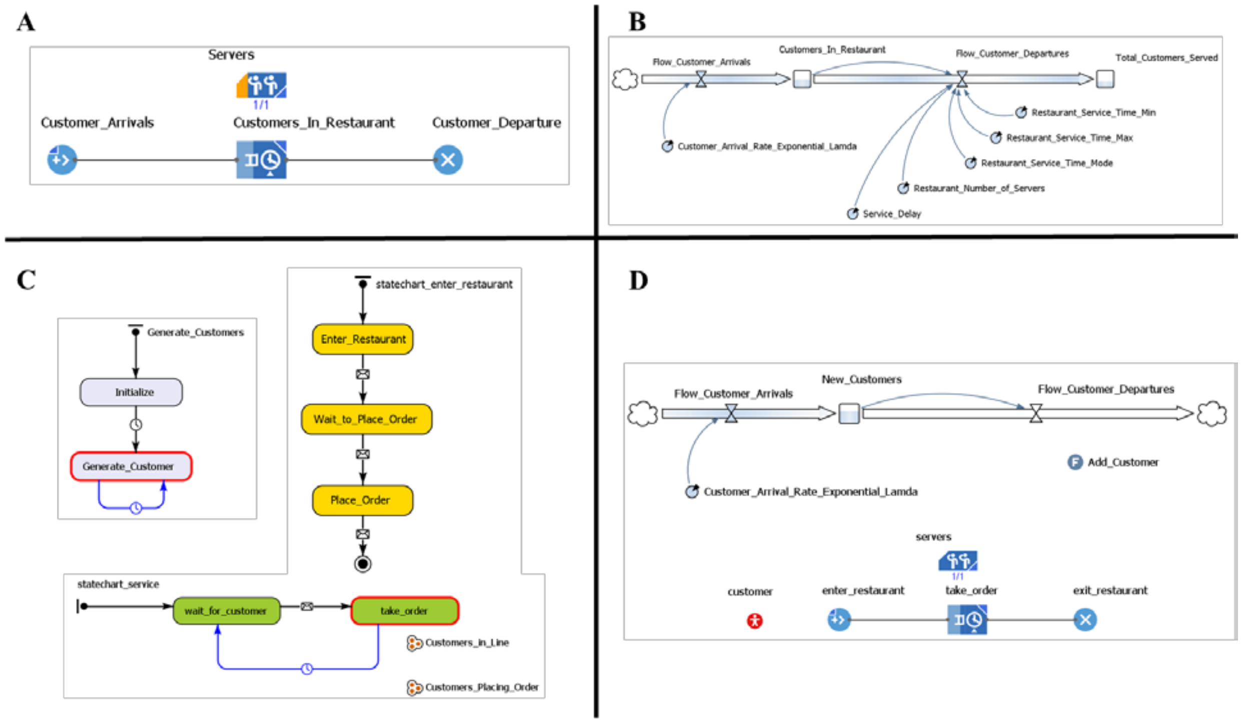

Figure 2: Simulation implementations using (A) Discrete Event Simulation, (B) System Dynamics, (C) Agent-based Modeling, and (D) a multi-method model that combines Discrete Event Simulation, System Dynamics, and Agent-based Modeling.

To help maintain consistency across models, and to form a basis for comparing results, we maintain the following assumptions across each implementation: (1) we use minutes as the base time unit for execution; (2) we use AnyLogic version 7.3.3 for all implementations and executions; (3) an exponential distribution represents customer arrivals; (4) a triangular distribution represents the processing time at the server; (5) there is only one server; and (6) initially there are no customers in the system. The implementations of each paradigm contains numerous differences with respect to time representations, bases of value, behaviors, expressions, and levels of resolution characteristics, leading to computational differences when executing the models. The implementations of the four models are presented within the following four subsections to identify the differences and commonalities between models.

3.1 The Discrete Event Simulation Model

The DES implementation of the restaurant description separates the components of the system into a series of discrete nodes that reflect the possible events that a customer encounters in the restaurant and represents each individual customer as an entity within the system. The simulation consists of one arrival node, one queue, one process node, one resource type, one entity type, and one exit node. Entities are dynamic objects usually created with respect to the expected interarrival times, they move through the system following the links between the nodes, and they are destroyed when they leave the system (Kelton, Sadowski, and Swets 2010). The server is captured as a resource and acts as a required object in order to process an entity through a process node. Representing the server as a resource allows us to restrict the number of customers being processed at any given time within the simulation (in this case one to match the single server specification given for the restaurant), since a resource must be available in order to serve an incoming entity.

The arrival node (*Customer_Arrivals*) captures the average interarrival time between customers using an exponential distribution with an expected value (*E(X)*) equal to a mean value (β) of 0.75 minutes ($E(X) = \beta$) to decide when to generate each new customer during execution. The process node (*Customers_In_Restaurant*) represents the amount of time required for a customer to place an order, pay, and receive the order using a triangular distribution with a minimum value of 0.417 minutes, a maximum value of 1.5 minutes, and with a most likely value of 0.75 minutes. The process node also seizes an available resource to ensure that a customer can only be served when a server is available. The process releases the resource once the customer has been served in order to free up the resource for the next customer. Since only one customer can be served at a time, the queue provides a location for entities to wait in first-come first-served order until the server is available. Immediately after a customer is finished at the process node, that customer moves directly to the exit node (*Customer_Departure*) and is removed from the simulation. The *Customer_Departure* node keeps track of the total number of customers that exit the restaurant.

3.2 The System Dynamics Model

The SD implementation represents the system as a flow of customers through the restaurant (Figure 2B). Unlike the DES and ABM implementations, customers are not differentiated as distinct entities. We use a Stock and Flow diagram to capture (1) the rate of flows over time within the system (represented by thick, directed arrow paths in the figure), and (2) stocks which calculate the number of customers within the restaurant at a given point in time. Flow rates provide the amount of customers that move into or out of (depending on the direction of the arrow) a stock at each time unit. Each flow is internally represented by an equation that includes all variables that are externally connected to it within the diagram. In Figure 2, these variables are portrayed as circles with a wedge inserted in the top-right corner. The stocks are displayed as square boxes and they are mathematically represented using differential equations.

We include the customers' arrival rate (*Flow_Customer_Arrivals*), the rate at which customers are served and thereby leave the restaurant (*Flow_Customer_Departures*), the current number of customers inside of the restaurant (*Customers_In_Restaurant*), and the total number of customers that have been served (*Total_Customers_Served*). Customer arrivals are characterized as an exponential rate with an average of one customer arriving every 45 seconds (*Customer_Arrival_Rate_Exponential_Lambda* = 0.75 minutes) as shown in equation (1) to match the interarrival distribution used in the DES and ABM implementations. However, computationally this exponential function is divided by the time step (dt) used for executing the simulation to provide the change (positive or negative) to the stock since the previous time step in order to produce the rate of change (i.e. the flow) into/out of a stock every time step.

$$Flow\_Customer\_Arrivals = exponential^{(Customer\_Arrival\_Rate\_Exponential\_Lambda)}. \tag{1}$$

Customer departures are a rate defined using a single server (*Restaurant_Number_Of_Servers* = 1) who takes 25 to 90 seconds to serve each customer (customers are reflected by the current value of the *Customers_In_Restaurant* (*CIR*) stock at any point in time during execution) with an average of 45 seconds modified by a delay which represents the time that a customer remains inside the restaurant (possibly queueing) while waiting to be served (*Service_Delay*) as shown in equation (2). Traditionally, SD models do not incorporate randomness within their functions and produce deterministic outcomes; however, recent applications of SD have incorporated stochastic functions in order to account for uncertainty within the modeled system (Djanatliev and German 2013; Langroodi and Amiri 2016; Pretorius and Benade 2011). We maintain the stochastic function of service time for the outflow of customers from the system in order to capture the uncertainty in the outflow rate arising from the variability that human servers bring within a terminating simulation and to account for the variability in total time spent in the system.

$$Flow\_Customer\_Departures = delay\left[min\left(CIR,\ \frac{1}{X}\right), Service\_Delay, Z\right], \tag{2}$$

where CIR ≥ 0 to ensure that the number of customers in the restaurant cannot fall below 0, $Z = 0$ to reflect that the restaurant initially is empty, and $min\left(CIR,\ \frac{1}{X}\right)$ takes the minimum between *CIR* and $\frac{1}{X}$ to bound the flow rate between 0 and the maximum outflow value. *X* equals the number of servers in the restaurant multiplied by a triangular distribution using a minimum of 0.417 minutes (25 seconds), a maximum of 1.5 minutes (90 seconds), and a mode of 0.75 minutes (45 seconds).

The stocks reflect the current number of customers within the restaurant and that have exited the restaurant at time *t* as shown in the differential equations given by (3) and (4). These equations provide the representation of the effects of the flows on each stock (with outflows subtracted from inflows) added to the stock's initial value. Since there is no outflow from the *Total_Customers_Served* (*TCS*) stock, this stock serves as an accumulator for the cumulative number of customers that have traveled through the restaurant at any time *t*.

$$CIR(t) = CIR(t_0) + \int_0^t \left(Flow\_Customer\_Arrivals(t) - Flow\_Customer\_Departures(t)\right) dt, \tag{3}$$
$$TCS(t) = TCS(t_0) + \int_0^t \left(\ Flow\_Customer\_Departures(t)\right) dt, \tag{4}$$

where *CIR*$(t_0) = 0$ and *TCS*$(t_0) = 0$ since the restaurant is initially empty at the start of each lunch period.

As an initial sanity check that the implementation is correct, we check that the time units are consistent across all flows and stocks (Forrester and Senge 1980; Sterman 2000) and that they correctly map to the base time unit of the model.

3.3 The Agent-based Model

The ABM implementation divides the components of the restaurant scenario description into agents that interact in order to capture all of the actions that occur in getting a customer successfully through the restaurant. The model consists of two types of agents: (1) customers (a population of agents); and (2) the restaurant (a single agent). The agents interact by passing messages to each other and acting upon the messages that they receive (Gilbert 2008). Our implementation employs state charts to show the sequence of states that each agent takes during execution and to capture the transitions between states. A state chart displays the current status of an agent at any point in time during execution and transitions between states are unidirectional. The symbol (clock symbol) displayed on the link between each state conveys whether the transition is based on a specific time transition or on receiving a message (envelope symbol). Figure 2C displays three state charts: one to capture the behavior of the customers (*statechart_enter_restaurant*); one to capture the behavior of the restaurant (*statechart_service*); and one to generate customers throughout execution (*Generate_Customers*). The total number of customers processed through the restaurant are tracked within the restaurant.

The *statechart_enter_restaurant* state chart displays the sequence of states and transitions that apply to the customer. Once generated, the customer enters the *Enter_Restaurant* state which sends a message to the restaurant to indicate that the customer has arrived. A return notification from the restaurant that the customer has been added to the list of waiting customers triggers the customer's transition to the *Wait_to_Place_Order* state where it then waits until the server is available to take its order. A message from the restaurant that the server is available then triggers the customer to transition to the *Place_Order* state. The customer remains in this state until the restaurant notifies it that its meal is ready at which point it exits the system.

The *statechart_service* state chart along with the two list objects (*Customers_In_Line* and *Customers_Placing_Order*) represent the behavior of the restaurant. Immediately upon the creation of the restaurant, the state chart proceeds to the *wait_for_customer* state. Every time that the state chart enters this state, the restaurant checks its *Customers_In_Line* list to see if any customers are ready to be served. Alternatively, whenever a new customer enters the restaurant, if the restaurant is waiting in this state, then it will trigger the transition to the *take_order* state. Upon entering the *take_order* state, the restaurant takes the first customer on the list, notifies that customer that it is now being serviced, and removes the customer from the *Customers_In_Line* list and adds it to the *Customers_Placing_Order* list. Since the restaurant only has a single server, only a single customer can be on the *Customers_Placing_Order* list at a time; however, any number of customers can be on the *Customers_In_Line* list. Using the same distribution as the DES and SD implementations, the restaurant remains in the *take_order* state for an amount of time determined by a triangular distribution with a minimum value of 0.417 minutes, a maximum value of 1.5 minutes, and with a most likely value of 0.75 minutes at which point it (1) transitions back to the *wait_for_customer* state, (2) notifies the customers that its business is complete, (3) increments the counter for the total number of customers processed, and (4) removes the customer from the *Customers_Placing_Order* list.

The final state chart shown in Figure 2C (*Generate_Customers*) displays the process for generating agents during execution. The *initialize* state generates the restaurant agent before starting the process of generating customers to ensure that customers cannot try to access the restaurant before it has been created. The state chart then transitions to the *Generate_Customer* state after one simulated time step passes. This state links back to itself based on a time delay and generates one customer each time that the state is executed. The delay is stochastic based on the average interarrival time between customers using an exponential distribution with an expected value (*E(X)*) with a mean (β) of 0.75 minutes ($E(X) = \beta$) in an identical process to that of the DES implementation.

The combination of these state charts results in the following system behavior: (1) a customer agent is generated using the interarrival distribution defined above; (2) the customer sends a message to the restaurant agent as a notification that the customer has arrived; (3) the customer is then added to the list of people waiting to place orders (*Customers_In_Line*); (4) if there is currently nobody being served, the restaurant moves to the *take_order* state, the customer is moved from the *Customer_In_Line* list to the *Customers_Placing_Order* list, and the customer moves to the *Place_Order* state; (5) otherwise, the customer waits in a first-in first-out sequence on the *Customers_In_Line* list before moving to the *Place_Order* state; (6) the customer waits for a period of time determined by the triangular processing distribution described above; (7) the restaurant then notifies the customer that the order is complete, the customer is removed from the *Customers_Placing_Order* list, and the customer leaves the system; and (8) the restaurant checks to see if another customer is waiting in line and (8a) if yes, then for the first customer in line, repeat steps 4-8 otherwise, (8b) wait for a new customer to arrive to the restaurant and the restaurant remains in the *wait_for_customer* state.

3.4 The Multi-method Model

The multi-method implementation combines elements from all three of the previous implementations. Customer arrivals are represented as a flow rate using a stock and flow diagram similar to Section 3.2. Customers are represented as agents which contain the same *statechart_enter_restaurant* state chart that is shown in Section 3.3. The restaurant is represented using the same sequence of DES arrival, queue, process, and exit nodes with the server represented as a resource as in Section 3.1. All of the various implementation methods now interact in order to represent the interactions between the customers and the restaurant and simulate the total number of customers processed through the system. Figure 2D displays the layout of the multi-method implementation.

The flow of customer arrivals occurs using the same formula provided in (1) by the SD implementation. This provides a continuous accumulator into the *New_Customers* stock; however, unlike the SD model, there is not an accumulator on the stock and flow diagram that counts the total number of arrivals to the system as these are now tracked in the DES portion of the model. The outflow from the *New_Customers* stock is handled by the *Add_Customer* function which is triggered every time that the value of *New_Customers* exceeds a whole number value. This decreases the value of the stock by one and manually injects a new customer agent into the DES system. Therefore, the arrival node on the DES model does not operate using an interarrival distribution in this model, instead it waits for the specific notification that it needs to inject a customer. Upon creation, a customer enters the *Enter_Restaurant* state and waits for the restaurant to tell it when to transition to its next state.

The *take_order* process node handles the remainder of the actions and is comprised of a First-In First-Out queue followed by a single server process (driven by the availability of a single resource). When arriving to *take_order*, a customer switches to the *Wait_to_Place_Order* state. Once the server is ready to take a new order, the first customer in line is notified to switch to the *Place_Order* state and the server resource is seized to ensure that only one customer is processed at a time. The processing time for each customer uses a triangular distribution with a minimum value of 0.417 minutes, a maximum value of 1.5 minutes, and with a most likely value of 0.75 minutes just like the previous implementations. After the processing time has passed, the server resource is released, and the customer (1) transitions to the endpoint of its *statechart_enter_restaurant* state chart, (2) moves to the *exit_restaurant* node, and (3) is removed from the simulation. The *exit_restaurant* node keeps track of the total number of customers that pass through the restaurant.

4 RESULTS AND DISCUSSION

We conduct Monte Carlo experiments with the same configurations on each of the simulation implementations in order to collect equivalent types of results from each. The outcomes of interest include the total number of customers served, the total size of the queue at the end of the run, and the total amount of real time required to run the simulations. All of the experiments were conducted on the same machine with the following properties: Windows 7 Professional – Service Pack 1; AMD Opteron(TM) Processor 6274 @ 2.20 GHz (2 processors); 96.0 GB of Installed memory (RAM); 64-bit Operating System; and 32 cores. Differential equations are handled using Euler approximations.

The Monte Carlo experiments conduct 10 runs with 10 to 30 replications for each implementation using random seeds in each run. The experiment automatically stopped running once a 95% confidence interval was reached with respect to our target variable which is the number of customers in the queue (the *Customers_In_Restaurant* node of the DES model, the *Customers_In_Restaurant* stock for the SD model, the size of the *Customers_In_Line* list for the ABM model, and the *take_order* stock for the MPM model) once at least 10 runs had completed. AnyLogic uses the OptQuest engine to conduct the minimum number of runs needed to reach the objective solution based on the target variable (AnyLogic 2016). Table 1 provides the configurations of the Monte Carlo experiments.

Table 1: Experiment values for Monte Carlo experiments across each modeling method implementation.

Model Component	**Modeling Method**	**Type of Distribution**	**Value**
Customer Arrivals - Discrete	ABM, DES	$Exponential^{(Mean)}$	Mean = 0.75 minutes
Customer Arrivals - Flow	SD, MPM	$Exponential^{(Lamda)}$	Lamda = 0.75 minutes

Time to Process Customer at Server	ABM, DES, MPM, SD	*Triangular* (Minimum, Maximum, Mode)	Min = 0.417 minutes
			Max = 1.5 minutes
			Mode = 0.75 minutes

Service Delay	SD	Constant	1.997 minutes
Number of Servers	ABM, DES, MPM, SD	Constant	1

For the SD implementation, we first run an optimization experiment to determine the value of the *Service_Delay* to use for the Monte Carlo experiments. This experiment varies *Service_Delay* between 0 and 2 minutes with the objective of producing as close a match as possible to the mean number of customers served from the DES Monte Carlo experiment (274.9 customers). We set 0 as the minimum bound to explore the cases where delays within the system prior to being serviced are assumed to be negligible. The optimization found that 1.997 minutes for the Service_Delay produced the closest match on average with an objective value of 79.65 *Customers_In_Restaurant*.

For each Monte Carlo experiment, we collect statistics on the *total customers out* (TCO_f), the *total queue size* (TQ_f) remaining at the end of each simulation as well as the amount of real-time elapsed while running each simulation. The minimum, maximum, and mean values for TCO_f and TQ_f over each set of runs are collected and the mean confidence interval is displayed in Table 2. Including replications, each experiment produced 100 runs, thus providing 100 data points to calculate the means and mean confidence intervals for each output.

Table 2: Monte Carlo experimental results for the *total customers out* (TCO_f) and *total queue size* (TQ_f). Columns 2, 3, 6, and 7 reflect single runs and columns 4, 5, 8, and 9 reflect all runs per implementation.

	Total Customers Out (TCO_f)				**Total Queue Size (TQ_f)**			
Modeling Method	**Min**	**Max**	**Mean**	**Mean Confidence Interval**	**Min**	**Max**	**Mean**	**Mean Confidence Interval**
ABM	319	350	335.66	±0.948	24	118	66.54	± 3.678
DES	316	345	334.48	±1.036	8	120	64.44	± 4.109
SD	357.25	358.11	357.78	±0.033	40.69	44.28	42.18	± 0.128
MPM	326	350	336.63	±0.899	48	73	61.83	± 0.898

We conduct a check to estimate the expected number of customers using an assumption that a queue always exists since the average arrival rate exceeds the average service rate. On average we expect 400 arrivals ($E[A] = \frac{1}{0.75} * 300$), 337 departures ($E[D] = \frac{1}{0.889} * 300$), and 63 in the queue ($E[Q] = 400 - 337$). The 337 departures fits inside the confidence interval of TCO_f for the MPM model and lands just above the intervals from the ABM and DES models and 63 expected customers remaining in the queue falls inside the confidence intervals for ABM, DES, and MPM. It is not surprising that these numbers fall slightly outside of the other confidence intervals since the models do not assume a constantly full queue.

The TCO_f confidence intervals display that: (1) ABM overlaps with both DES and MPM; (2) DES overlaps ABM only; (3) SD does not overlap any of the other confidence intervals; and (4) MPM overlaps

ABM only. The TQ_f confidence intervals shows: (1) ABM, DES, and MPM all overlap; and (2) SD does not overlap any of the others. Therefore, we conclude that the selection of a paradigm contributes significantly to the computation of simulation outputs.

These results show the importance placed on the variables of interest for the system for selecting a modeling paradigm, including the choice to mix more than one paradigm. We purposefully selected queue length and throughput to illustrate the impact that each paradigm's assumptions have, including: (1) interaction type (i.e. agent-agent communication or differential equation) and (2) representation of time advance. The fact that independent implementations of the same conceptual model on the same platform yields significantly different results provides a warning sign to modelers that MPM might take small differences between paradigms and amplify their effect or reduce large differences between paradigms when they should not. We observe that the MPM model yields a significantly different outcome from the SD model, but we do not find evidence that the implementation provides a significantly different outcome from ABM or DES with respect to the variables of interest in this case. The selection of ABM, DES, or MPM does not provide any additional gains in accuracy with respect to these variables.

Additionally, the results show that (1) DES overlaps MPM's confidence intervals for execution time and executed much quicker than (2) ABM and SD whose confidence intervals overlap. The SD and ABM implementations taking the longest to run makes sense as the SD model's stocks and flows update every time tick (DT) and the ABM model's agents operate using a sequence of communications (a process that is simplified and abstracted out of the DES model). This result challenges the intuitive notion that MPM takes longer to run in general and points to the underlying simulation implementation as a source of additional execution time. Table 3 provides the amount of real time taken to execute the Monte Carlo simulations in seconds. Columns 2 and 3 provide the minimum and maximum time any single run lasted, column 4 provides the average length of time that the runs lasted, and column 5 provides the interval around the mean that we would generally expect a run to take to reach completion during execution.

Table 3: Execution time (real time passed) for each model in seconds.

Modeling Method	**Minimum**	**Maximum**	**Mean**	**Mean Confidence**
ABM	1.264	10.081	5.701	± 0.527
DES	1.079	5.727	3.46	± 0.271
SD	1.796	10.554	6.21	± 0.527
MPM	1.125	4.875	3.736	± 0.196

Interestingly, the MPM implementation, which contains SD flows , took less time to execute than the ABM model on average (comparatively fewer updates per time step). This suggests that the number of computations required to handle the interactions between agents approaches the number of calculations within the SD model. The MPM model taking less time to execute than the ABM and SD models may suggest that the MPM paradigm allowed us to efficiently represent the system's parts. However, this finding may have been driven by the fact that the conceptual model contains an intuitive mapping to a queueing-based configuration, which simultaneously produced the most computationally effective result. We recommend the work of Spiegel et al. (2005) for a discussion on the role of assumptions and constraints in model development.

5 CONCLUSION

The selection of a modeling paradigm has a significant impact on the range of results produced when executing a simulation. Therefore, it is important to select an appropriate paradigm when designing a model based on the requirements of the model and the assumptions that pertain to the modeling paradigm. This work is limited by using a generic restaurant scenario and not having actual data with which to validate the results, because of this we cannot determine which model produced the closest to expected results. The best indicator of success that we have between paradigms, lies with the DES and MPM implementations executing the fastest indicating that the use of DES queues may have been the most appropriate method for implementing the simulation models.

Future work involves expanding the evaluation of simulation implementations using varied modeling methods to include: more complex systems; running simulations using different hardware configurations; testing simulation objectives that may be better fits for non-DES paradigms (i.e. present a use case that is primarily modeled in SD and another that is prime for ABM); and conduct experiments with real scenarios that include data that can be used for validating the results, thereby providing a way to determine which method is a better candidate for implementing different conceptual modes. Conducting these future studies will provide a more complete idea of the effects that using different paradigms to implement the same model produces on the results.

REFERENCES

AnyLogic. 2016. "AnyLogic Help: Optimizing Stochastic Models." AnyLogic 7 Pro. Help, AnyLogic North America, Chicago, Illinois. Accessed June 29, 2016. www.anylogic.com/anylogic/help/.

Balaban, M., P. Hester, and S. Diallo. 2014. "Towards a Theory of Multi-method M&S Approach: Part I." In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Diallo, I. Ryzhov, L. Yilmaz, S. Buckley, and J. Miller, 1652-1663. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Press.

Djanatliev, A., and R. German. 2013. "Prospective Healthcare Decision-making by Combined System Dynamics, Discrete-event and Agent-based Simulation." In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 270-281. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Press.

Fishwick, P. A., and B. P. Zeigler. 1992. "A Multimodel Methodology for Qualitative Model Engineering." *ACM Transactions on Modeling and Computer Simulation (TOMACS) 2*(1): 52-81.

Fishwick, P. A. 1995. *Simulation Model Design And Execution: Building Digital Worlds*. Upper Saddle River, New Jersey: Prentice Hall PTR.

Forrester, J. W. 1961. *Industrial Dynamics*. Cambridge, Massachusetts: The MIT Press.

Forrester, J. W., and P. M. Senge. 1980. "Tests for Building Confidence in System Dynamics Models." *System Dynamics, TIMS Studies in Management Sciences 14*: 1-40.

Gilbert, N. 2008. *Agent-based Models (Series/Number 07-153: Quantitative Applications in the Social Sciences)*. Los Angeles, California: SAGE Publications.

Hardebolle, C., and F. Boulanger. 2009. "Exploring Multi-paradigm Modeling Techniques." *SIMULATION 85*(11/12): 688-708.

Kelton, W. D., R. P. Sadowski, and N. B. Swets. 2010. *Simulation with ARENA (Fifth Edition)*. New York, New York: McGraw-Hill, Inc.

Kuhn, T. S. 1970. *The Structure Of Scientific Revolutions (International Encyclopedia Of Unified Science, Vol. 2, No. 2)*. Chicago, Illinois: University of Chicago Press.

Langroodi, R. R. P., and M. Amiri. 2016. "A System Dynamics Modeling Approach for a Multi-level, Multi-product, Multi-region Supply Chain under Demand Uncertainty." *Expert Systems with Applications 51*: 231-244.

Lorenz, T., and A. Jost. 2006. "Towards an Orientation Framework in Multi-paradigm Modeling: Aligning Purpose, Object and Methodology in System Dynamics, Agent-based Modeling and Discrete-event Simulation." In *Proceedings of the 24th International Conference of the System Dynamics Society*, 1-18. Hoboken, New Jersey: Wiley.

Lynch, C., J. Padilla, S. Diallo, J. Sokolowski, and C. Banks. 2014. "A Multi-paradigm Modeling Framework for Modeling and Simulating Problem Situations." In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Diallo, I. Ryzhov, L. Yilmaz, S. Buckley, and J. Miller, 1688-1699. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Press.

Lynch, C. J., and S. Diallo. 2015. “A Taxonomy for Classifying Terminologies that Describe Simulations with Multiple Models.” In *Proceedings of the 2015 Winter Simulation Conferenc*e, edited by L. Yilmaz, W. Chan, I. Moon, T. Roeder, C. Macal, and M. Rossetti, 1621-1632. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Press.

Mackenzie, N., and Knipe, S. 2006. “Research Dilemmas: Paradigms, Methods and Methodology.” *Issues in Educational Research 16*(2): 193-205.

McGregor, S. L., and J. A. Murnane. 2010. “Paradigm, Methodology and Method: Intellectual Integrity in Consumer Scholarship.” *International Journal of Consumer Studies 34*(4): 419-427.

McKenzie, F. D. 2010. “Systems Modeling: Analysis and Operations Research.” In *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, edited by J. A. Sokolowski and C. M. Banks, 147-180. Hoboken, New Jersey: John Wiley & Sons, Inc.

Mosterman, P. J., and H. Vangheluwe. 2004. “Computer Automated Multi-paradigm Modeling: An Introduction.” *SIMULATION 80*(9): 433-450.

Pretorius, L., and S. J. Benade. 2011. "A Systems Dynamics Approach to Competing Technologies: Exploring Uncertainty of Interaction and Market Parameters." *South African Journal of Industrial Engineering* 22(2): 27-39.

Sokolowski, J. A. and C. M. Banks. 2009. *Modeling and Simulation for Analyzing Global Events*. Hoboken, New Jersey: John Wiley & Sons.

Spiegel, M., P. F. Reynolds Jr, and D. C. Brogan. 2005. “A Case Study of Model Context for Simulation Composability and Reusability.” In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 437-444. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Press.

Sterman, J. D. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Boston, Massachusetts: Irwin/McGraw-Hill.

Vangheluwe, H., J. de Lara, and P. J. Mosterman. 2002. “An Introduction to Multi-paradigm Modelling and Simulation.” In *Proceedings of the AIS’2002 Conference (AI, Simulation and Planning in High Autonomy Systems)*, 9-20. Vista, California: The Society for Modeling and Simulation International.

Villa, F., and R. Costanza. 2000. “Design of Multi-paradigm Integrating Modelling Tools for Ecological Research.” *Environmental Modelling and Software 15*(2): 169-177.

Zeigler, B. P. 1984. *Multifacetted Modelling and Discrete Event Simulation*. Orlando, Florida: Academic Press Professional, Inc.

AUTHOR BIOGRAPHIES

SAIKOU Y. DIALLO is a Research Associate Professor at the Virginia Modeling, Analysis and Simulation Center at Old Dominion University. He received a BS in Computer Engineering and a MS and PhD in Modeling and Simulation from Old Dominion University. His email is sdiallo@odu.edu.

CHRISTOPHER J. LYNCH is a Senior Project Scientist at the Virginia Modeling, Analysis and Simulation Center. He is currently pursuing a Ph.D. in M&S from ODU where he also received his MS in M&S in 2012 and a BS in Electrical Engineering in 2011. His email address is cjlynch@odu.edu.

JOSE J. PADILLA is a Research Assistant Professor at the Virginia Modeling, Analysis and Simulation Center at Old Dominion University and head of the Human Dynamics Lab. He received his Ph.D. in Engineering Management from Old Dominion University. His email is jpadilla@odu.edu.

ROSS GORE is a Research Assistant Professor at the Virginia Modeling, Analysis and Simulation Center. He holds a Ph.D. and a MS in computer science from the University of Virginia and a bachelor’s degree in computer science from the University of Richmond. His email address is rgore@odu.edu.