USING QUALITATIVE DISCRETE EVENT SIMULATION TO CALCULATE NEAR EXACT OUTPUT STATISTICS

By

YEN PING LEOW

Bachelor of Science in Mathematical and Computer
Science
University of Adelaide
Adelaide, Australia
2000

Master of Science in Industrial Engineering and
Management
Oklahoma State University
Stillwater, Oklahoma
2002

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 2008

USING QUALITATIVE DISCRETE EVENT

SIMULATION TO CALCULATE NEAR EXACT

OUTPUT STATISTICS

Dissertation Approved:

Dr. Ricki G. Ingalls
Dissertation Adviser

Dr. David B. Pratt

Dr. Camille DeYong

Dr. Dursun Delen

Dr. A. Gordon Emslie
Dean of the Graduate College

DEDICATION

This dissertation is dedicated to my loving husband,

Dr. Loay Sehwail

and my parents,

Swee Meen Leow and See Moy Soo

whose love and devotion have helped me accomplished my educational goals.

ACKNOWLEDGEMENT

First, I would like to express my gratitude to my advisor Dr. Ricki Ingalls for the guidance, encouragement and knowledge he has provided me throughout the course of writing this dissertation.

I would like to extend my sincere appreciation to my dissertation committee: Drs. David Pratt, Camille DeYong and Delen Dursun for all your time, input and ideas.

I am most grateful to my family for their support of my graduate studies. This work would not have been possible without the tireless support of my husband, Loay, who has endured many financial, personal and career sacrifices to see me reach this goal. To my son, Munir, who has helped me to overcome the stress of dissertation writing. To my father, Swee Meen, who has supported and encouraged me to complete my graduate studies. To my mother, See Moy, who inspired me to enjoy learning. To my brother, Kai Siong, and my sisters, Fong Ping, Li Ping and Siew Ping, I am grateful for their support of my efforts.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Overview

A simulation is the imitation of the operation of a process or system over time. It attempts to build an experimental device that will act like a real system in aspects that are important to the users. Simulation has been and will continue to be a widely used technique for solving many problems in engineering, business, physical science, artificial intelligence, economics and many other fields. Simulation is a powerful and important tool that provides a method for evaluating existing or alternative decisions, plans and policies without having to conduct experiments on the real system.

Discrete event simulation is intuitively appealing to users. Despite the fact that it mimics what happens in a real system, it also handles discontinuous functions, randomness and dynamics [Ingalls, et al. 2000]. It also allows users to analyze both the long term behavior and transient behavior. One can say that it is only limited by the amount of time you want to spend in gathering the data and programming it in the model. Otherwise, you can make the model of the world include anything you want. However, this comes at the cost of collecting the necessary data and spending enough time and effort building a complex model. Thus, modeling and analysis for a discrete event simulation can be time consuming and expensive.

A real world system is full of uncertainties and the processes or systems under study are sometimes highly random and poorly understood. In order to study a real world system, we often have to make a set of assumptions about how it works using statistical, mathematical or logical relationships. The statistical relationships used in discrete event simulation are often based on the incomplete information that could be gathered about the real system. Users would often have to make intuitive choices of a specific statistical distribution to be used as input to the simulation model.

Discrete event simulation outputs are random variables that are usually based on random inputs [Banks, et al. 2005]. The results of a discrete event simulation can be difficult to interpret because it can be hard to distinguish whether an observation is a result of system interrelationships or of randomness [Banks, et al. 2005]. Even if the results are interpreted correctly, the results of a discrete event simulation models could only tell a user how a system works under given conditions, it does not tell the user how to arrange the system so that it works best under these conditions. The issues discussed above are among some of the pitfalls of using discrete event simulation.

Qualitative discrete event simulation (QDES) is the qualitative counterpart of discrete event simulation. The qualitative approach to discrete event simulation was developed by Ingalls, et al. (2000) in an attempt to improve the techniques used in discrete event simulation and make it an even more robust decision tool. It is useful in situations where quality data is not available. In fact, it is designed to represent whatever level of knowledge is available about the real system [Ingalls, et al. 2000]. Discrete event simulation can be qualitatively defined by permitting imprecise specification of elements that are typically quantitatively specified either deterministically or in the form of a probability distribution [Ingalls 2000]. This approach assumes imprecise specification of event occurrences. More specifically, event execution time uncertainty is represented in the form of closed intervals in *R*. These intervals are called temporal intervals [Ingalls, et al. 2000]. The simulation time clock is also represented in the form of an interval. As a result of this new representation, the simulation executive, a key part of a simulation system that is responsible for controlling the time advance, has evolved to allow the implementation of the temporal intervals and the construction of a temporal interval clock. When temporal intervals are used in QDES, the simulation executive runs the execution processes and orders the events according to their execution times. The ordering of interval execution times is made possible with Allen's interval algebra [Allen 1983]. Allen's interval algebra formally expresses the temporal relations between intervals, the operations on them and the reasoning about them [Allen 1983].

Even in a simplest QDES model, it is likely to have ties in the ordering of event execution times. Instead of assuming a tie breaking strategy as in the regular discrete event simulation, the QDES simulation executive creates a process for every possible ordering and produces all possible outcomes of a simulation model. This characteristic makes QDES distinctive from the regular discrete event simulation. QDES characterizes all possible outcomes, including the outcome from a regular discrete event simulation, which makes this a far reaching potential benefit. One could check state variable values to see if they are in valid ranges in every process [Ingalls, et al. 2000]. For planning and scheduling problems, schedules would not have to be re-run every time something did not happen according to plan [Ingalls 2004]. Thus, as long as the input intervals are respected, at least one process in the run would characterize the events that had taken place and would give information on the next scheduling decision [Ingalls 2004]. The execution sequences of low or high probability events (the tail probability events) could be executed in the QDES model if users are interested in these events.

Ingalls (2000) developed scoring methodologies to assist in the analysis of data generated by the QDES model. However, more work needs to be done in this area. These scoring methodologies create some simple approximation that will provide users with information to rank the processes. They are made in attempt to approximate the event execution probabilities. With these approximations in hand, algorithms could be implemented to separate the more likely processes from the less likely processes. This would greatly expand the benefits of using qualitative discrete event simulation as a decision tool as it would help to answer questions like “What is the likelihood of disastrous events? What leads to these disastrous events? How can we prevent them?” These are the intriguing questions that users want from a decision tool.

1.2 Research Objective and Motivation

Previous work on Qualitative Discrete Event Simulation (QDES) by Ingalls et al. (2000), and Ingalls (2003) uses time interval representations in discrete event simulation (without any assumptions on the distribution on these intervals) to generate execution threads that characterize all possible outputs of the simulation.

Motivated by the intention to advance qualitative discrete event simulation as a decision tool and to expand its capability to provide meaningful output information, the author intends to conduct research in the qualitative discrete event simulation area. In particular, the author is interested in creating approximations to rank the processes generated from the QDES output. The current methodologies on scoring techniques fall short on ranking the processes according to the probabilities of the event sequences. The current QDES framework also does not have statistical analysis features to collect and report statistics on certain states or the values of global variables and other performance statistics based on the attributes of entities in the simulation model.

The primary objective of this dissertation research is to calculate the probability of occurrence and the event time distribution of each event that is generated using QDES, by imposing an assumption on the distribution of the delay intervals in the simulation. We are particularly interested in imposing a uniform distribution on the delay intervals. The resulting calculations can be used to calculate the simulation time-persistent output statistics and tally statistics for variables that are of interests to the users.

The second objective of this dissertation is to use these calculations (the probability distribution of the event sequences and the thread probabilities) to produce near exact output statistics. The advancement in the area of reasoning with probability to produce near exact output statistics from QDES will allow modelers to make wise decisions based on a single run, instead of sampling from multiple runs as in the regular discrete event simulation models. The near exact output statistics will be able to describe the distribution of different variables, such as the queue length, customer delay, server utilization, etc., together with information for the probability of these average values occurring for each variable and the event sequences that lead to them. By imposing statistical distribution to describe the threads, modelers can also identify low probability threads that are not significant, which could contribute to the area of thread scoring.

CHAPTER 2: LITERATURE REVIEW

2.1 Discrete Event Simulation

Discrete Event Simulation is the modeling of a system where the state variables of the system change at discrete points in time at the instant an event occurs. In every simulation model, there is a global variable that is used to keep track of the current value of the simulated time. This variable is known as the simulation clock. Since discrete event simulation involves the modeling of a system over time, it needs a mechanism to advance the simulated time from its current value to another. The most widely used mechanism follows the next-event time advance approach. We reserve the term Regular Discrete Event Simulation (RDES) to relate to simulation models that follow this approach.

The next event time advance approach uses a future event calendar to advance the simulated time and thus guarantee that all events generated during the simulation are executed in the correct chronological order according to the execution time of each event. Every event is represented by an event notice that contains the event execution time, event type and other data related to each particular event. The future event calendar keeps a list of all event notices for events that are scheduled to occur at a future time. Every event may schedule and/or cancel other events.

At the start of a RDES model, the simulation clock is initialized to zero. The occurrence times of future events are computed and inserted into the future event calendar. The future event calendar is ordered by event times and arranged chronologically, with the most imminent future event placed at the top of the future event calendar. The first event notice is then removed from the top of the future event calendar. The simulation clock is advanced to the scheduled time of the next event, which is retrieved from the event notice. At the instant this event occurs, the state of the system is updated to account for the occurrence of this event. Then, the completion time of this event is computed, an “end” event is scheduled and its associated event notice is placed on the future event calendar to signify the completion of the event. At this time, cumulative statistics could be collected so that a summary statistics for the simulation run could be computed at the end of the simulation. Then, the next “new” event at the top of the future event calendar is removed and the whole process of advancing the simulation clock, updating the states of the system and scheduling or canceling events continues until all the events have been executed or the stopping condition of the simulation model is satisfied.

The future event calendar is a strongly ordered list according to the event times. The event times must satisfy the following condition [Banks, et al. 2005]:

Let t_1 be the time of the most imminent event

Let be $t_2, t_3, \ldots, t_n$ the time of the 2nd, 3rd, .., nth most imminent event

At any given time, t:

$$t < t_1 \leq t_2 \leq t_3 \leq \ldots \leq t_n$$

Sometimes, several events may have the same execution time, such situation is known as a tie. The events that have the same execution times are referred to as simultaneous events [Banks, et al. 2005]. The result of a simulation model may vary as it depends on the order of how simultaneous events are executed. There are several tie-breaking mechanisms to determine the order of simultaneous events, sometimes ad hoc and specific to the simulation protocols being used. There is a type of arbitrary tie-breaking mechanism which allows a non-deterministic ordering of simultaneous events. Another type of tie-breaking mechanism allows the modelers to explicitly specify the order of simultaneous events or use a default ordering. The advantage of such mechanism is that it guarantees that the results of a simulation are reproducible.

The creation of a discrete event simulation model could be highly dependent on the availability of system data. If data is available, modelers normally would begin with the development of a frequency distribution or histogram of the data in attempt to fit the data to a family of distributions. Sometimes, it is impractical or impossible to get these data and even if the data is available, there are still other issues that need to be addressed such as data overload and data abstraction [Ingalls 1999]. Collecting data from the real system is time-consuming and requires a substantial resource allocation. Other than that, a great deal of effort is required to distill quality data and encode the information that is available into building a discrete event simulation model.

In a discrete event simulation, the inputs to a simulation model are usually represented using probabilistic distributions. These models rely on the specification of the probability distributions and the associated parameters that serve as inputs to the model. Each of these probability distributions carries statistical assumptions such as the identical and independent (i.i.d.), normality assumption and so on. Often, these assumptions are rarely valid but modelers are forced to make these assumptions. The choice of input model could significantly impact the prediction or decision to be made based on the simulation results. Depending on the type of distributions that is used in the model, the modelers are required to make the necessary assumptions. Every simulation analysis is based on a variety of assumptions that are made about the nature of the data. It is likely to make erroneous conclusions if one does not carefully evaluate the validity of the assumptions behind the analysis. It has become more of a standard procedure to make these assumptions when using discrete event simulations as an analysis tool.

Even if the assumptions could be considered fairly valid (for example in a strictly controlled environment), the task of identifying a probability distribution to represent the input data is not easy. Moreover, the uncertainty in a discrete event simulation model only relates to the choice of a family of distributions and the selection of the parameters as the parameters could either be entered as constant values or chosen from one of the probability distributions.

Discrete event simulation is a computer-based statistical sampling experiment [Law and Kelton 2000], therefore the result of any analysis based on input represented by probability distributions is itself a probability distribution. Each simulation run denotes the trajectory behavior of the simulation model in response to a particular experiment. Therefore, discrete event simulation does not provide desired information in regards to the viability of the system under study. For example, it does not tell you under what conditions the system will fail or succeed because these types of questions deal with the transient behavior of the system under study [Ingalls et al. 2000]. As a result of this, the modelers are left with no choice but to run a wide range of simulation runs with different inputs in order to grasp a deeper understanding about the system viability.

Another important characteristic of discrete event simulation is that it uses a time-point representation to model the occurrence times of future events. The time-point representation does not allow any uncertainty of information and often the exact relationship between two time-points is not known, but some constraints on how it could be related are known [Allen 1993]. Thus, the notion of time point is not decomposable and is not useful in a reasoning system. According to Allen (1993), an event may occur instantaneously at a particular time point but it appears that such event could be decomposed into sub-events if it is examined closely. For example, an event of "an arriving customer to a bank" at a specific time point could be decomposed to "open the front door" and "look for a free bank teller or queue and walk towards it". This poses a question to search for another possible representation for the modeling of discrete event systems.

2.2 Qualitative Simulation

Qualitative simulation is a reasoning technique that derives useful inferences from the modeling of a system when there is a lack of good quantitative information about the system under consideration. It is motivated by the desire to reason about the objects, processes or events in order to uncover all possible behaviors that may exist in the system. One major distinction between qualitative simulation and quantitative simulation is that a quantitative model is a result from a particular experiment while the output from a qualitative model is in response to all possible experiments [Cellier 1991]. When a qualitative simulation determines the next possible state, it can easily determine that there are several next possible states because of the imprecise nature of the data. A qualitative simulation will then execute each of these possible next states. Thus, the results of a qualitative simulation contain all possible event sequences. Qualitative simulation is particularly useful when the level of knowledge about the system under consideration is imprecise.

Another major distinction is in the representation of state variables and time representation. A model is quantitative if the state variables are real-valued, otherwise the model is qualitative [Cellier 1991]. This is an important distinction which could identify different types of simulation model. Simulation models could be largely classified into two major types, namely the continuous simulation and discrete event simulation.

For the case of a continuous simulation, which often deals with the modeling of a physical system over time by a representation in which the state variables change continuously with respect to time, a traditional continuous simulation model often involves differential equations that describe the rate of change or the interaction of state variables with time. The abstraction is based on ordinary differential equation, which is numerical in character. Kuipers (2001) described a continuous time qualitative simulation model using qualitative differential equation model as an abstraction of an ordinary differential equation, consisting of a set of real-valued variables with functional, algebraic and differential constraints among them. Qualitative differential equation consists of variables which are described in terms of their ordinal relations with a finite set of symbolic landmark values. The relationships could be a first-order relationship as simple as: As a increases, b increases. The values a and b could be described as increasing or decreasing over a particular ranges.

The concept of combining discrete event simulation with qualitative simulation was explored by Ingalls (1999) who developed a qualitative discrete event simulation methodology using temporal interval as the simulation time specification. Temporal interval allows the user to define time as an interval in the simulation which means that the current state of the simulation can occur at any time during the interval defined by $t =$

$[t^-,t^+]$. Temporal intervals are represented by modeling their endpoints, assuming for any interval *t*, the lesser endpoint is denoted by t^- and the greater by t^+. The level of uncertainty about the exact timing of the event occurrence is reflected in the width of the interval [Ingalls et al. 2000]. The more uncertainty there is in the event timing, the wider the corresponding temporal interval. This simulation methodology is developed within the modeling framework of event graphs and simulation graph models [Schruben 1983; Som and Sargent 1989; Yücesan and Schruben 1992; Schruben and Yücesan 1993; Ingalls et al. 1996].

Another qualitative approach to discrete-event simulation was taken by Ziegler and Chi (1992) using linear polynomial representation, known as the Symbolic Discrete Event System. This approach allows the modelers to express times symbolically for both situations when timing of events is unknown and when timing is known, but can vary. The time advance values could be expressed as linear polynomials such as 1, 2, s, t, 2s, t+s, 2t-s+9, etc., which must evaluate to nonnegative real numbers [Ziegler and Chi 1992]. Temporal interval specification extends the regular time base from real numbers to interval representation. This specification also serves the purpose of representing uncertainty in event execution times. The linear polynomial representation is used to allow manipulation of expressions for time with symbols representing unspecified event times. For example, let p = waiting time at register *A*, and q = process time at register *A*, then "$p+q$" is a valid expression according to symbolic discrete event system specification. This expression could be used to represent the time a customer spends at register *A* where p and q are symbols that are used to represent unspecified event times. When p and q are assigned numerical values, the expression evaluates to a real number.

2.3 Qualitative Discrete Event Simulation (QDES)

Qualitative discrete event simulation (QDES) is an event-scheduling approach to simulation modeling developed by Ingalls (1999) and Ingalls et al. (2000). It extends the concept of qualitative simulation to be applied particularly in discrete event systems. QDES uses the next event time advance approach as in a regular discrete-event simulation (RDES). At the start of a simulation model, the simulation clock is initialized to zero and the times of occurrence of the most imminent future event are determined and inserted into an events calendar. The simulation clock is advanced from the occurrence of one event to another at which the state of the system and the times of the occurrence of future events are updated. This process advances the simulation clock until all the events in the events calendar have been executed or canceled, or a certain stopping criterion is met.

One distinct characteristic of QDES is that it allows the modelers to specify elements qualitatively. These elements include the time of occurrence of the future events, the simulation time clock and the state variables. The times of occurrence of the future events are represented as time points in RDES. Unlike in RDES, QDES assume imprecise specification of event occurrences. The uncertainty of the event times is represented in a closed time interval in *R*, which is also known as temporal interval. There are two types of temporal intervals, namely the constant interval and the uncertain interval. A constant interval is an interval whose value remains the same throughout the entire simulation, while an uncertain interval is an interval that changes values every time the interval is evaluated. In using constant intervals, it is assumed that the actual value of the variables is a constant that lies somewhere within an interval. An uncertain interval is equivalent to the modeling of sampling from an unknown probability distribution that is bounded by an interval.

As a result of the time interval representation in QDES, the future event calendar is also represented in temporal intervals. Future event calendar in QDES is also sorted according to event times but it is not a strongly ordered list. Events are sorted according to interval mathematics outlined in Allen (1983). It is likely that there would be ties on the future event calendar because of the uncertain order of events. If there is a tie, QDES would not assume a tie breaking strategy as in the case for RDES. Instead, QDES would create threads that make up all of the possible ordering of ties, which differentiates between QDES and RDES. The differences are that RDES's future event calendar is a strongly ordered list according to the event times and RDES uses several tie-breaking mechanisms to determine the order of simultaneous events, sometimes ad hoc and specific to the simulation protocols being used. The future event calendar in QDES collects all the event notices whose execution order is uncertain and group them in a set, called the non-deterministically ordered set (NOS).

The capability of generating all possible scenarios is achieved with the thread generation algorithm. There are currently two algorithms developed so far, namely the Depth-First algorithm and the Breadth-First algorithm. These two algorithms are discussed in the next section in more detail. The execution of QDES algorithms resembles the RDES to some extents. The algorithms contain the basic steps of RDES such as initializing the simulation clock, advancing simulation clock from its current value to another, inserting events into the future event calendar, determining the next event to be executed and so on. When QDES is executed, the next possible state is determined. Due to the lack of precise data about the real-world, there could be several next possible states or a tie. QDES algorithm will have some major additional steps to ensure that each of these states will be executed in turns and result in a set of threads that will include all of the possible ordering of event sequences. An example of a new thread being generated is when the execution times (expressed in temporal intervals) for two or more events overlap.

The distinctive characteristic of QDES of generating all possible ordering of event sequences is known as coverage [Ingalls et al. 2000]. The coverage property ensures that all outcomes of QDES are characterized and no outcome will be missed out. In RDES, the simulation outcome is based on a sampling approach. The coverage property in QDES is very useful for planning and scheduling problems. Schedules would not have to be rerun every time if something did not happen according to plan. The output of QDES would be able to characterize the changes of events and give information on the next scheduling position, as long as the input interval is respected. Another advantage of coverage property is in debugging simulation models. QDES would characterize all possible scenarios, including anything that is characterized in a RDES model and sequences that have low probability of execution. This would give the modeler absolute confidence in the validity of the simulation model [Ingalls et al. 2000].

In RDES, input parameters to the systems are usually represented using probabilistic distributions and thus some modeling assumptions are made due to the lack of quality information about the real system. DES is a computer-based statistical sampling experiment [Law and Kelton 2000]. The result of any analysis based on input represented by probability distributions is itself a probability distribution. On the other hand, input parameters to QDES are expressed in temporal intervals and there is no assumption required to determine which probability distribution fits these input data.

The capability of generating all possible scenarios is achieved using thread generation algorithm. When a qualitative simulation is executed, the next possible state is determined. Due to lack of precise data about the real-world system, there could be several next possible states or a tie. Each of these states will be executed in turns and result in a set of threads that will include all of the event sequences. An example of a new thread being generated is when the execution time (expressed in time interval) for two events overlaps. In case of a tie in the future events calendar, QDES would create a thread for every possible ordering of the ties. This distinctive characteristic of qualitative simulation is known as coverage [Ingalls et al. 2000].

Coverage property ensures that all outcomes of QDES are characterized and no outcome will be missed out. Unlike in RDES, the simulation outcome is based on a sampling approach. This property is very useful for planning and scheduling problems. Schedules would not have to be rerun every time if something did not happen according to plan. The output of QDES would be able to characterize the changes of events and give information on the next scheduling position, as long as the input interval is respected. Another advantage of coverage property is in debugging simulation models. QDES would characterize all possible scenarios, including anything that is characterized in a RDES model and sequences that have low probability of execution. This would give the modeler absolute confidence in the validity of the simulation model [Ingalls et al. 2000].

2.4 Event Graphs and Simulation Graph Models

QDES is designed and developed using event graphs (EG) and simulation graph model (SGM). The event graphs and simulation graph framework was first introduced by Yücesan and Schruben (1992), Schruben and Yücesan (1993) and further extended by Ingalls (1999) and Ingalls et al. (2000). Event graphs were introduced in order to have a discrete event simulation methodology that is based on systems events. There are other types of discrete event models such as the block diagrams, process networks, and activity wheel or activity life cycle [Ingalls 1994]. These models are structured from the activity standpoint. The event orientation allows discrete event simulation concept to work without the traditional entity definition.

Let simulation graph, $G\ (V(G),E_S(G),E_C(G),\Psi_G)$ be a directed graph where $V(G)$ is the set of vertices of G, E_S(G) is the set of scheduling edges, $E_C(G)$ is the set of canceling edges and Ψ_G is an incidence function that associates with each edge of G. The Simulation Graph Model (SGM) is then defined as $S = (F,C,X,T,\Gamma,G)$ where

$F = \{f_v\text{: } v \in V(G)\},$ the set of state transitions functions associated with vertex v

$C = \{C_e\text{: } e \in E(G)\ \}$, the set of scheduling edge conditions

$X = \{X_e\text{: } e \in E(G)\ \}$, the set of execution edge conditions

$T = \{t_e\ \text{:}e \in E_S\ (G)\}$, the set of edge delay times as time intervals, and

$\Gamma = \{\gamma_e\ e \in E_S\ (G)\}$, the set of event execution priorities

The simulation graph G specifies the relationships that exist between the elements of the set of entities in a simulation model. The basic construct of the event graph with edge execution condition is given in Figure 1. The nodes labeled A and B represent events and the edge specifies that there is a relationship between the two events. The construct is interpreted as follows: If condition (*i*) is true at the instant event A occurs, then event *B* will be scheduled to occur *t* time units later. Event *B* will be executed *t* time units later with the state variables in array n set equal to the values in array *k* if condition (*j*) is true *t* time units later, in which t may assume the value of zero. Note that it is possible to specify an edge with no condition.

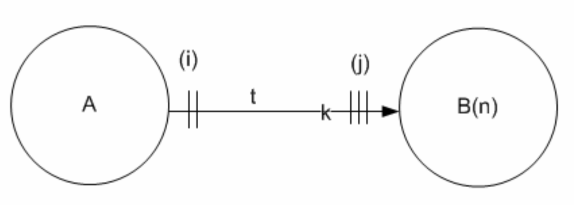


Figure 1. The basic construct for an event graph

2.5 Interval Mathematics and Its Use in Modeling

Allen (1983) mentioned that time-point representation does not allow any uncertainty of information and often the exact relationship between two time-points is not known. Thus, the notion of time point is not decomposable and is not useful in a reasoning system. Temporal interval representation is sometimes more useful in certain situations. An example to illustrate the use of temporal interval is shown in the modeling of the start time of a crisis. Let’s say that there is an alarm system that triggers to inform the appropriate authority when there is a crisis. In this case, the start time of the crisis is not known even if the start time of the triggering alarm could be determined accurately. An upper bound could be placed on the start time of the crisis if we assume that the crisis happen at a time earlier than the alarm time. In this case, it is only possible to specify the start time of the associated crisis as a time interval.

2.6 Temporal Intervals and Qualitative Time Calendar Construct

The uncertainty of the event time is represented in a closed time interval in $\mathcal{R}$, which is also known as temporal interval. Temporal intervals are used as a timing mechanism in the QDES framework. There are two types of temporal intervals used in QDES, namely the constant interval and the uncertain interval. A constant interval is an interval whose values remain the same throughout the entire simulation, while an uncertain interval is an interval that changes values every time the interval is evaluated. In using constant intervals, it is assumed that the actual value of the variables is a constant that lies somewhere within an interval. An uncertain interval is equivalent to the modeling of sampling from an unknown probability distribution that is bounded by an interval.

A temporal interval allows users to define time in terms of either a constant interval or uncertain interval in a QDES model. For example, the current simulation time could be defined as $t = [t^-,t^+]$. This means that the current state of the simulation can occur at any time during the interval *t*. As a result of the use of temporal intervals in QDES, events in the future events calendar and the future events calendar itself are also defined in the same manner. The future event calendar in a QDES framework is also sorted according to event times but it will not be a strongly ordered list. Events are sorted according to interval mathematics outlined in Allen (1983). Let's say, there are two events, A and B. Event A is scheduled to execute any time during the interval $t_A=[\ t_A^-,\ t_A^+]$ and event B is scheduled to execute any time during the interval $t_B=[\ t_B^-,\ t_B^+]$. Event A precedes event B in the future events calendar in a QDES model if:

1. $t_A < t_B$ (e.g. [2,3] < [4,5])

2. $t_A^- < t_B^-$ (e.g. [2,6] and [3,6])

3. $t_A^- = t_B^-$ and $t_A^+ < t_B^+$

It is likely that there would be ties in the order of events in a QDES model. Such situation happens when the execution times for these events intersect. For example, the order of events A and B will be uncertain if $t_A \cap t_B \neq \varnothing$. When this condition arises, QDES creates different threads to execute the possible orderings of this set of events. This set of events is known as the non-deterministically ordered set (NOS). In a QDES model, the future events calendar is made up of a union of NOSs. Given the current state of the model, the future events calendar will determine the NOS and then create a thread for each event in this set. If there are a total of N events in the NOS, there will be N threads created. In the i^{th} thread, the i^{th} event would be the first event to be executed. The remaining events in the set will still be in the future events calendar for that thread. Each thread maintains its own calendar and calendar time. The implementation of the future events calendar for NOSs in the QDES is known as the Temporal Interval Calendar (TIC).

2.7 Qualitative Discrete Event Simulation (QDES) Algorithm

The Simulation Graph Model (SGM) that was defined earlier as $S = (F,C,X,T,\Gamma,G)$ was used as the modeling framework in the QDES algorithm in Ingalls's work. There is additional mechanism that was introduced along with the definition of SGM in order to facilitate the execution of the model. These additional mechanisms include the global simulation clock τ that is represented in time interval, the events calendar L and the terminating conditions for the simulation ω. According to Ingalls (2003), the definition of the L is an ordered set such that L ={(x_1,γ_1,v_1,e_1,a_1,b_1), (x_2,γ_2,v_2,e_2,a_2,b_2),…} where x_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, a_i are the values of the edge attributes, and b_i are the times at which events are scheduled for the i^{th} event notice.

Two new sets are introduced to handle temporal intervals and they are defined as follows:

H = the set of saved states. This set is used to iterate through all the possible states in the simulation

N_h = the NOS

Two new variables are also introduced. Variable h is used as a counter to count the number of saved states and iterate through the set H. Variable n_h is used to iterate through the N_h set. The execution of the QDES model with the SGM and other definitions mentioned above will be explained in the next section. There are currently two QDES algorithms defined, the depth first algorithm [Ingalls et. al. 2000] and the breadth first algorithm [Agrawal 2002]. Each of these algorithms is explained in the next sections.

2.7.1 Depth First Algorithm

A depth-first algorithm was created and implemented by Ingalls et. al. (2000). The algorithm is designed to completely finish generating one whole thread before moving on to another thread. Any additional threads will be stored on a stack waiting to be executed at a later time. When there is a tie, two threads or more will be generated. The state of the simulation after each thread explosion is saved in a stack called the save-state stack, H. Each save-state consists of a global event calendar and state variable information so that all possible states in the simulation could be executed recursively. The algorithm would assume first thread, put the rest of the threads on stack and continue. When the generation of this thread is complete, the algorithm restores the system from the save-state stack and continues the simulation for the second thread. A save-state counter is set up to count the number of saved states and to iterate through the save-state stack. Recall that all event notices whose execution order is uncertain are grouped in a set, called the non-deterministically ordered set (NOS).

In depth-first algorithm, a NOS counter is used to iterate through the NOS. The depth-first algorithm is very useful if a complete analysis of all possible scenarios is needed in decision-making. Since all possible schedules are characterized and as long as input interval is not violated, users could make fast strategic decisions based on the previously run output, and thus saving time and effort.

2.7.2 Breadth First Algorithm

A breadth-first algorithm was developed and implemented by Agrawal (2003). In a breadth-first algorithm, all the active threads are evaluated simultaneously. The explosion of threads in this algorithm could be viewed as a tree diagram. The QDES simulation starts either with one parent node or a set of parent nodes. The simulation execution continues and spawns new threads from each of these parents, one by one, until all possible child nodes from each of these parents are explored. Before advancing to the next level down the tree structure, the breadth-first algorithm ensures that all sibling nodes are executed. Agrawal (2003) proposed a queue structure in his implementation to store the *breadth-first nodes*, a set consists of the event notice that are to be executed next, together with information such as the state variable and global events calendar. This queue is denoted as breadth-first node queue.

The breadth-first algorithm proceeds and gives system snapshots of all event sequences through time. It keeps track of possible system state that is available and how it leads to that system state. It would also be useful to eliminate threads that are considered unimportant or unlikely to give good information. For example, elimination of thread could be added to the breadth-first algorithm if the number of thread explosion exceeds a certain limit. However, depth-first algorithm has a speed advantage over breadth-first algorithm. This is due to the way breadth-first algorithm is structured.

2.8 Thread Scoring Techniques

Some of the threads that are generated by either of the above QDES algorithms may have a less likelihood to happen than other threads. As the complexity of the system increases, the number of threads generated using QDES will also increase. The thread generation could increase exponentially and causes the problem of extracting meaningful information from the output of the model. Thus, some scoring techniques were introduced to approximately rank the threads according to their likelihood of event execution sequences. Thread scores could be used in breadth-first algorithms to eliminate threads that have lower scores in relative to other threads and thereby reducing run time of the algorithm. Ingalls (1999) described three scoring techniques.

The midpoint ranking calculates the midpoint of each interval and ranks them accordingly. The second method is the multiple midpoint method, which also uses the midpoint of each interval. However, the resulting midpoint has taken into account the relative magnitude of all the midpoints and thus the event that is likely to execute first has a higher rank.

Let E_n, $n = 1, 2, \ldots, N$ ($N \geq 2$) denotes events with execution intervals $[L_n, U_\mathrm{n}]$ that overlap. Let M_n be the midpoint of interval $[L_n, U_n]$ for $n = 1, 2, \ldots, N$. Let Rank(M_n) denotes the rank of M_n. The calculation of using multiple midpoint ranking is given as in the following equation:

$$Rank(M_n) = \frac{M_n - \min\limits_{1 \leq n \leq N}(L_n)}{\min\limits_{1 \leq n \leq N}(M_n) - \min\limits_{1 \leq n \leq N}(L_n)}$$

The uniform method assumes that each overlap interval sections have equal chances of being executed next and each interval follows a uniform distribution. The score is given to each interval, determined according to the probability that a given event would be executed first. The uniform distribution is chosen because it could be easily programmed into the simulation and it has a closed form density function.

CHAPTER 3: DELAY CONSTRAINT VALIDITY

3.1 Introduction

In a RDES model, entities migrate from state to state while they work their way through the model. There are five entity states as described by Schriber and Brunner (2007), which are the Active state, Ready state, Time-Delayed state, Condition-Delayed State and Dormant state. The Active state is the state of the currently moving entity and only one entity moves at any instant time point. An entity in a Ready state indicates that it is ready to move. There could be more than one entity that are in Ready state, however, only one entity can move one-by-one. An entity in the Time-Delayed state is waiting for a known future simulated time to be reached so that it can then (re)enter the Ready state. The Condition-Delayed state is the state in which the entity is delayed until some specified condition comes about. If an entity is in the Dormant state, then it is in a state in which no escape will be triggered automatically by changes in model conditions, only modelers-supplied logic will be able to transform the entity from Dormant state to Ready state.

Generic RDES software uses five lists to organize entities in the five entity states, which are:

- Active Entity List - a list consists of only the active entity
- Current Event List - a list consists of entities in the Ready state
- Future Events List - a list of entities in the Time-Delayed state, ranked chronologically to the events' execution time
- Delay List - a list consists of entities in the Condition-Delayed state

- User-Managed List - a list of entities in the Dormant state

The interdependencies and the delay constraints that exist between events are managed using these lists. For example, by the time an entity's insertion in the Future Events list, the entity's delay time (also known as move time) is calculated by adding the value of the simulation clock to the exact known (sampled) duration of the time-based delay. Since the Future Events list is a strongly ordered list and events are executed based on exact (sampled) execution times, there is no need to check for the validity of the order of event execution.

Based on the QSGM algorithm proposed by Ingalls (1999) and Ingalls et. al. (2000), when the execution of a node triggers a new event to be scheduled at a later time interval, the new event's delay time is calculated by adding the time interval of the simulation clock to the time-based delay which is also represented in a time interval. Since time interval representation does not provide a means of strongly ordering the Future Events calendar in QDES, all possible order of event executions are simulated. Due to the uncertainties in the timing of the events, it is necessary to check for the validity of the delay constraint that exists between an event and the event that scheduled it. The algorithm proposed by Ingalls (1999) and Ingalls, et. al. (2000) did not consider checking for validity of delay constraints and thus may cause invalid threads to be generated.

Ingalls (1999) presents an example to demonstrate that QDES has the same functionality as RDES and also to show the ability of QDES to model at a more strategic level than lower operations level. The same example will be used to illustrate the discovery of invalid threads and the approach to eliminate the generation of invalid threads in the QDES algorithm. This chapter starts with the illustration of the Simple Queuing Model (same as the simple inbound outbound logistic pipeline model in Ingalls(1999)).

3.2 Simple Queuing Model

Figure 2 shows a simple queuing model with four events, “BEGIN”, “ENTER”, “START” and “LEAVE”. This model could be viewed as bank teller system where the system will “BEGIN” at time [0,0], customers “ENTER” the bank and wait to be served by the bank teller. The next customer in line will “START” the service process. When the service is finished, the customer will then “LEAVE” the bank. “BEGIN” is the first event to be executed and it is used to initialize the state variables. The variables that we are tracking in this model are the queue length, Q, status of the bank teller, S and number of customers that have exited, E. The status of the server is busy if S=0. If *S=1*, then the bank teller is idle. Changes in the state variables occur when an event occurs. The changes of the state variables of an event are stated below the event itself in Figure 2. For example, when “START” event occurs, the queue length is decremented by 1(*Q=Q-1*) and the status of the bank teller is changed to busy (*S=0*).

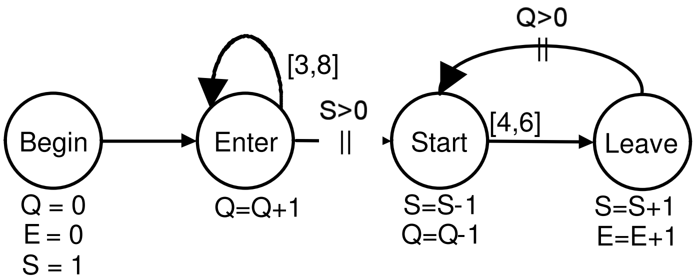


Figure 2. Simple Queuing Model

The conditions on the edges in Figure 2 are based on the state of the system. The condition of *S>0* is to check that the bank teller is available to service the next customer in line. If a customer arrives to the bank and the bank teller is not busy, the customer would be serviced immediately. Using the example model above, the author want to demonstrate how the approximation of the probability for each event sequence is done. In order to make the example model a simple model to illustrate the approximation, the QDES model for this example is set to terminate after two customers exited. So, the terminating condition is reached when the number of exits equals two (*E=2*). There are a total of 11 threads generated by Ingalls(1999) QDES algorithm with the above terminating condition. Figure 3 shows the event sequences of each thread [Ingalls 1999]. Threads 3, 5, 6 and 7 will not be considered in the research. This is because at node 14, event Enter[6,6] has an exact execution time and at node 11 and Enter[12,12] also has an exact execution time. In probability theory the probability that event Enter will execute at any exact time *x* is zero, for all real numbers *x*. Since there is no information gain from considering these threads, they will be eliminated from this example. Figure 4 shows the event sequences for the simple queuing model, with threads 3, 5, 6 and 7 eliminated.

3.3 Discovery of Invalid Threads

In this section, each of the eleven threads from Figure 4 will be reviewed and checked for the delay constraint validity. Starting at thread 1, events Begin[0,0], Enter[0,0] and Start[0,0] have execution time of [0,0]. There is no need to check for delay constraint validity for events that have zero execution time, as the delay time from the scheduling events to these events will be equal to [0,0].

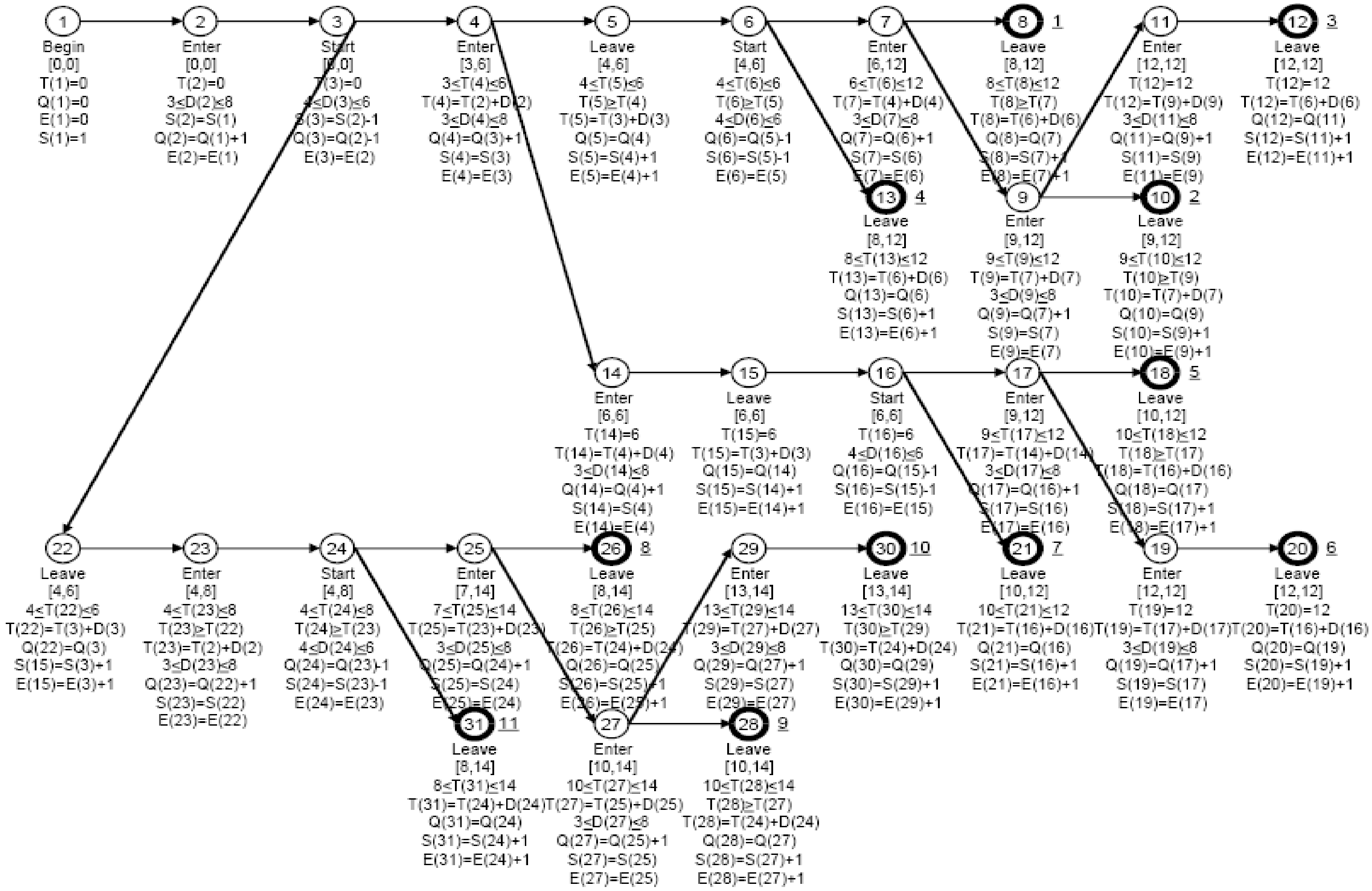


Figure 3. Event sequences of the simple queuing model

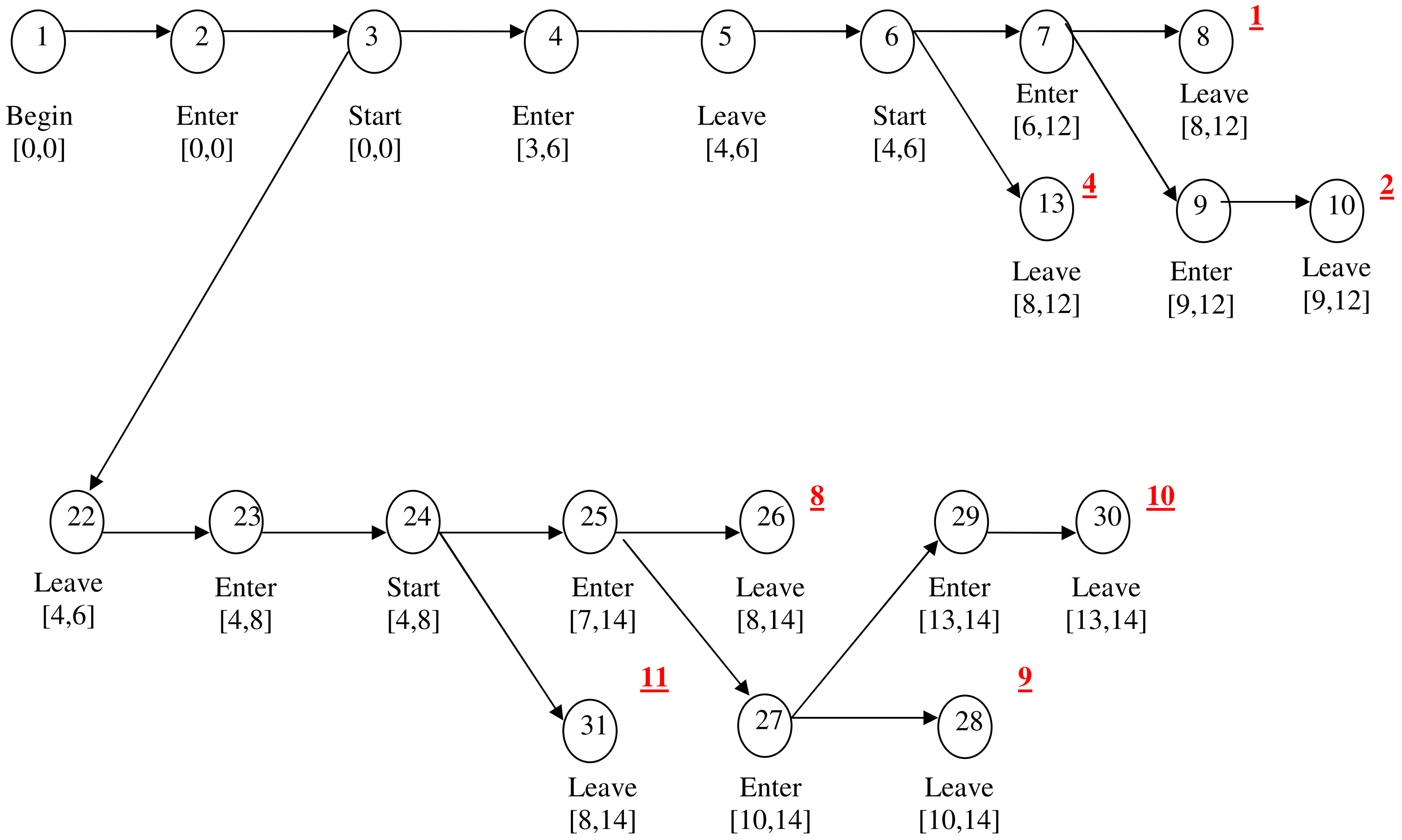


Figure 4. Event sequences for the simple queuing model with threads 3, 5, 6 and 7 eliminated

Another exception to checking delay constraint validity is when the scheduling event has a zero time-based delay. Recall that a newly scheduled event's delay time is calculated by adding the time interval of the simulation clock to the time-based delay which is also represented as a time interval. If the time-based delay is equal to [0,0], then the scheduled event is to be executed immediately, thus there is no need to check for delay constraint validity.

The next event on thread 1 would be Enter[3,6] (node 4), it is scheduled by Enter[0,0] (node 2) with the delay constraint of [3,8] (refer to Figure 5). In order for the thread to be valid (at least until node 4), the delay time from node 2 to node 3, add to the delay time from node 3 to node 4 should be within [3,8]. In other words, the total elapsed time from node 2 to node 4 has to fall in between [3,8]. Since [0,0]+[3,6] = [3,6] and [3,6] is within the delay constraint of [3,8], the execution of thread 1 until node 4 is considered valid, as $[3,6] \cap [3,8] = [3,6] \neq \varnothing$. Recall from interval arithmetic, [a,b]-[c,d]=[(a-d), (b-c)]. Delay constraint validity check is performed for all the subsequent nodes for thread 1 and for all other threads as well, which is shown in the next two pages.

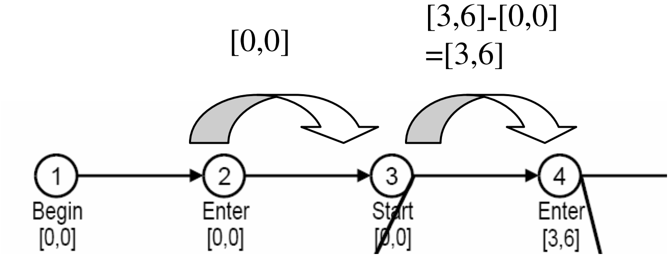


Figure 5. Checking delay constraint validity for Enter[3,6]

Thread 1

Note: Node 6 is omitted because there is a zero time delay from node 5 to node 6
Leave[4,6] (node 5) is scheduled by Start[0,0] (node 3) with delay constraint of [4,6]
([3,6]-[0,0])+([4,6]-[3,6])=[3,6]+[0,3]=[3,9]
Since $[3,9]\cap[4,6]=[4,6]\neq\emptyset$, thus thread 1 is valid up to node 5
Valid!

Enter[6,12] (node 7) is scheduled by Enter[3,6] (node 4) with delay constraint of [3,8]
([4,6]-[3,6])+([6,12]-[4,6])=[0,3]+[0,8]=[0,11]
Since $[0,11]\cap[3,8]=[3,8]\neq\emptyset$, thus thread 1 is valid up to node 7
Valid!

Leave[8,12] (node 8) is scheduled by Start[4,6] (node 6) with delay constraint of [4,6]
([6,12]-[4,6])+([8,12]-[6,12])=[0,8]+[0,6]=[0,14]
Since $[0,14]\cap[4,6]=[4,6]\neq\emptyset$, thus thread 1 is valid up to node 8
Valid!

Thread 2

Enter[9,12] (node 9) is scheduled by Enter[6,12] (node 7) with delay constraint of [3,8]
[9,12]-[6,12]=[0,6]
Since $[0,6]\cap[3,8]=[3,6]\neq\emptyset$, thus thread 2 is valid up to node 9
Valid!

Leave[9,12] (node 10) is scheduled by Start[4,6] (node 6) with delay constraint of [4,6]
([9,12]-[9,12])+[3,6](from node 9's delay bound)+([6,12]-[4,6])=[0,3]+[3,6]+[0,8]=[3,17]
Since $[3,17]\cap[4,6]=[4,6]\neq\emptyset$, thus thread 2 is valid up to node 10
Valid!

Thread 4

Leave [8,12] (node 13) is scheduled by Start [4,6] (node 6) with delay constraint of [4,6]
[8,12]-[4,6]=[2,8]
Since $[2,8]\cap[4,6]=[4,6]\neq\emptyset$, thus thread 4 is valid up to node 13
Valid!

Thread 8

Note: Node 24 is omitted because there is a zero time delay from node 23 to node 24
Leave[4,6] (node 22) is scheduled by Start[0,0] (node 3) with delay constraint of [4,6]
[4,6]-[0,0]=[4,6]
Since $[4,6]\cap[4,6]=[4,6]\neq\emptyset$, thus thread 8 is valid up to node 22
Valid!

Enter[4,8] (node 23) is scheduled by Enter[0,0] (node 2) with delay constraint of [3,8]
([4,8]-[4,6])+[4,6]=[0,4]+[4,6]=[4,10]
Since $[4,10]\cap[3,8]=[4,8]\neq\emptyset$, thus thread 8 is valid up to node 23
Valid!

Enter[7,14](node 25) is scheduled by Enter[4,8] (node 23) with delay constraint of [3,8]
[7,14]-[4,8]=[0,10]
Since $[0,10] \cap [3,8] = [3,8] \neq \varnothing$, thus thread 8 is valid up to node 25
Valid!

Leave[8,14] (node 26) is scheduled by Start[4,8] (node 24) with delay constraint of [4,6]
([8,14]-[7,14])+[3,8]=[0,7]+[3,8]=[3,15]
[3,8] is the delay bound for node 25. Since there is a zero time delay from node 23 to node 24, thus the delay bound from node 23 to node 25 could be extended also from node 24 to node 25.
Since $[3,15] \cap [4,6] = [4,6] \neq \varnothing$, thus thread 8 is valid up to node 26
Valid!

Thread 11
Leave[8,14] (node 31) is scheduled by Start[4,8] (node 24) with delay constraint of [4,6]
[8,14]-[4,8]=[0,10]
Since $[0,10] \cap [4,6] = [4,6] \neq \varnothing$, thus thread 11 is valid up to node 31
Valid!

Thread 9
Enter[10,14] (node 27) is scheduled by Enter[7,14] (node 25) with delay constraint of [3,8]
[10,14]-[7,14]=[0,7]
Since $[0,7] \cap [3,8] = [3,7] \neq \varnothing$, thus thread 9 is valid up to node 27
Valid!

Leave[10,14] (node 28) is scheduled by Start[4,8] (node 24) with delay constraint of [4,6]
[3,8] (delay bound from node 25) + [3,7] (delay bound from node 27)+([10,14]-[10,14])
=[3,8]+[3,7]+[0,4]=[6,19]
Since $[6,19] \cap [4,6] = [6,6]$ and P([6,6])=0, thus thread 9 is not valid.
Invalid!

Thread 10
Enter[13,14] (node 29) is scheduled by Enter[10,14] (node 27) with delay constraint of [3,8]
[13,14]-[10,14]=[0,4]
Since $[0,4] \cap [3,8] = [3,4] \neq \varnothing$, thus thread 10 is valid up to node 29
Valid!

Leave[13,14] (node 30) is scheduled by Start[4,8] (node 24) with delay constraint of [4,6]
[3,8] (delay bound from node 25) + [3,7](delay bound from node 27) + [3,4] (delay bound from node 29) + ([13,14]-[13,14])
=[3,8]+[3,7]+[3,4]+[0,1]=[9,20]
Since $[9,20] \cap [4,6] = \varnothing$, thus thread 10 is not valid.
Invalid!

Validity check is only required to be performed once for each node, if the node has been checked and verified that it is valid, it does not need to be checked again. Once a node has been verified as valid, the intersection of the delay constraint with the total elapsed time interval (from the scheduling event to its scheduled event) will become a delay bound that could be used in the calculation of total elapsed time for subsequent nodes in the same thread. For example, in node 10, instead of using the delay time for node 9, which equals to [10,14]-[7,14]=[0,7], the delay bound of [3,6] is used in the calculation of total elapsed time for node 10. This is because it has been calculated in node 9 that the minimum total elapsed time from node 7 to 9 is equal to 3 time units and the maximum total elapsed time is equal to 6. Thus, this information gained from checking delay constraint in preceding nodes need to be used in the validity check for subsequent nodes.

The delay bound for a node can be extended to other nodes under certain circumstances. This is especially true for nodes with zero delay time. For example, node 23 has a zero delay time, therefore the delay bound for a (scheduled) node that has node 23 as the scheduling node could be extended to the immediate successor node following node 23. In this case, the delay bound is extended from node 23 to node 24.

Based on the verification that has been done on each node, the QSGM output from the Simple Queuing Model has two invalid threads, which are threads 9 and 10.

These threads will be deleted from the QSGM output and will not be considered in further research. There are a total of five valid threads remaining in the Simple Queuing Model as shown in Figure 6.

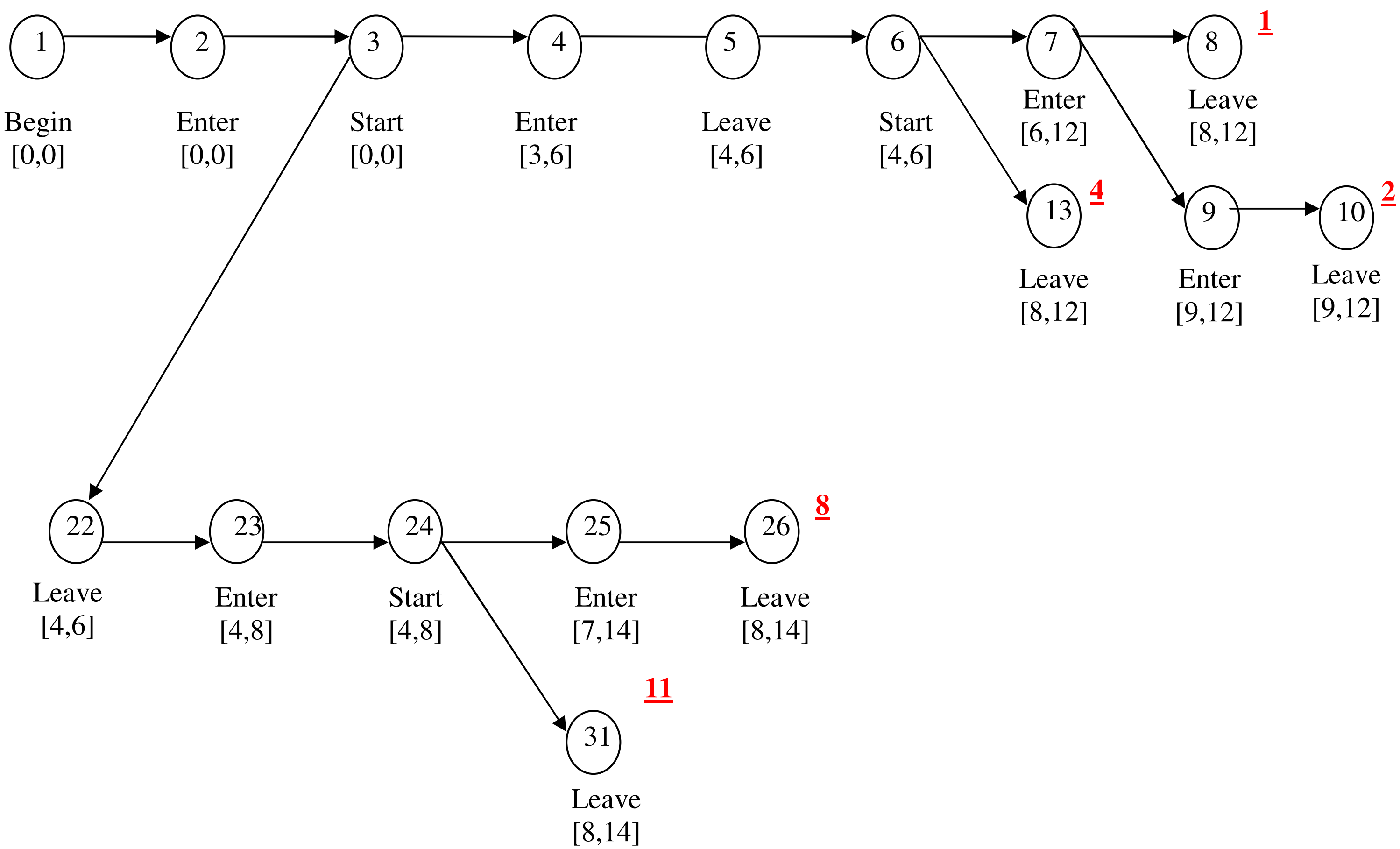


Figure 6. The Remaining Valid Threads from QSGM Output

3.4 Delay Constraint Algorithm

In this section, the algorithm that is created to check and verify delay constraints for a QSGM output using the methodology described in Section 3.3 is presented. The algorithm could be run to check the validity of each thread one by one, visiting each node only once. The definition of *L* is an ordered set, *L* ={ $(x_1, \gamma_1, v_1, e_1, a_1, b_1), (x_2, \gamma_2, v_2, e_2, a_2, b_2), \ldots$ } where x_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, a_i are the values of the edge attributes, and b_i are the times at which events are scheduled for the i^{th} event notice. If γ_i=1, this means that event *l* has a high priority and it will be executed immediately. The sets and variables that are used in the algorithm are defined as follows:

Let T={threads from QSGM output}.
For the simple queuing model example, T={1,2,8,9,10,11}

Let L_t={the events from thread t}={1,2,3,…}
L_t is an ordered set according to the sequence from QSGM output from thread *t*. For example,
$l_1 = (b_1, x_1, d_1, z_1, f_1, od_1)$
$l_2 = (b_2, x_2, d_2, z_2, f_2, od_2)$
Let $pos_t(l)$ = the position of event *l* in thread *t*
Let $md_{(l_i, l_j)}$ = the delay (in temporal interval) from event l_i to event l_j
Let *d'* = the accumulated elapse time

The algorithm loops through all the threads one by one, checking the delay constraint validity for each event in a thread, starting from the first event in each thread. The first task in the algorithm is to loop through all the events in the thread and determine the total elapsed time between each event and its scheduling event. If the total elapsed time falls within the delay constraint for event *l* then the execution of event *l* is considered valid. If the current event *l* has zero execution time or zero delay time, then the delay between *l* and its scheduling event is equal to [0,0]. If the current event *l* has non-zero execution time or non-zero delay time, the algorithm will proceed to check delay constraint validity.

In order to determine the total elapsed time between the current event *l* and its scheduling event, the algorithm will loop through each event (node) between the current event *l* and its scheduling event in reverse (starting from the last event) to check if there is a delay bound. If there is no delay bound for the current node, the delay time between the current node and its predecessor node will be calculated. Let's say the current node has an execution time interval of [a,b] and its predecessor node has an execution time interval of [c,d], then the delay time between the current node and its predecessor node is equal to [(a-d),(b-c)]. The algorithm will proceed to the next event.

If there is a delay bound for the current node, we want to determine if the position of the delay bound falls within the current event *l* and its scheduling event's position. For delay bound that meets this criterion, the delay time between the current node and its scheduling node will be retrieved from the variable $md_{(l_i,l_j)}$. The algorithm will now proceed to the event before the scheduling node and continue checking the delay constraint validity. Because we skip the nodes between current node and the current node's scheduling event, we did not check to see if these nodes have delay bound. The search for delay bounds starts from the node right before the scheduled event, namely the scheduled event's predecessor. There may more than one delay bound that exists between the scheduled event and the scheduling event. On top of that, there may be intersections that exist between the delay bounds. In the algorithm that we propose here, not all delay bounds will be considered as the search for delay bounds are based on the order of the event sequences in reverse, meaning starting from the scheduled event. Thus, the proposed checking delay constraint algorithm in this dissertation does not consider checking all delay constraints for all combinations of bounds that exists in a thread.

The effect of this approach in calculating the total elapsed time is that it is possible for a thread that has been determined to be valid using this approach, but in fact when using other approaches it is invalid. This is mainly due to the fact that not all combinations of delay bounds are considered when calculating the total elapsed time between the current event l and its scheduling event. However, this situation did not arise in the Simple Queuing System model as there is no intersection between delay bounds and all delay bounds are considered in the model when calculating the total elapsed time for an event in a thread.

The second task for the delay constraint algorithm is to determine the intersection of the total elapse time with the delay constraint ($d' \cap b_l$). If the result of the intersection turns out to be an empty set or is a zero-probability time interval, this means that the delay constraint for that particular thread is not valid and this thread will be deleted from the QSGM output. The remaining algorithm deals with storing the delay bounds in variable $md_{(l_i, l_j)}$.

The algorithm is shown in the following page. The comments that are meant for a statement in the algorithm are in *Italic* fonts under the statement.

Delay Constraint Algorithm

```
For each $t \in$ T
This for loop is created to assign the position of an event in a thread t to the variable $pos_t(z)$
$counter = 0$
    For each $l \in$ L$_t$
        $counter = counter + 1$
        $pos_t(l) = counter$
    Next $l$
Next $t$

For each $t \in$ T
    Initialize $M = \varnothing$, $V = \varnothing$, $md_{i,j} = 0, \forall$ defined $(i, j)'s$
    For $l \in L_t \mid pos_t(l) = 1, pos_t(l) + +$
        If $x_l \neq [0,0]$ or $b_l \neq [0,0]$ then
        If current event l has zero execution time, then the delay between l and its
        scheduling event is equal to [0,0] as well
            $d' = [0,0]$
            For $p = pos_t(l)$ to $pos_t(z_l) + 1$ step -1
            To loop through all the nodes between event l and its scheduled event's
            successor
                If $p > pos_t(z_l)$ then
                If p is greater than the position of the event that scheduled l,
                meaning there's a delay bound for event with position p that
                exists between l and its scheduled event
                    $d' = d' + (md_{z_{l'},l'} \mid pos_t(l') = p)$
                    Go to $pos_t(z_{l'}) + 1$
                Else if $b_{l'} = [0,0] \mid pos_t(l') = p$ then
                    $d' = d' + [0,0]$
                Else
                    $x1 = (x_{l'} \mid pos_t(l') = p)$
                    $x2 = (x_{l''} \mid pos_t(l'') = p - 1)$
                    $d' = d' + x1 - x2$
                    x1-x2 is the delay between event in position p
                    and p-1, or between p and its successor event
                End if
            Next $p$
        If $d'\cap b_l = \varnothing$ or $d'\cap b_l = [i,i] \mid i \in \Re$ then
            If the intersection of the total elapse time with the delay constraint
            is an empty set or is a zero-probability interval
                $T = T \setminus \{t\}$
                Go to next $t$
            End if

            If $\gamma_{l'} = 1 \mid pos_t(l') = pos_t(l) + 1$ then
            If the event that scheduled l's successor has a high priority flag,
            meaning that if it's execution time is the same as event l's execution time

                $md_{z_l,l} = d'\cap b_l$
                $md_{z_l+1,l} = d'\cap b_l$
                Assign new delay bounds with zero delay complication
            Else
                $md_{z_l,l} = d'\cap b_l$
            End if
          End if
        End if
    Next $l$
Next $t$
```

CHAPTER 4: PROBABILITY OF THREAD OCCURRENCE

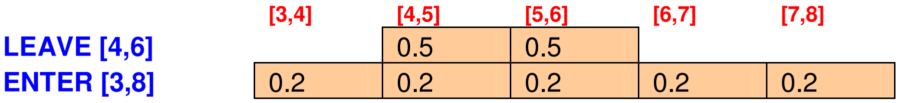
4.1 Introduction

The probabilistic approach to creating a statistical analysis for a QDES model depends on the possibility of finding a way to calculate the probability of a thread occurring in the model, the probability of an event occurring in a thread and the probability of an event executing first whenever there is more than one event in the NOS. In this chapter, we will present the method for calculating these probabilities and the associated algorithm that can be programmed and implemented in the QDES framework.

4.2 Probability of an Event Executing First

In order to estimate the probability for each individual thread, the minimal assumption is required. The assumption is that all event delay times are assumed to be uniformly distributed. The simulation clock is initialized to [0,0]. There are three events scheduled to execute at time [0,0] and they are BEGIN[0,0], ENTER[0,0] and START[0,0] in this order (refer to Figure 6). At the instant event ENTER[0,0] is executed, it schedules another event ENTER to be executed [3,8] time units later. Now, event START[0,0] is at the top of the future event calendar, so it is executed next. The instant event START[0,0] is executed, it schedules an event LEAVE to be executed [4,6] time units later. At this time, there are two events in the future event calendar, which are events ENTER[3,8] and LEAVE[4,6].

From the uniform distribution assumption on all the events, the event time intervals can be sectioned into predetermined time intervals and the probability of the event executing in each of these sections can be calculated. Without loss of generality, we will assume that the intervals are sectioned to one-unit time intervals. For example, if the time interval for event LEAVE[4,6] is sectioned to one time unit intervals, then LEAVE[4,6] would have a 0.5 probability of executing either in interval [4,5] or [5,6]. On the other hand, ENTER[3,8] has 0.2 probability of executing in [3,4], [4,5], [5,6], [6,7] or [7,8] (as shown in Figure 7).



	[3,4]	[4,5]	[5,6]	[6,7]	[7,8]
LEAVE [4,6]		0.5	0.5		
ENTER [3,8]	0.2	0.2	0.2	0.2	0.2

Figure 7. Probability distribution for events LEAVE[4,6] and ENTER[3,8]

As the exact timing of these two events is uncertain due to execution times and the order of event execution sequences is uncertain due to the overlapping execution times. QDES algorithm will spawn two threads, one assumes that event ENTER[3,8] will be executed first and the second thread assumes that event LEAVE[4,6] will be executed first. If event ENTER[3,8] executes first, then it must be executed in [3,8], so that the next imminent event LEAVE can execute next in [4,6]. On the other hand, if event LEAVE[4,6] executes first in [4,6], the event ENTER[3,8] will execute next in [4,8]. Notice how the execution time interval for the next event changes as a result of the uncertain event execution times and uncertain order of event execution sequences. With the uniform probability information on the two events, the probability of each event executing first could be calculated, as shown below. Note that the probability of an event executing in an interval [i,j] is denoted with P(event$\in$[i,j]).

Question: What is the probability of event LEAVE executing first?

P(LEAVE execute first in [4,5])

= P(LEAVE$\in$[4,5]) *($\frac{1}{2}$*P(ENTER$\in$[4,5]+P(ENTER$\in$[5,8]))

= $0.5*(0.5*0.2+0.6)=0.35$

P(LEAVE execute first in [5,6])

= P(LEAVE$\in$[5,6])* ($\frac{1}{2}$*P(ENTER$\in$[5,6]) + P(ENTER$\in$[6,8]))

$=0.5*(0.5*0.2+0.4)=0.25$

Question: What is the probability of event ENTER execute first?

P(ENTER execute first in [3,4]) = P(ENTER$\in$[3,4]) = 0.2

P(ENTER execute in [4,5])

= P(ENTER$\in$[4,5])*($\frac{1}{2}$*P(LEAVE$\in$[4,5] + P(LEAVE$\in$[4,5]))

$=0.2*(0.5*0.5+0.5)=0.15$

P(ENTER execute first in [5,6])

= P(ENTER$\in$[5,6])*($\frac{1}{2}$*P(LEAVE$\in$[5,6]))

$=0.2(0.5*0.5)=0.05$

As a result of the above calculations, P(LEAVE execute first in [4,6])

= P(LEAVE execute in [4,5]) + P(LEAVE execute in [5,6])

$= 0.35 + 0.25 = 0.6$

P(ENTER execute first in [3,6])

= P(ENTER execute in [3,4]) + P(ENTER execute in [4,5])+ P(ENTER execute in [5,6])

$= 0.2 + 0.15 + 0.05 = 0.4$

Figure 8 shows the results so far.

P{LEAVE[4,6] execute first in interval (i,j)}

(i,j)	(4,5)	(5,6)	Total
	0.35	0.25	0.6

P{ENTER [3,6] execute first in interval (i,j)}

(i,j)	(3,4)	(4,5)	(5,6)	Total
	0.2	0.15	0.05	0.4

Figure 8. The probabilities of events LEAVE[4,6] and ENTER[3,8] executing first, respectively

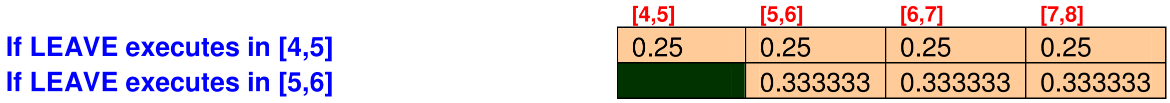
4.3 Conditional Probability

If we assume that LEAVE[4,6] is executed first, the probability of event ENTER executing next in [4,5], [5,6], [6,7] and [7,8] can be calculated. Based on the condition of when LEAVE[4,6] is executed first, in this case, we have two conditions, namely: (1) LEAVE is executed first in [4,5] and (2) LEAVE is executed first in [5,6], the probabilities of event ENTER executing next in [5,6], [6,7] and [7,8] are calculated and shown as below.

$$\text{P(ENTER in [4,8]|LEAVE }1^{st}\text{ in [4,5])=}\left(\begin{array}{l}\text{P(ENTER 2nd in [4,5] | LEAVE 1st in [4,5])}\\ +\text{P(ENTER 2nd in [5,6] | LEAVE 1st in [4,5])}\\ +\text{P(ENTER 2nd in [6,7] | LEAVE 1st in [4,5])}\\ +\text{P(ENTER 2nd in [7,8] | LEAVE 1st in [4,5])}\end{array}\right)$$

$$\text{P(ENTER in [5,8]|LEAVE }1^{st}\text{ in [5,6])}=\left(\begin{array}{l}\text{P(ENTER 2nd in [5,6] | LEAVE 1st in [5,6])}\\ +\text{P(ENTER 2nd in [6,7] | LEAVE 1st in [5,6])}\\ +\text{P(ENTER 2nd in [7,8] | LEAVE 1st in [5,6])}\end{array}\right)$$

If LEAVE executes in [4,5], then ENTER can execute next in [4,8]. Without considering the fact that LEAVE was executed in [4,5], we could redistribute the probability of ENTER executing in [4,8] by looking at the original probability distribution of ENTER[3,8]. Since it is equally likely for ENTER[3,8] to execute in any interval within [3,8] with a probability of 0.2, ENTER will still be equally likely to execute anywhere within [4,8] with a probability of 0.25. The probability distribution for ENTER[4,8] is shown in the first row of Figure 9. The same reasoning goes for the case if LEAVE executes in [5,6], the probability distribution for ENTER[4,8] is shown in the second row of Figure 9.



	[4,5]	[5,6]	[6,7]	[7,8]
If LEAVE executes in [4,5]	0.25	0.25	0.25	0.25
If LEAVE executes in [5,6]	■	0.333333	0.333333	0.333333

Figure 9. Redistribution of the original probability of occurrence for ENTER[4,8]

If LEAVE [4,6] executes in [4,5], then the probability for ENTER to execute in [4,5] is equal to 0.125 (half of 0.25). Since ENTER has a 0.25 probability to be in [4,5], the conditional probability of ENTER to execute in [4,5], given that LEAVE executes in [4,5] is equal to 0.5*0.25 = 0.125. Thus, P{ENTER execute second in [4,5]/LEAVE executed in [4,5]}=0.125/(0.125+0.25*3) ≅ 0.142857

For the ease of calculation and to save the hassle of redistributing the original probability of occurrence for ENTER[4,8], the following shows two calculations that are equivalent to the one described above. The third calculation is the simplified version that will be used in developing the probability algorithm in 4.8.3 Execution of the Probability Algorithm.

P(ENTER execute in [4,5] |LEAVE executed first in [4,5])

= P(ENTER in [4,5])

$$= \frac{\frac{1}{2}*0.2}{\left(\frac{1}{2}*0.2+3*0.2\right)} \cong 0.14286$$

P(ENTER execute in [4,5] | LEAVE executed first in [4,5])

= P(ENTER in [4,5])

$$= \frac{\frac{1}{2}}{\left(\frac{1}{2}+3\right)} \cong 0.14286$$

The rest of the conditional probabilities show similar calculations, except that we have to factor in the probability of ENTER not executing in preceding time intervals.

P(ENTER execute in [5,6] | LEAVE executed first in [4,5])

= P(ENTER did not execute in [4,5] out of [4,8])*P(ENTER in [5,6] out of [5,8])

$$= \left(1-0.14286\right)*\frac{0.2}{(3*0.2)} \cong 0.28571$$

P(ENTER execute in [6,7] | LEAVE executed first in [4,5])

= P(ENTER did not execute in [4,5] out of [4,8] and [5,6] out of [5,8])*P(ENTER in [6,7] out of [6,8])

$$= \left(1-0.14286\right)*\left[1-\frac{0.2}{(3*0.2)}\right]*\frac{0.2}{(2*0.2)} \cong 0.28571$$

P(ENTER execute in [7,8] |LEAVE executed first in [4,5])

= P(ENTER did not execute in [4,5] out of [4,8], [5,6] out of [5,8] and [6,7] out of [6,8])*P(ENTER in [7,8] out of [7,8])

$$= (1-0.14286)*\left[1-\frac{0.2}{(3*0.2)}\right]*\left[1-\frac{0.2}{(2*0.2)}\right]*1 \cong 0.28571$$

P(ENTER execute in [5,6] |LEAVE execute first in [5,6])

= P(ENTER in [5,6] out of [5,8])

$$= \frac{{}^{1}\!/_{2}*0.2}{\left({}^{1}\!/_{2}*0.2+2*0.2\right)} = 0.2$$

P(ENTER execute in [6,7] / LEAVE execute first in [5,6])

= P(ENTER did not execute in [5,6] out of [5,8])* P(ENTER in [6,7] out of [6,8])

$$= (1-0.2)*\frac{0.2}{2*0.2} = 0.4$$

P(ENTER execute in [7,8] / LEAVE execute first in [5,6])

= P(ENTER did not execute in [5,6] out of [5,8] and [6,7] out of [6,8])* P(ENTER in [7,8] out of [7,8])

$$= (1-0.2)*\left[1-\frac{0.2}{2*0.2}\right]*1 = 0.4$$

The result so far is shown in Figure 10.

P{ENTER [4,8] execute second in interval (i,j) / LEAVE [4,6] execute first in (p,q)}

	(i,j)				
(p,q)	**(4,5)**	**(5,6)**	**(6,7)**	**(7,8)**	**TOTAL**
(4,5)	0.14286	0.28571	0.28571	0.28571	**1**
(5,6)	0	0.2	0.4	0.4	**1**

Figure 10. Probability of occurrence for ENTER[4,8] given that LEAVE[4,6] was executed

With all the information we have so far, we can now calculate the probability of ENTER execute next in [4,5], [5,6], [6,7] and [7,8]. The probability distribution of ENTER [4,8] is given in Figure 11.

P(ENTER execute in [4,5])

= P(ENTER execute in [4,5] / LEAVE execute first in [4,5]) * P(LEAVE execute first in [4,5])

$$= 0.142857 * \frac{0.35}{(0.35 + 0.25)} \cong 0.08333$$

P(ENTER execute in [5,6])

= {P(ENTER execute in [5,6] / LEAVE execute first in [4,5]) * P(LEAVE execute first in [4,5])} + {P(ENTER execute in [5,6] / LEAVE execute first in [5,6]) * P(LEAVE execute first in [5,6])}

$$= \left(0.28571 * \frac{0.35}{(0.35 + 0.25)}\right) + \left(0.2 * \frac{0.25}{(0.35 + 0.25)}\right) \cong 0.25$$

P(ENTER execute in [6,7])

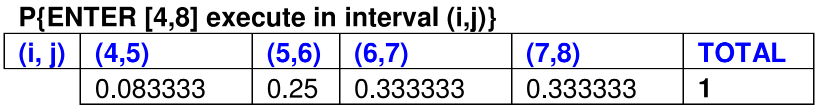
= {P(ENTER execute in [6,7] / LEAVE execute first in [4,5]) * P(LEAVE execute first in [4,5])} + {P(ENTER execute in [6,7] / LEAVE execute first in [5,6]) * P(LEAVE execute first in [5,6])}

$$= \left(0.28571 * \frac{0.35}{(0.35 + 0.25)}\right) + \left(0.4 * \frac{0.25}{(0.35 + 0.25)}\right) \cong 0.3333$$

P(ENTER execute in [7,8])

= {P(ENTER execute 2nd in [6,7] / LEAVE execute first in [4,5]) * P(LEAVE execute first in [4,5])} + {P(ENTER execute 2nd in [7,8] / LEAVE execute first in [5,6]) * P(LEAVE execute first in [5,6])}

$$= \left(0.285714 * \frac{0.35}{(0.35+0.25)}\right) + \left(0.4 * \frac{0.25}{(0.35+0.25)}\right) \cong 0.3333$$



P{ENTER [4,8] execute in interval (i,j)}

(i, j)	(4,5)	(5,6)	(6,7)	(7,8)	TOTAL
	0.083333	0.25	0.333333	0.333333	**1**

Figure 11. Probability of occurrence for ENTER[4,8]

4.4 Probability Distribution for a New Event

4.4.1 Generating New Probability Distribution

Let's assume that the simulation proceeds with the current simulation clock at [4,8], ENTER[4,8] has just executed. As soon as ENTER [4,8] has executed, it schedules a new event START to happen immediately since the server is available there is no other customer waiting for the server. START[4,8] could execute immediately in [4,8] with the same probability distribution as ENTER[4,8] (refer to node 24 in Figure 6). A new event ENTER is also scheduled by ENTER[4,8] to happen [3,8] time units later, which leads to ENTER[7,16]. START[4,8] schedules a LEAVE event to execute [4,6] time units after time interval [4,8], which then leads to LEAVE[8,14]. Events ENTER[7,16] (refer to node 25 in Figure 12) and LEAVE[8,14] (refer to node 31 in Figure 12) are currently on the future event calendar. Because their execution times overlap, they create a non-deterministically ordered set (NOS).

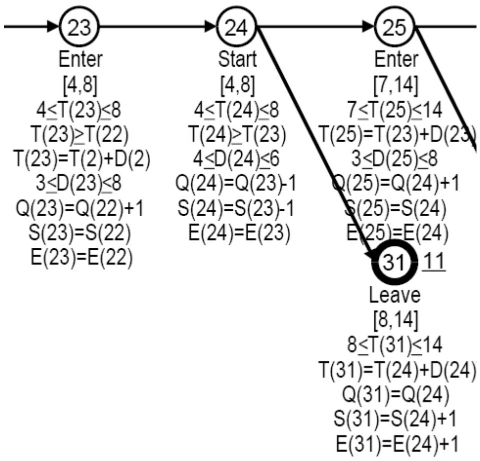


Figure 12. Events ENTER[7,14] and LEAVE[8,14]

ENTER[7,16] is scheduled to occur [3,8] time units after [4,8], which leads to time interval [7,16]. The delay of [3,8] is assumed to follow uniform distribution, but time interval [4,8] from ENTER[4,8] is not uniform. The probability distribution for ENTER[7,16] can be obtained by adding the distribution of ENTER[4,8] to the distribution of Uniform[3,8].

	ENTER[4,8]		**UNIFORM[3,8]**
[4,5]	0.083333333	[3,4]	0.2
[5,6]	0.25	[4,5]	0.2
[6,7]	0.333333333	[5,6]	0.2
[7,8]	0.333333333	[6,7]	0.2
		[7,8]	0.2

Table 1. Probability distributions for events ENTER[4,8] and Uniform[3,8]

Table 1 shows the probability distribution for ENTER[4,8] and the uniform distribution [3,8]. Adding these two distributions is equivalent to the result of drawing one sample from each distribution and add the values of the samples together to get the final value, *x*. The addition of ENTER[4,8] and Uniform[3,8] probability distributions creates the probability distribution for this final value, *x*.

To get the probability distribution of ENTER[4,8]+Uniform[3,8], we could simply multiply the ENTER[4,8] column (refer to Table 1) with the Uniform column. For example, P(E[4,5] + U[3,4]) = P(ENTER[7,9]) = 0.083333333* 0.2 ≅ 0.016666.

Table 2 shows the resulting probability distribution. E[4,5]+U column holds the probability values for ENTER[7,9], [8,10], [9,11], [10,12], and [11,13], which are the results of adding the probability of ENTER[4,5] with Uniform probability of [3,4], [4,5], [5,6], [6,7] and [7,8], respectively.

In order to get the probability of ENTER executing in a certain interval, the sum of probabilities attributed to that interval is computed. For example, P(ENTER[8,10]) = { P(E[4,5]+U[4,5])=[8,10]) } + { P(E[5,6]+U[3,4])=[8,10]) } ≅ 0.016666667 + 0.05 ≅ 0.066666667

E[4,5] +U	P{E[4,5]+U}	E[5,6]+U	P{E[5,6]+U}	E[6,7]+U	P{E[6,7]+U}	E[7,8]+U	P{E[7,8]+U}
[7,9]	0.016666667	[8,10]	0.05	[9,11]	0.06666667	[10,12]	0.06666667
[8,10]	0.016666667	[9,11]	0.05	[10,12]	0.06666667	[11,13]	0.06666667
[9,11]	0.016666667	[10,12]	0.05	[11,13]	0.06666667	[12,14]	0.06666667
[10,12]	0.016666667	[11,13]	0.05	[12,14]	0.06666667	[13,15]	0.06666667
[11,13]	0.016666667	[12,14]	0.05	[13,15]	0.06666667	[14,16]	0.06666667

Table 2. Probability distribution of ENTER[7,16]

The resulting interval will be of length 2, which are [7,9], [8,10], [9,11], [10,12] and [11,13] with the same probability of 0.016666 , respectively. The researcher chooses to work with the same interval length by reducing the interval length to one time unit. If we leave the interval length as it is, as the simulation continues the length of the interval will keep increasing every time we add probability distributions. However, if we reduce the interval to one time unit by assuming that the each of these resulting intervals of [7,9], [8,10], [9,11], [10,12] and [11,13] follows a uniform distribution. If we assume that the probability of ENTER[8,10] which equals to 0.06666 is evenly distributed throughout [8,10], then ENTER could be executed in [8,9] and [9,10] each with the probability of $\frac{0.06666666}{2} \cong 0.0333333$. The same assumption is applied to all other intervals, including ENTER[7,9] (which has a probability of 0.0166666), then we could find the probability of ENTER[8,9] by adding the half of the probability from [8,10] and half of the probability from [7,9], which is shown as below.

P(ENTER[8,9])

$$\cong \frac{0.066666667}{2} + \frac{0.016666667}{2} \cong 0.03333333 + 0.00833333 \cong 0.04166666$$

Table 3 shows the probability distribution of ENTER[7,16], after the reduction of the interval length to one time unit.

ENTER[7,16]	Probability
[7,8]	0.008333333
[8,9]	0.041666667
[9,10]	0.1
[10,11]	0.166666667
[11,12]	0.2
[12,13]	0.191666667
[13,14]	0.158333333
[14,15]	0.1
[15,16]	0.033333333

Table 3. Probability distribution for ENTER[7,16] with interval width of 1 time units

The probability distribution of LEAVE[8,14] can be obtained by multiplying the START[4,8] column with the Uniform distribution of [4,6] (i.e. the delay distribution). Figure 13 shows the multiplication of these two columns, the addition of the probabilities with the same time interval and finally the probability of LEAVE[8,14] after the reduction of time interval to length one.

LEAVE [8,14]

	START[4,8]		Uniform
[4,5]	0.083333	[4,5]	0.5
[5,6]	0.25	[5,6]	0.5
[6,7]	0.333333		
[7,8]	0.333333		

E[4,5] +U	P{E[4,5] +U}	E[5,6] +U	P{E[5,6] +U}	E[6,7] +U	P{E[6,7] +U}	E[7,8] +U	P{E[7,8] +U}
[8,10]	0.041666	[9,11]	0.125	[10,12]	0.166666	[11,13]	0.166666
[9,11]	0.041666	[10,12]	0.125	[11,13]	0.166666	[12,14]	0.166666

Interval	Probability
[8,10]	0.041666
[9,11]	0.166666
[10,12]	0.291666
[11,13]	0.333333
[12,14]	0.166666
SUM	**1**

Interval	Probability
[8,9]	0.02083
[9,10]	0.10416
[10,11]	0.22916
[11,12]	0.3125
[12,13]	0.25
[13,14]	0.08333
SUM	**1**

Figure 13. Probability distribution of LEAVE[8,14]

4.4.2 Generating Conditions for the Conditional Probability

The probability distributions that are generated in Section 4.4.1 carry important information about the probability of occurrence for the newly scheduled events ENTER[7,16] and LEAVE[8,14]. It assumes that ENTER[7,16] and LEAVE[8,14] are independent of all previous events. This assumption is not valid when it comes to examining the probability of occurrence for sequential events. ENTER[7,16] and LEAVE[8,14] are exclusive (i.e. disjoint) events, which capture the intuition of non-compatible outcomes. Non-compatible events cannot happen at the same time. This is not the same as independent outcomes. If *A*, *B* are disjoint and we know that *A* occurred, then we do know a lot about *B*. Namely we know that *B* cannot occur. Thus there is an interaction between events *A* and *B*. Knowing whether *A* occurred influences the probability of *B*, which is not possible under independence. Independence captures the intuition of non-interaction, and lack of information. In modeling it is often assumed rather than verified.

The probability of ENTER[7,16] executing in a specific time interval (i.e. [7,8],…, [15,16]) depends on the condition of two things:

1. The time interval for which ENTER[7,16]'s scheduling event is executed.

If given that its scheduling event ENTER[4,8] is executed in [4,5], then ENTER[7,16] can execute in [7,9], [8,10], [9,11], [10,12] and [11,13] with the probability of 0.2, respectively. This distribution with overlapping time interval represents the conditional probability distribution of ENTER[7,11], given the time interval of ENTER[4,5]. Note that it has the same shape as the delay distribution of [3,8]. Recall that the delay distribution of [3,8] has a uniform probability distribution of 0.2 in each of its one-unit time interval.

We know from 4.4.1 that P(ENTER[7,16] is in [7,9])

= P(E[4,5] + U[3,4]) = P(ENTER[7,9]) $\cong$ 0.083333333* 0.2 = 0.016666…

And, P(ENTER[4,8] is in [4,5]) $\cong$ 0.083333333

Since P(ENTER[7,16] is in [7,9])

= P([ENTER[7,16] is in [7,9]/ENTER[4,8] is in [4,5])* P(ENTER[4,8] is in [4,5])

Thus, P([ENTER[7,16] is in [7,9]/ENTER[4,8] is in [4,5])

= P(ENTER[7,16] is in [7,9])/ P(ENTER[4,8] is in [4,5])

$\cong$ (0.083333333* 0.2)/ 0.083333333

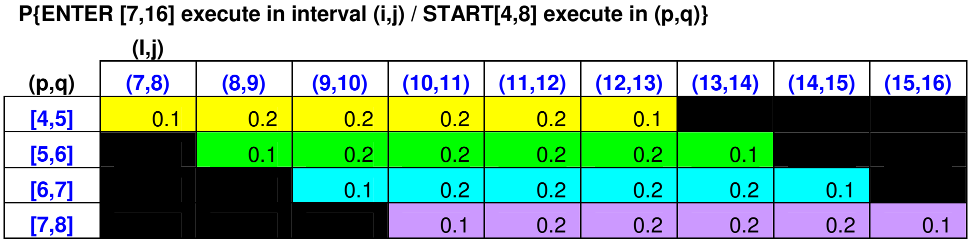
= 0.2

If we assume that the probability of all the two-unit time intervals are evenly distributed within their time intervals, using the same interval probability reduction method that was discussed in 4.1.1, the conditional probability distribution of ENTER[7,16] executing in [7,8], [8,9], [9,10], [10,11], [11,12] and [12,13] given that its scheduling event ENTER[4,8] was executed in [4,5] is equal to 0.1, 0.2, 0.2, 0.2, 0.2 and 0.1 respectively.

2. Whether ENTER[7,16] intersects with its immediate preceding event.

The conditional probability distribution that was obtained in part 1 only takes into account the effect of ENTER[4,8]'s execution time on the probability of occurrence for ENTER[7,16]. If the immediate preceding event for ENTER[7,16] is ENTER[4,8], there will not be any intersection of time intervals when calculating the conditional probability. Clearly, the conditional probability for ENTER[7,16] will be less in the time intervals that have intersection with the immediate preceding event. This is likely to be the case if ENTER[7,16] is one of the NOS event in the NOS set.

Even though the immediate preceding event for ENTER[7,16] is not its scheduling event, ENTER[4,8], but START[4,8] has the exact probability distribution as ENTER[4,8]. This means that there will not be any intersection of time intervals when calculating the conditional probability for ENTER[7,16]. Thus, the conditional probability for ENTER[7,16] that is calculated in part 1 will not be affected and could be used for further analysis. Figure 14 shows the conditional probability for ENTER[7,16].



P{ENTER [7,16] execute in interval (i,j) / START[4,8] execute in (p,q)}

	(I,j)								
(p,q)	(7,8)	(8,9)	(9,10)	(10,11)	(11,12)	(12,13)	(13,14)	(14,15)	(15,16)
[4,5]	0.1	0.2	0.2	0.2	0.2	0.1			
[5,6]		0.1	0.2	0.2	0.2	0.2	0.1		
[6,7]			0.1	0.2	0.2	0.2	0.2	0.1	
[7,8]				0.1	0.2	0.2	0.2	0.2	0.1

Figure 14. Conditional probability for ENTER[7,16]

If we were just interested in finding the probability of a thread occurring from the QSGM output, it is sufficient to use the conditional probability for ENTER[7,16] in Figure 14. However, it is not sufficient to use it if we want to determine output statistics for the QSGM output. This is because the conditional probability for ENTER[7,16] in Figure 14 only carries information all the way up to its scheduling event. Important information about any preceding events before ENTER[7,16]'s scheduling event is not included. If we were to stop the simulation at this point, we could calculate the probability of the thread up to ENTER[7,16], but we could not accurately calculate statistics such as the server utilization.

From Figure 14, we know that from [4,5] to [7,8], the server utilization is 1 as the server is busy during that time and the conditional probability of this event is 0.1. But we could not determine the server utilization for the events before that, even though we know that the server is busy from [0,0] (at the start of event START[0,0]) to [4,6] (at the start of event LEAVE[4,6]). An average value for the server utilization could be estimated but as the simulation continues, the estimated average value will not be accurate.

Another way to getting the information that we need in order to calculate statistics is to generate all possible conditions for the conditional probability. Any events that are to be executed after the first non-zero execution time event will carry information about the timing of all preceding events. Each combination of the timing of these preceding events that are generated will become the condition for the conditional probability. The conditions that are generated for the conditional probability of ENTER[7,16] are shown in the left column of Table 4. The number 5 of condition "**5,4**" represents the time interval [5,6] of the immediate preceding event, START[4,8] and ENTER[4,8] since they both have exactly the same distribution and if START is executed in [4,5], ENTER will also be executed in [4,5]. The number 4 of condition "**5,4**" represents the time interval [4,5] of the second last event, which is LEAVE[4,6]. The same conditions are generated for LEAVE[8,14]. The conditional probability for LEAVE[8,16] is shown in Table 5.

Conditional Probability for ENTER [7,16]

Condition	[7,8]	[8,9]	[9,10]	[10,11]	[11,12]	[12,13]	[13,14]	[14,15]	[15,16]
4,4	0.1	0.2	0.2	0.2	0.2	0.1			
5,4		0.1	0.2	0.2	0.2	0.2	0.1		
6,4			0.1	0.2	0.2	0.2	0.2	0.1	
7,4				0.1	0.2	0.2	0.2	0.2	0.1
5,5		0.1	0.2	0.2	0.2	0.2	0.1		
6,5			0.1	0.2	0.2	0.2	0.2	0.1	
7,5				0.1	0.2	0.2	0.2	0.2	0.1

Table 4. Conditional probability for ENTER[7,16] with all possible combination of conditions

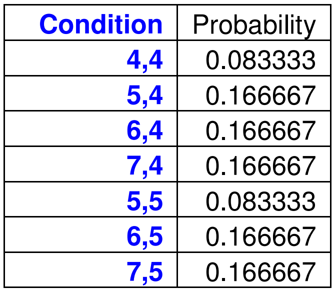
Conditional Probability for Leave [8,14]

Condition	[8,9]	[9,10]	[10,11]	[11,12]	[12,13]	[13,14]
4,4	0.25	0.5	0.25			
5,4		0.25	0.5	0.25		
6,4			0.25	0.5	0.25	
7,4				0.25	0.5	0.25
5,5		0.25	0.5	0.25		
6,5			0.25	0.5	0.25	
7,5				0.25	0.5	0.25

Table 5. Conditional probability for LEAVE[8,14] with all possible combination of conditions

As can be seen from Table 4 and Table 5, the conditional probability for ENTER[7,16] and LEAVE[8,14] is not affected by what happens before its scheduling event, ENTER[4,8]. The delay distribution and scheduling event's distribution determine the timing of the newly scheduled event and its probability distribution. The probability of occurrence that is obtained for the new event will be less in the time intervals where it intersects with its immediate preceding event. If the immediate preceding event for the new event happens to be its scheduling event, there will not be any intersection of time intervals. In this case, the probability of occurrence that is obtained from the delay distribution and scheduling event's distribution will remain the same.

The probability of each condition occurring could be obtained by normalizing the conditional probability in Figure 10, which is shown in Table 6. The probability of each condition occurring is needed when calculating the probability of a NOS event executing first.



Condition	Probability
4,4	0.083333
5,4	0.166667
6,4	0.166667
7,4	0.166667
5,5	0.083333
6,5	0.166667
7,5	0.166667

Table 6. Probability of the conditions that are generated

Basically, the method for calculating the probability of each NOS event executing first is the same. Instead of using the probability distribution that is computed from the uniform delay distribution and the scheduling event's probability distribution, for each condition that is generated, the conditional probability of each NOS event occurring in a given time interval is used in calculating the probability of which event executing first.

So, the algorithm that determines the probability of which event executing first in a given time interval would be constructed in a way that it is repeatable for each condition that is generated and for each defined time interval.

The probability of ENTER[7,14] executing first and LEAVE[8,14] executing first are computed and are given in Table 7 and Table 8, respectively. ENTER[7,14] has a probability of 0.00833 to execute first in the time interval [7,8], 0.03958 to execute first in [8,9], 0.08542 to execute first in [9,10] and so on. These probabilities are obtained from summing the product of the conditional probability in Table 6 with the conditional probability of ENTER[7,14] in each time interval. Combining the probabilities from all these time intervals, ENTER[7,14] has a probability of 0.4 to execute first in [7,14]. On the other hand, LEAVE[8,14] has a probability of 0.6 to execute first in [8,14]. Notice that the addition of ENTER[7,14] and LEAVE[8,14]'s probability equals to one, as they are exclusive events.

ENTER[7,14]	7	8	9	10	11	12	13	Node 25
4,4	0.1	0.175	0.1	0.025				
5,4		0.1	0.175	0.1	0.025			
6,4			0.1	0.175	0.1	0.025		
7,4				0.1	0.175	0.1	0.025	
5,5		0.1	0.175	0.1	0.025			
6,5			0.1	0.175	0.1	0.025		
7,5				0.1	0.175	0.1	0.025	
P(Execute 1st)	0.00833	0.03958	0.08542	0.11875	0.09792	0.04167	0.00833	**0.4**
Normalized p	0.02083	0.09896	0.21354	0.29688	0.24479	0.10417	0.02083	**1**

Table 7. Probability of ENTER[7,14] executing first

LEAVE[8,14]	**8**	**9**	**10**	**11**	**12**	**13**	NODE 31
4,4	0.2	0.3	0.1				
5,4		0.2	0.3	0.1			
6,4			0.2	0.3	0.1		
7,4				0.2	0.3	0.1	
5,5		0.2	0.3	0.1			
6,5			0.2	0.3	0.1		
7,5				0.2	0.3	0.1	
P(Execute 1st)	0.016667	0.075	0.15	0.19167	0.13333	0.03333	**0.6**
Normalized p	0.027778	0.125	0.25	0.31944	0.22222	0.05556	**1**

Table 8. Probability of LEAVE[8,14] executing first

4.5 Probability Distribution for the Timing of an Event

In the process of calculating the probability of event occurrence, for the case when there is more than one event in the NOS set, the probability distribution for the timing of an event can be obtained from normalizing the probability of an NOS event executing first. For the case when there is one event in the NOS set, the probability distribution for the timing of an event is already in the normalized form.

The "Normalized p" rows in Table 7 and Table 8 are the probability distributions for the timing of ENTER[7,14] and LEAVE[8,14], respectively, as the result of normalization from the probability in the "P(Execute 1st)" row.

4.6 Probability of Thread Occurrence

When attempting to determine a sample space (the complete possible outcomes from an experiment), it is often helpful to draw a diagram which illustrates the paths to all the possible outcomes. One such diagram is a tree diagram. In addition to determining the number of outcomes in a sample space, the tree diagram can be used to determine the probability of individual outcomes within the sample space. The probability of any outcome in the sample space is the product (multiplication) of all possibilities along the path that represents that outcome on the tree diagram. By following the branches of the tree diagram, we can find all the possible outcomes.

The QSGM output (refer to Figure 6) is a tree diagram, since it illustrates the possible outcomes from a QSGM output. Each branch of the tree has its probability of occurrence that has been calculated and the probability distribution of the timing of the event. If there is only one branch at any level, the probability of occurrence for this branch is equal to 1. The sum of all the branches at any level must sum to 1. Figure 15 shows the probability tree diagram for the QSGM output. The bolded number **0.5969** in Figure 15 on the arc emanating from node 6 to node 7 is the probability of occurrence for ENTER[6,12]. The bolded number **0.4031** on the arc emanating from node 6 to node 13 is the probability of occurrence for LEAVE[8,12].

The thread probability for the Simple Queuing Model could be calculated by multiplying the probabilities of occurrence for the events along the path that represents the outcome from the tree diagram. For example, thread 1 has a probability of 0.2336 (correct to 4 decimal places), which is the product of 0.4 (taken from node 4's probability of occurrence), 0.5969 (taken from node 7's probability of occurrence) and 0.9786(taken from node 8's probability of occurrence). The thread probability for all the possible outcomes of the Simple Queuing model is shown in Table 9.

Thread	Thread Probability
1	0.2336
2	0.0051
4	0.1613
8	0.2400
11	0.3600

Table 9. Thread Probability

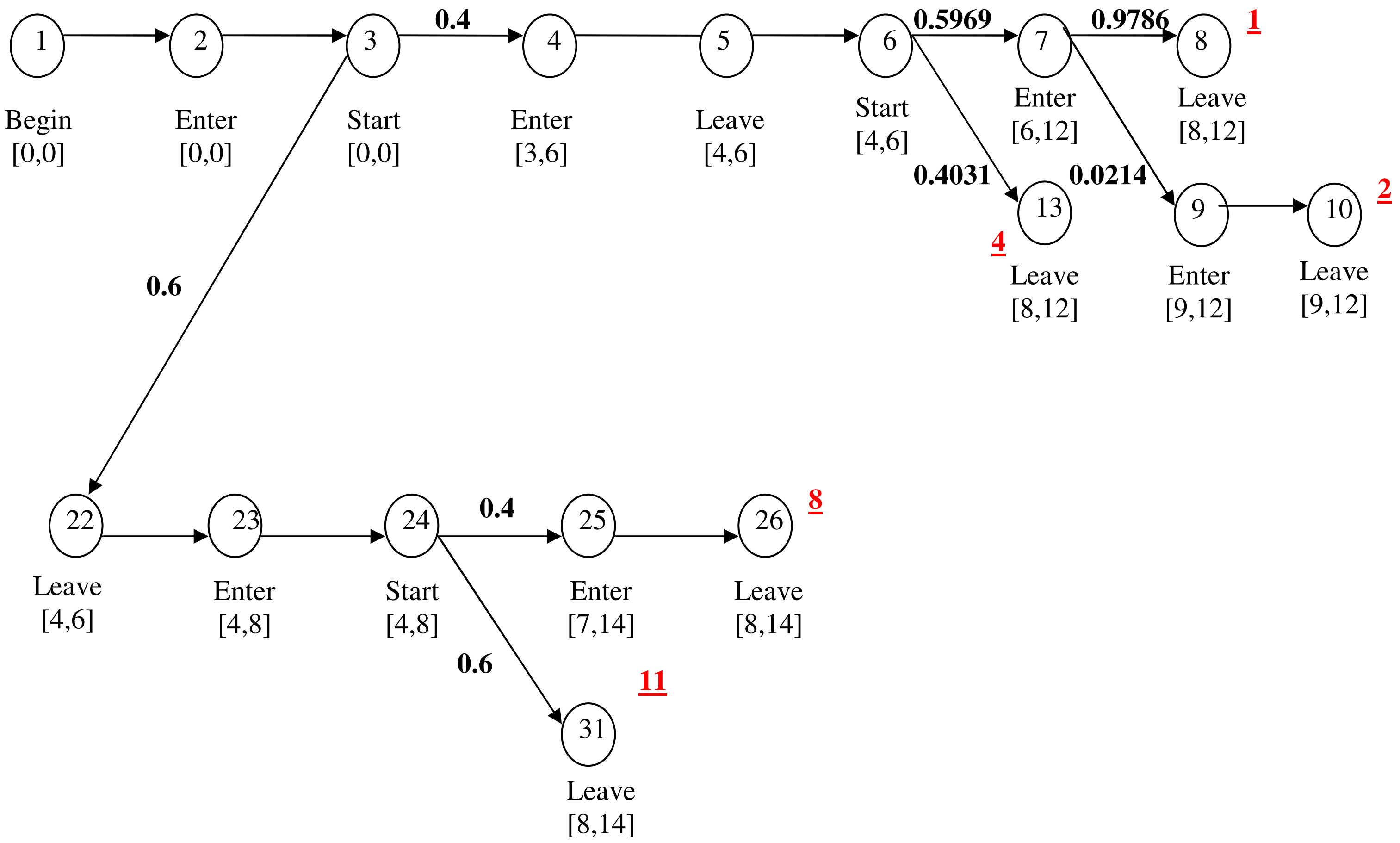


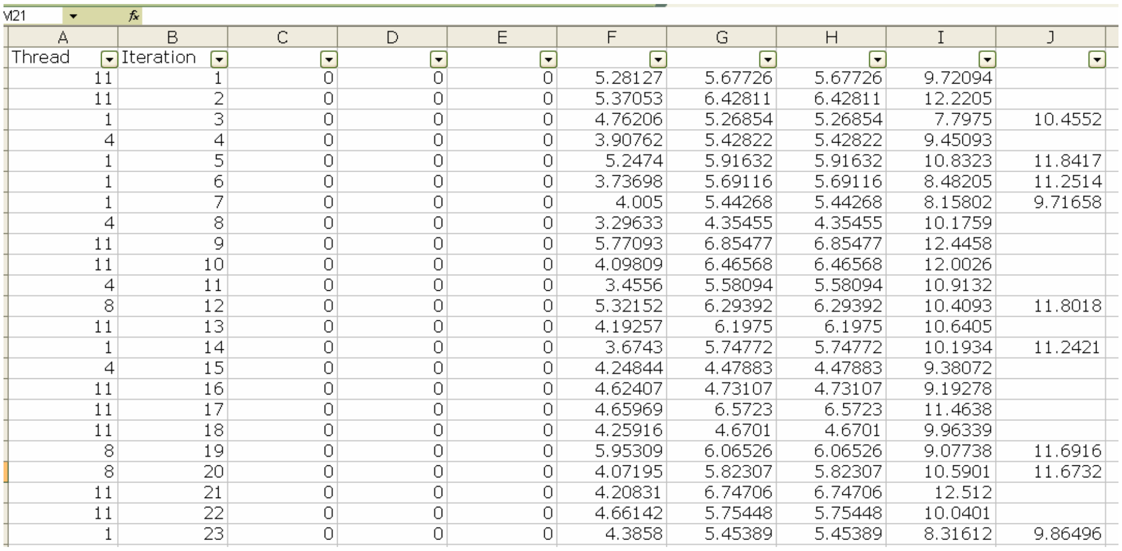
Figure 15. Probability tree diagram

4.7 Comparison of the Manually Computed Probability with Simulated Probability Using Excel

In order to compare the thread probabilities that are calculated using the method discussed in the previous sections, a simulation model of the Simple Queuing System was created using Microsoft Excel. The timing of all the scheduled events is randomly generated (see Figure 16). The event name that represents each of these timings is shown in Figure 17. All the iterations are sorted according to the order of event sequences and are then assigned the appropriate thread number.

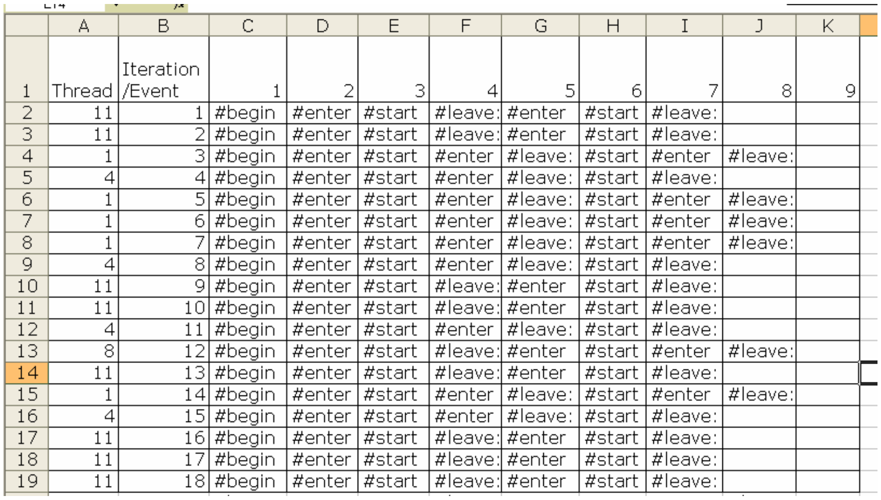
Figure 16 shows a table from the Excel Worksheet that contains the 10,000 generated timing of events for the Simple Queuing System. Each row contains all the timing of events for an iteration (column B) that are then sorted into thread number (column A). The numbers in Column C to J of Figure 16 are the generated timing for the first event, second event and so on. The matching event name for each of these timings of event from each iteration is shown in Figure 17 Column C to J.

The simulated thread probabilities based on the 10,000 output data are computed and tabulated in Table 10. The calculated probability for thread 1 is equal to 0.2336 (correct to four decimal places), compared to the simulated probability for thread of 0.2418 (correct to four decimal places). The difference between the two probabilities is 0.0082, with the percentage difference less than 3.4%.



A	B	C	D	E	F	G	H	I	J
Thread	Iteration								
11	1	0	0	0	5.28127	5.67726	5.67726	9.72094	
11	2	0	0	0	5.37053	6.42811	6.42811	12.2205	
1	3	0	0	0	4.76206	5.26854	5.26854	7.7975	10.4552
4	4	0	0	0	3.90762	5.42822	5.42822	9.45093	
1	5	0	0	0	5.2474	5.91632	5.91632	10.8323	11.8417
1	6	0	0	0	3.73698	5.69116	5.69116	8.48205	11.2514
1	7	0	0	0	4.005	5.44268	5.44268	8.15802	9.71658
4	8	0	0	0	3.29633	4.35455	4.35455	10.1759	
11	9	0	0	0	5.77093	6.85477	6.85477	12.4458	
11	10	0	0	0	4.09809	6.46568	6.46568	12.0026	
4	11	0	0	0	3.4556	5.58094	5.58094	10.9132	
8	12	0	0	0	5.32152	6.29392	6.29392	10.4093	11.8018
11	13	0	0	0	4.19257	6.1975	6.1975	10.6405	
1	14	0	0	0	3.6743	5.74772	5.74772	10.1934	11.2421
4	15	0	0	0	4.24844	4.47883	4.47883	9.38072	
11	16	0	0	0	4.62407	4.73107	4.73107	9.19278	
11	17	0	0	0	4.65969	6.5723	6.5723	11.4638	
11	18	0	0	0	4.25916	4.6701	4.6701	9.96339	
8	19	0	0	0	5.95309	6.06526	6.06526	9.07738	11.6916
8	20	0	0	0	4.07195	5.82307	5.82307	10.5901	11.6732
11	21	0	0	0	4.20831	6.74706	6.74706	12.512	
11	22	0	0	0	4.66142	5.75448	5.75448	10.0401	
1	23	0	0	0	4.3858	5.45389	5.45389	8.31612	9.86496

Figure 16. The generated timing of events for the Simple Queuing System



	A	B	C	D	E	F	G	H	I	J	K
1	Thread	Iteration /Event	1	2	3	4	5	6	7	8	9
2	11	1	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
3	11	2	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
4	1	3	#begin	#enter	#start	#enter	#leave:	#start	#enter	#leave:	
5	4	4	#begin	#enter	#start	#enter	#leave:	#start	#leave:		
6	1	5	#begin	#enter	#start	#enter	#leave:	#start	#enter	#leave:	
7	1	6	#begin	#enter	#start	#enter	#leave:	#start	#enter	#leave:	
8	1	7	#begin	#enter	#start	#enter	#leave:	#start	#enter	#leave:	
9	4	8	#begin	#enter	#start	#enter	#leave:	#start	#leave:		
10	11	9	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
11	11	10	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
12	4	11	#begin	#enter	#start	#enter	#leave:	#start	#leave:		
13	8	12	#begin	#enter	#start	#leave:	#enter	#start	#enter	#leave:	
14	11	13	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
15	1	14	#begin	#enter	#start	#enter	#leave:	#start	#enter	#leave:	
16	4	15	#begin	#enter	#start	#enter	#leave:	#start	#leave:		
17	11	16	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
18	11	17	#begin	#enter	#start	#leave:	#enter	#start	#leave:		
19	11	18	#begin	#enter	#start	#leave:	#enter	#start	#leave:		

Figure 17. The generated events for the Simple Queuing System

Thread	Calculated Thread Probability	Simulated Thread Probability	Difference
1	0.23363689	0.24176667	0.00812978
2	0.00511311	0.00418333	-0.00092978
4	0.16125000	0.15380000	-0.00745000
8	0.24000000	0.23865000	-0.00135000
11	0.36000000	0.36160000	0.00160000

Table 10. Comparison of the calculated thread probability with simulated thread probability

The Excel simulation model for the Simple Queuing System contains all the required information to plot the probability distribution for the timing of an event execution. The simulated and calculated probability for the execution of LEAVE[4,6] (Node 22) in the interval with the lesser endpoint of 4 and 5 are shown in Table 11. The associated event timing probability distributions are plotted in Figure 18. LEAVE[4,6] is the first non-zero execution time event in thread 8 and 11. Table 11 shows that the calculated probability is very close to the simulated probability for an event that was executed early in the system.

Figure 19, Figure 20, Figure 21 and Figure 22 show the progression of the calculated probability distribution and the simulated probability distribution for the timing of events from thread 8 and 11 as the system approaches the terminating condition of 2 customers exiting the system. Table 12, Table 13, Table 14 and Table 15 are the associated tables which contain the calculated and simulated probability distributions for the events from thread 8 and 11. The tables show that the calculated probabilities are very close to the simulated probabilities as the simulation progresses. The ability of the probability algorithm to capture the shape of the event timing probability distribution as the simulation progresses, even for rare event sequences with low thread probability such as LEAVE[9,12] (Node 10), which is the last event to execute in thread 2 with thread probability of 0.0051(correct to 4 decimal places) are exhibited in the graphs.

LEAVE[4,6] (Node 22)

Interval	Simulated Probability	Calculated Probability	Difference
[4,5]	0.586982331	0.583333333	-0.003649
[5,6]	0.413017669	0.416666667	0.003649

Table 11. Calculated and Simulated Probability for the timing of LEAVE[4,6] (Node 22)

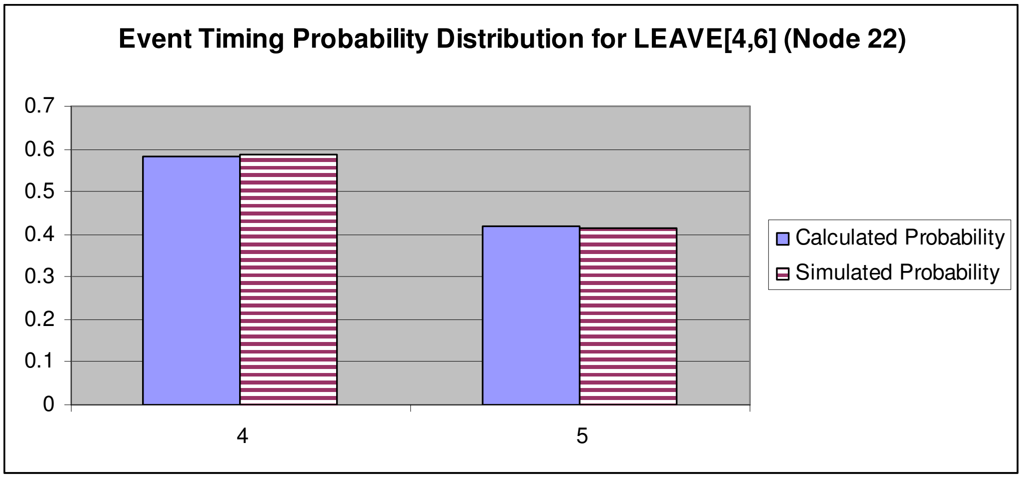


Figure 18. Event Timing Probability Distribution for LEAVE[4,6] (Node 22)

ENTER[4,8] (node 23)

Interval	Simulated Probability	Calculated Probability	Difference
4	0.078636776	0.083333333	0.00469656
5	0.260213702	0.25	-0.0102137
6	0.330609679	0.333333333	0.00272365
7	0.330539842	0.333333333	0.00279349

Table 12. Calculated and Simulated Probability for the timing of ENTER[4,8] (Node 23)

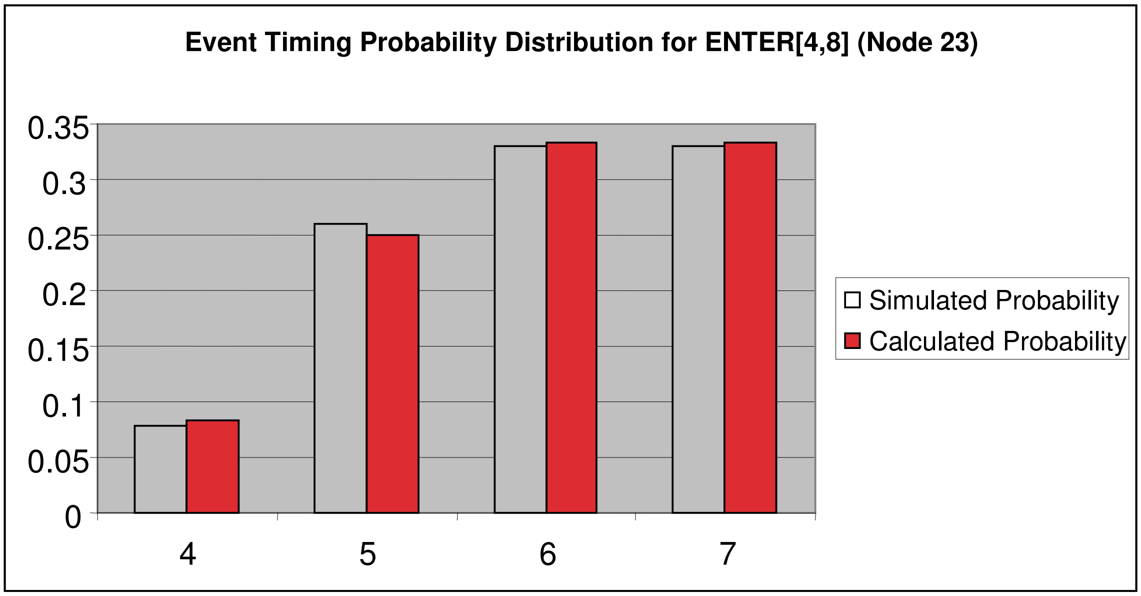


Figure 19. Event Timing Probability Distribution for ENTER[4,8] (Node 23)

ENTER[7,14] (Node 25)

Interval	Simulated Probability	Calculated Probability	Difference
7	0.014316642	0.020833333	0.00651669
8	0.095537398	0.098958333	0.00342094
9	0.222431734	0.213541667	-0.0088901
10	0.309448984	0.296875	-0.012574
11	0.247922341	0.244791667	-0.0031307
12	0.097073818	0.104166667	0.00709285
13	0.013269083	0.020833333	0.00756425

Table 13. Calculated and Simulated Probability for the timing of ENTER[7,14] (Node 25)

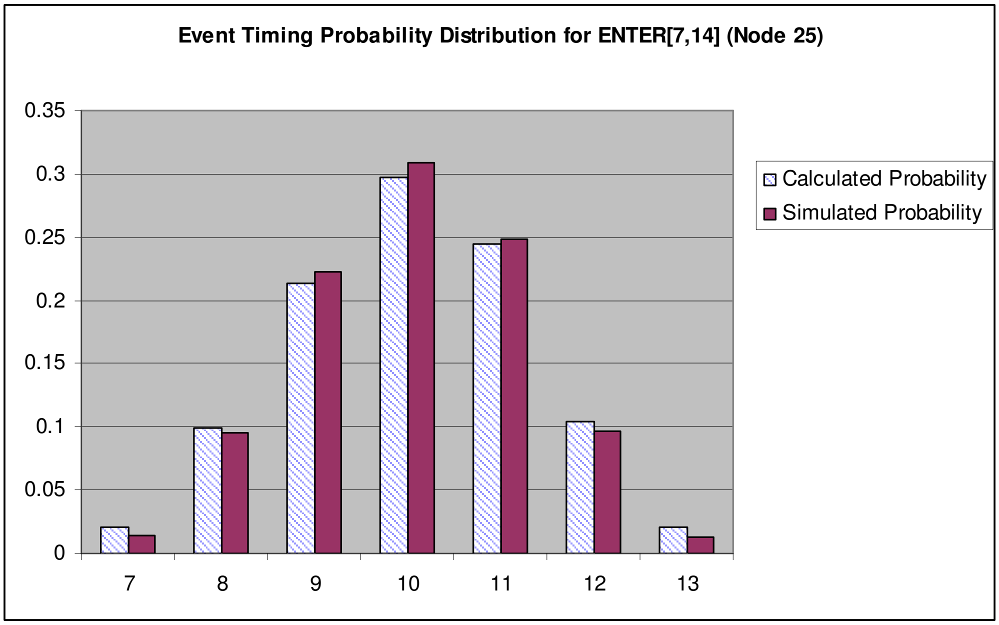


Figure 20. . Event Timing Probability Distribution for ENTER[7,14] (Node 25)

LEAVE[8,14] (Node 26)

Interval	Simulated Probability	Calculated Probability	Difference
8	0.009009009	0.010416667	0.00140766
9	0.073538655	0.072916667	-0.000622
10	0.21509882	0.197916667	-0.0171822
11	0.316781898	0.302083333	-0.0146986
12	0.277184161	0.291666667	0.01448251
13	0.108387457	0.125	0.01661254

Table 14. Calculated and Simulated Probability for the timing o LEAVE[8,14] (Node 26)

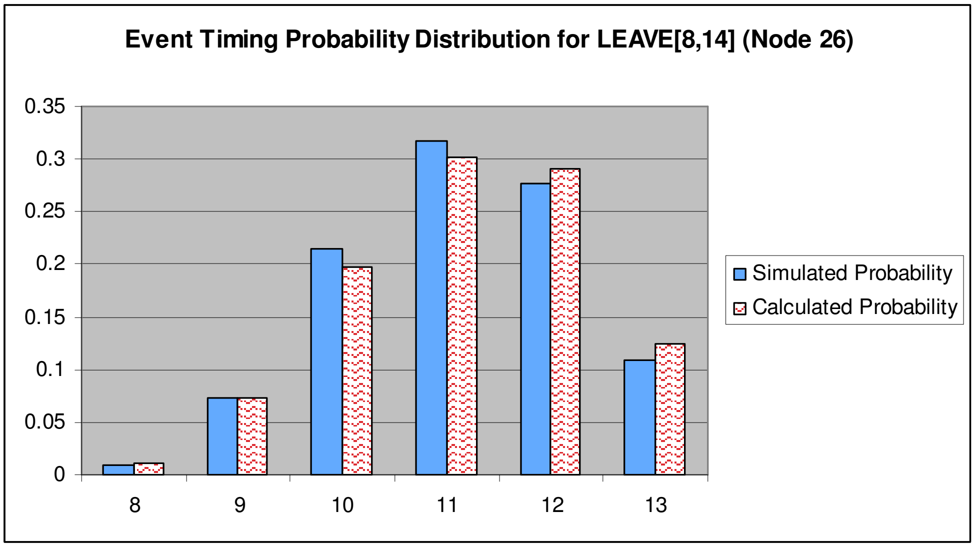


Figure 21. Event Timing Probability Distribution for LEAVE[8,14] (Node 26)

LEAVE[9,14] (Node 10)

Interval	Simulated Probability	Calculated Probability	Difference
9	0.015936255	0.0671685	0.05123225
10	0.466135458	0.435147655	-0.0309878
11	0.517928287	0.497683845	-0.0202444

Table 15. Calculated and Simulated Probability for the timing of LEAVE[9,14] (Node 10)

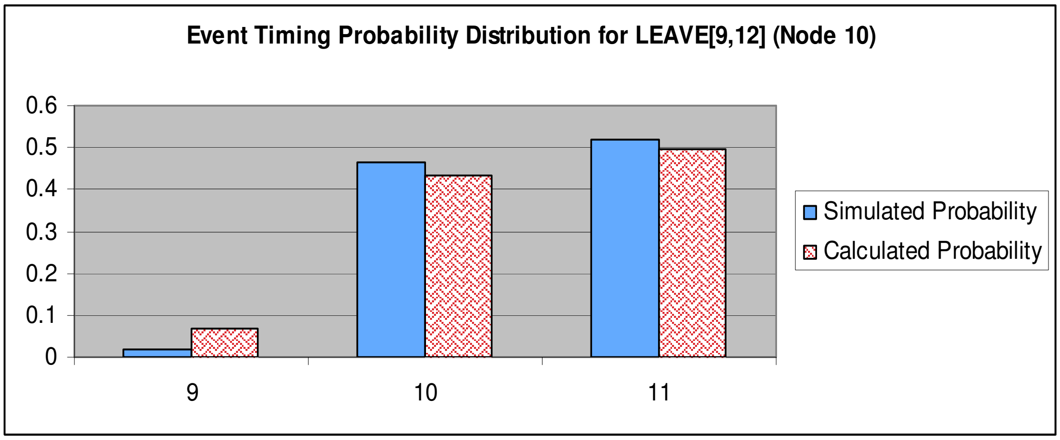


Figure 22. Event Timing Probability Distribution for LEAVE[9,12] (Node 10)

4.8 Development of Probability Algorithm Using QDES Framework

4.8.1 QDES Framework

The simulation graph model (SGM) modified by Ingalls (1996, 2001) provides a general framework in defining and developing QDES. The general framework and the algorithm have been briefly introduced in Section 2.4 and 2.7. In this section, further details of QDES framework will be discussed. Let simulation graph, $G\ (V(G),E(G),\Psi_G)$ be a directed graph where $V(G)$ is the set of vertices of G, E(G) is the set of edges and Ψ_G is an incidence function that associates with each edge of G. The Simulation Graph Model (SGM) is then defined as $S = (F,C,X,T,\Gamma,G)$ where

$F = \{f_v\colon v \in V(G)\}$, the set of state transitions functions associated with vertex v

$C = \{C_e\colon e \in E(G)\ \}$, the set of scheduling edge conditions

$X = \{X_e\colon e \in E(G)\ \}$, the set of execution edge conditions

$T = \{t_e :e \in E_S\ (G)\}$, the set of edge delay times as time intervals, and

$\Gamma = \{\gamma_e\ e \in E_S\ (G)\}$, the set of event execution priorities

The global simulation clock and the event calendar will be stored as time interval. The symbol τ is used to represent the global simulation clock, while the event calendar will be represented by the capital letter, L. The definition of L is an ordered set, L $=\{\ (x_1,\gamma_1,v_1,e_1,a_1,b_1),(x_2,\gamma_2,v_2,e_2,a_2,b_2),...\ \}$ where x_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, a_i are the values of the edge attributes, and b_i are the times at which events are scheduled for the i^{th} event notice.

According to Ingalls, et al. (1996), the following sets are defined:

S_v :the set of state variables that can be modified at vertex v (since S is the set of all state variables, $S_v \subset S$);

P_v :the set of state variables that can be modified at vertex v;

Φ_e : the set of state variables involved in the scheduling conditions on edge *e*;

ϑ_e : the set of state variables involved in the execution conditions on edge *e*;

A_e : the set of state variables used to determine the values of a_e on edge *e*;

H : the set of saved states, which is used to iterate through all possible states in the simulation

N_h : the NOS or the set of possible next events

In addition to these sets, ω is used to represent the termination conditions for the simulation. Two variables, h and n_h are also defined. The variable h is used to count the number of saved states while iterating through the set *H*. The variable n_h is used to iterate through the set N_h.

4.8.2 Execution of the QDES Algorithm

With the definitions defined in Section 4.8.1, the execution of the QDES algorithm is carried out as follows.

Run Initialization
Initialize the saved state set and counter. $H = \emptyset, h \leftarrow 1$
New Process Initialization
Step 1. Initialize the global simulation clock. $\tau \leftarrow [0,0]$.
Step 2. Insert one event notice into the event calendar: For simplicity, we will assume that the event notice could be executed at time [0,0] where $L = L \cup \{([0,0], x_1.\gamma_1, e_1, a_1, b_1), ([0,0], x_2.\gamma_2, e_2, a_2, b_2), ...\}$
Execute (execution of the model implementation)
Step 1. Determine the NOS, the set Q is the set of all event notices that could be executed next without regard to priority.

$Q = \{l \mid x_l^- \leq (\min(x_l^+)\ \forall\ l \in L)\ \forall\ l \in L\}$

$N_h = \{q \mid \gamma_q = (\min(\gamma_q) \forall q \in Q) \forall q \in Q\}$

Step 2. If $|\boldsymbol{N_h}| = 1$, then go to Step 6 of Execute.

Step 3. Initialize the variable to loop through the NOS. $n_h \leftarrow 1$

Step 4. Save the state of the simulation by saving the state information in the save-state stack and incrementing the save-state counter. $H_h = \{S, L\}$. $h \leftarrow h + 1$.

Step 5. Remove the $(N_{h-1})_{n_{h-1}}$ event notice from L. $L = L \backslash \{l | l = (N_{h-1})_{n_{h-1}}\}$. Event notice l is removed from the calendar. Go to Step 7 of Execute.

Step 6. Remove the first event notice from L. $L = L \backslash \{l | l = (x_1, \gamma_1, e_1, a_1, b_1, z_l)\}$. Event notice l is removed from the calendar.

Step 7. Evaluate the execution edge condition, $X_{e_l}(\vartheta_{e_l}, a_l)$. If $X_{e_l}(\vartheta_{e_l}, a_l) = FALSE$ then go to Step 15 of Execute, else go to Step 8 of Execute.

Step 8. Determine the possible new simulation clock time. $\tau' \leftarrow [\max(x_l^-, \tau^-), \min(x_l^+, \forall\ l \in L)]$.

Step 9. If t_{e_l} is a constant delay interval, determine if the constant delay time is still valid. It is still valid if $t_{e_l} = \tau' - b_l$. If t_{e_l} is a constant interval and still valid, or if t_{e_l} is an uncertain interval, go to Step 11 of Execute, else go to Step 10 of Execute.

Step 10. Set $t_{e_l} \leftarrow \tau' - b_l$. If $t_{e_l}^- \leq t_{e_l}^+$, then the new t_{e_l} is valid, but the process must be started at the beginning so that t_{e_l} can be consistent throughout the process. Go to step 1 of New Process Initialization. Otherwise there is no valid constant interval for this process and the process is declared "invalid" and terminates. In that case, go to step 16 of Execute.

Step 11. Update the simulation clock. $\tau \leftarrow \tau'$.

Step 12. Assign the attributes to the parameters of the vertex. $P_{v_l} \leftarrow a_l$. If Y is the i^{th} state variable in the vertex parameter list, i.e. $(P_v)_i = Y$, then $Y \leftarrow (a_l)_i$.

Step 13. Evaluate the state change. $S_{v_1} \leftarrow f_{v_l}(S)$.

Step 14. Schedule further events. For each edge, e_{lj}, emanating from v_l, if $C_{e_{lj}}(\Phi_{e_{lj}}) = TRUE$ then evaluate $A_{e_{lj}}$ and assign the attribute value of the new event notice, k, $a_k \leftarrow A_{e_{lj}}$. Generate the inter-event time, $b_k = f(t_{e_{lj}})$, and schedule the event notice where $L = L \cup \{(\tau + b_k, \gamma_{e_{1j}}, v_j, e_{1j}, a_k, b_k, z,)\}$.

Step 15. If any of the following conditions are satisfied: (i) $\tau > T_{stop}$; (ii) the simulation stopping condition, ω, evaluates TRUE; (iii) L is empty, then the simulation has reached the end of the process. Go to Step 17 of Execute.

Step 16. Go to Step 1 of Execute.

Step 17. If $h = 1$, then terminate the simulation.

Step 18. Restore the last saved system state off the saved-state stack: $h = h - 1,\quad L = (L \mid L \in H_h),\quad S = (S \mid S \in H_h)$.

Step 19. Increment $n_{h-1} \leftarrow n_{h-1} + 1$.

<u>*Step 20.*</u> If $n_{h-1} \leq| N_{h-1} |$ then go to Step 5 of Execute.
<u>*Step 21.*</u> Go to Step 17 of Execute.

4.8.3 Execution of the Probability Algorithm

The QDES framework and algorithm in Section 4.8.1 and 4.8.2 will be used as the general framework for developing the probability algorithm that will compute the probability of an event execution and the probability distribution of the event's execution time.

In order to determine the new scheduled time given the conditions of preceding events that have executed, each event notice in the event calendar will carry a new piece of information that will provide the vertex of the event that scheduled the current event notice. Thus, the new definition of L is an ordered set, *L* $=\{ (x_1, \gamma_1, v_1, e_1, a_1, b_1, z_1), (x_2, \gamma_2, v_2, e_2, a_2, b_2, z_2),... \}$ where x_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, a_i are the values of the edge attributes, b_i are the times at which events are scheduled for the i^{th} event notice, and z_i represents the vertex of the event that scheduled the i^{th} event notice. All executed events will be assigned a vertex number, z and an edge number, y that represents the vertex and the edge that has been executed for an event. The scheduled execution time for an event is denoted as s_l. The user-specified time unit that will be used for the simulation will be denoted as θ. The value of θ for the Simple Queuing Model is equal to 1.

Recall from Section 4.4.2 that in order to calculate statistics, we propose to generate all possible conditions for calculating the conditional probability of an event executing in a given interval. Any event that is to be executed after the first non-zero execution time event will carry information about the timing of all preceding events.

Each combination of the timing of these preceding events that are generated will become the condition for the conditional probability.

A new variable is defined as the condition number that will be generated at each node, denoted as f . At each node, as new conditions are being created, the numbering of f will be re-ordered. This is because the number of condition may increase or decrease from node to node. When determining whether a new condition (for the execution of the next event in the future calendar) should be generated, the current event's conditional execution probability will be calculated. A new condition will only be created if the current event's conditional execution probability is not equal to zero.

As an example, let's examine the following table that shows the conditional scheduled time probability distribution for events LEAVE[8,12] (node 8 in the probability tree diagram in Figure 15. Probability tree diagram) and ENTER[9,12] (node 9). These two events are in the NOS because they have overlapping execution time. The rows of "**6,4,3,0**", "**7,4,3,0**" and "**8,4,3,0**" are the first three conditions that are generated from the previous node, node 7. Comparing the probabilities of occurrence for both events in the time interval [8,9],…,[15,16], under all conditions, LEAVE[8,12] can execute first in the interval of [8,9], [9,10] and [10,11]. On the other hand, ENTER[9,12] has a zero probability of executing first under the condition "**8,4,3,0**" because ENTER[9,12]'s earliest possible execution time is equal to [11,12] while LEAVE[8,12]'s latest possible execution is [10,11]. However, ENTER[9,12] has a non-zero probability of executing first under the conditions of "**6,4,3,0**" and "**7,4,3,0**".

Leave[8,12]	[8,9]	[9,10]	[10,11]	[11,12]
6,4,3,0	0.25	0.5	0.25	
7,4,3,0	0.25	0.5	0.25	
8,4,3,0	0.142857	0.571429	0.285714	

Enter[9,12]	[9,10]	[10,11]	[11,12]	[12,13]	[13,14]	[14,15]	[15,16]
6,4,3,0	0.1	0.2	0.2	0.2	0.2	0.1	
7,4,3,0		0.1	0.2	0.2	0.2	0.2	0.1
8,4,3,0			0.1	0.2	0.2	0.2	0.2

Table 16. Conditional scheduled time probability of Leave[8,12] and Enter[9,12]

Let's now consider creating new conditions for ENTER[9,12]. Since there is a zero probability for ENTER[9,12] to execute first under the condition "**8,4,3,0**", no new conditions will be generated. Table 17. Conditional Probability of Executing First for Enter[9,12] shows the calculated conditional probability for ENTER[9,12] executing first in the interval of [9,10] and [10,11] for the conditions of "**6,4,3,0**" and "**7,4,3,0**". For condition "**6,4,3,0**", two new conditions will be generated, which are "**9,6,4,3,0**" and "**10,6,4,3,0**". For condition "**7,4,3,0**", only one new condition will be generated, namely "**10,7,4,3,0**".

Enter[9,12]	9	10
6,4,3,0	0.05	0.025
7,4,3,0		0.0125

Table 17. Conditional Probability of Executing First for Enter[9,12]

Four variables will be defined to handle the generation of new conditions for the next event(s), the linkage of these conditions to the vertex that executed the current event and the ordering of event sequences. They are defined as follows.

f : is the condition number that will be generated at each node.

w : is the element number for the f^{th} condition. It represents the tier or level of the nodes in a probability tree diagram from QSGM output.

$c(f,w)$: is the lesser endpoint of the timing of the event that is executed under the f^{th} condition and it is the w^{th} element in the order of the event sequence. The series of $c(f,w)$ for a given condition f will give us the timing of all preceding events, which will allow to us to calculate the average execution time for all preceding events.

$d(w)$: is the vertex number of the event that is executed for the w^{th} element in the order of the event sequence

$\Pr(f,w)$: is the probability of occurrence for the f^{th} condition and w^{th} element. It is obtained by normalizing the conditional probability of event *l* executing first in the interval with the lesser endpoint *pp*. If $c(f,w) = pp$, then

$$\Pr(f,w+1) = \frac{Cond\_Exe\_First_l(pp,(f,w)) * \Pr(f,w)}{Sum\_Cond\_Exe\_First(f,w)}$$

Figure 23 shows the values of $c(f,w)$ and $d(w)$ for $w = 1,2,3,4,5$, assuming that the conditional probability that is computed based on the f^{th} condition and the w^{th} element is not equal to zero. For example, if the conditional probability of ENTER[3,6] executing first in [4,5] given that Begin[0,0] executed in [0,0], ENTER[0,0] executed in [0,0] and START[0,0] executed in [0,0] is not equal to zero, then a new condition with c(2,4)=4 and d(4)=4 will be generated. The values of c(2,1), c(2,2) and c(2,3) will be copied from c(1,1), c(1,2) and c(1,3) respectively, so that the values of c(2,1), c(2,2), c(2,3) and c(2,4) become the time series of an event sequence from the *2^nd^* condition.

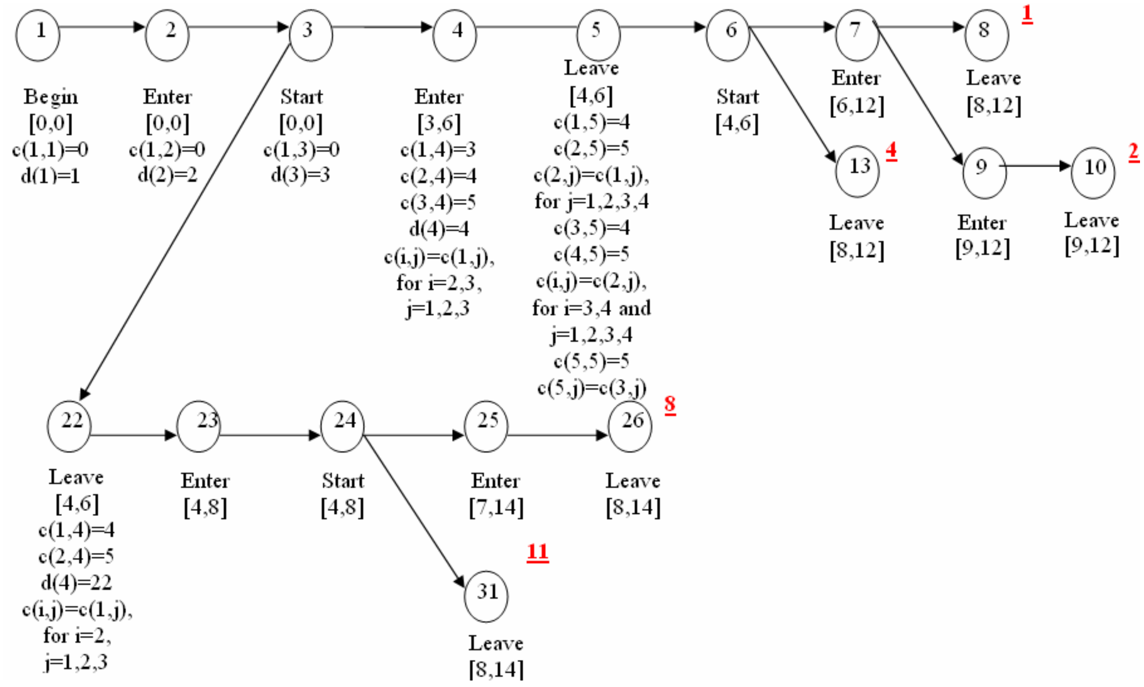


Figure 23. The values of c(f,w) and d(w) for w=1,..,5

If the simulation was to stop at node 5, the following table shows the value of *c(f,w)* for all the five conditions that are generated at node 5. For example, c(4,5)=5 is the lesser endpoint of the execution time for LEAVE[4,6] and d(5)=5 means that node 5 is the vertex that executed the event that is associated with c(4,5). The value of f=4 is the condition number and w=5 means that the event is the 5th element in the thread.

f\w	**1**	**2**	**3**	**4**	**5**
1	0	0	0	3	4
2	0	0	0	3	5
3	0	0	0	4	4
4	0	0	0	4	5
5	0	0	0	5	5

Table 18. The values of c(f,w) at node 5

The following shows a list of new variables that are added in the probability algorithm to determine the expected execution time for an event and the expected delay time between two events. The definition of the variables is also given.

$Timestamp(f,w)$: is the expected execution time for an event from the f^{th} condition and w^{th} element.

$Delay(f,w)$: is the expected delay time between the event from the f^{th} condition and w^{th} element with the event from the same f^{th} condition and $(w\text{-}1)^{th}$ element.

$cursor(f,w)$: is the pointer to the condition which is the predecessor for *c(f,w)*. It is defined to handle copying old values of *c(f,w)*'s from preceding events to the newly created *c(f,w)*.

$rvalue(f,w)$: is the lesser endpoint of the current time interval. It is saved using the index of *(f,w)* the expected execution time (which is denoted as $Timestamp(f,w)$) for consecutive events that have the same execution times can be calculated.

$countsametime(f,w)$: is defined to count the number of consecutive events that have the same execution time, starting from the event associated with the fth condition and wth element, decrementing the values of *w* (or going up the tiers of the probability tree diagram).

The following variables are also defined to compute and store the values of different probabilities.

$\Pr_l(r,(f,w))$: is the probability of scheduled execution time for event *l*. In the algorithm, this probability will first be calculated based on the time when event *l* is scheduled to occur. The calculated probability will be adjusted if it is determined that event *l* intersects with the immediate preceding event.

$Cond\_Exe\_First_l(r,(f,w))$: is the conditional probability for event *l* to execute first in the interval with lesser endpoint *r*. The calculation of this condition execution probability is based on the f^{th} condition and w^{th} element.

$Sum\_Cond\_Exe\_First_l(f,w)$: is the sum of the conditional probability from the f^{th} condition and w^{th} element for event *l*, $Cond\_Exe\_First_l(r,(f,w)),\forall r$.

$PExe\_First_l(r)$: is the probability for event *l* to execute first in the interval with lesser endpoint *r*, which is equal to $\sum_{\forall(f,w)}\left(Cond\_Exe\_First_l(r,(f,w)) * \Pr(f,w)\right)$

$Norm\_Exe\_First_l(r)$: the normalized probability for event *l* to execute first in the interval with lesser endpoint *r*, which is equal to $\dfrac{PExe\_First_l(r)}{Arc\_P_l}$.

Arc_P_z: the sum of $PExe\_First_l(r),\forall r$, i.e. the probability of branching towards the execution of event *l*. If $|N_h|=1$(there is only one event in the NOS), $Arc_P_z=1$as there is only one branch (edge) going out to the vertex that represents event *l*.

$Threadpr(i)$: is the thread probability for thread number i

$thread$: is the variable that label the thread number and is also used as an index for the variable $Threadpr(thread)$

With these definitions, the probability algorithm is carried out as follows. Comments in *Italic* fonts are inserted throughout the algorithm, explaining the main points of the algorithm.

Run Initialization

Initialize the saved state set and counter. $H = \varnothing , h \leftarrow 1, y \leftarrow 0, z \leftarrow 0 \quad f = 0, w = 0, counter\_f = 0$

New Process Initialization

Step 1. Initialize the global simulation clock. $\tau \leftarrow [0,0]$.

Step 2. Insert one event notice into the event calendar: For simplicity, we will assume that the event notice is executed at time [0,0]. $L = L \cup \{([0,0], x_1 . \gamma_1, e_1, a_1, b_1, z_l)\}$

Execute (execution of the model implementation)

Step 1. Determine the NOS, the set Q is the set of all event notices that could be executed next without regard to priority.

$Q = \{ l \mid x_l^- \le (\min(x_l^+) \ \forall \ l \in L) \ \forall \ l \in L \}$

$N_h = \{ q \mid \gamma_q = (\min(\gamma_q) \forall q \in Q) \forall q \in Q \}$

Step 1a. This step is created to calculate the conditional probability for all the events in the event calendar, and for all the conditions that were generated by the immediate preceding event.

```
If $z \ne 0$ then
        For all $l \in L$
        Loop through all the events in the event calendar

                If $x_l \ne [0,0], | N_h | = 1$ then
                If the current event l's execution time is not equal to [0,0] and there is only one event in the NOS

                        For each f where $c(f,w)$ and $d(w)$ exists and $d(w) = z-1$
                        For each condition f from the previous node (z-1), given that the current node is equal to z. This for
                        loop is created to find the conditional scheduled execution time probability for all the events in the
                        event calendar and for each condition f.

                        If $x_{z_l} = [0,0]$ then
                        If the event that scheduled l has zero execution time
                                $s_l = [\max\{c(f,w), b_l^-\}, b_l^+]$
                                The scheduled execution time for event l =[max{lesser endpoint of previous
                                event's execution time, lesser endpoint of the delay interval}, lesser endpoint of
                                the delay interval

                                For $i = s_l^-$ to $s_l^+ - \theta$ step $\theta$
                                        $\Pr_l(i,(f,w)) = \frac{\overline{b}_l(i)}{\sum_{k=i}^{k=b_l^+-\theta} \overline{b}_l(k)}$
                                Next $i$
                        Else
                                $g = c(f,w)$ such that $d(w) = z_l$
                                This step is created to find the location(or the w-th element) of the event that
                                scheduled l (under the f-th condition) and then assign its lesser endpoint of
                                execution time to g.

                                $s_l = [g + b_l^-, g + b_l^+]$

                                For $j = g + b_l^-$ to $g + b_l^+ - \omega$ step $\omega$
                                        If $j = g + b_l^-$ then
                                                $\Pr_l( j,(f,w) ) = \left[ \frac{\overline{b}_l(j-g)}{2} \right]$
                Else if $j = g + b_l^+ - \omega$ then
                        $\Pr_1(j,(f,w)) = \left[ \frac{\bar{b}_l(j-g-\omega)}{2} \right]$
                Else
                        $\Pr_1(j,(f,w)) = \left[ \frac{\bar{b}_l(j-g) + \bar{b}_l(j-g-\omega)}{2} \right]$
                End if
        Next j
End if

If $s_l^- \le c(f,w)$ then
If the lesser endpoint of the scheduled execution time for event l is less than the lesser
endpoint of previous event's execution time, i.e. the scheduled execution time for
the current event l intersects with the previous event's execution time.
        $s_l = [c(f,w), s_l^+]$
        The scheduled execution time for the current event l has to be at least greater
        than or equal to the lesser endpoint of previous event's execution time.

        For $m = s_l^-$ to $s_l^+ - \theta$ step $\theta$
        This for-loop refines the conditional scheduled execution time probability
        calculation, taking into consideration of the effect that the scheduled execution
        time for the current event l intersects with the previous event's execution time.

                If $s_l^- = s_l^+ - \theta$ then
                If $S_l$ is a one-time-unit interval, the conditional probability is equal
                to 1.
                        $\Pr_l(m,(f,w)) = 1$
                Else if $m = s_l^-$ and $s_l^- \ne s_l^+ - \theta$ then
                If this is the first interval (first loop)
                        $Temp\_prl = \frac{\frac{1}{2} * \Pr_l(m,(f,w))}{\frac{1}{2} * \Pr_l(m,(f,w)) + \sum_{k=m+\omega}^{k=s_l^+ - \omega} \Pr_l(k,(f,w))}$
                        $\Pr_l(m,(f,w)) = Temp\_prl$
                        $Temp(m) = 1 - \Pr_l(m,(f,w))$
                Else if $m = s_l^+ - \theta$ and $s_l^- \ne s_l^+ - \theta$ then
                If this is the last interval (last loop)
                        $\Pr_l(m,(f,w)) = Temp(m-\theta)$
                Else
                        $Temp\_prl = Temp(m-\theta) * \frac{\Pr_l(m,(f,w))}{\sum_{k=m+\omega}^{k=s_l^+ - \omega} \Pr_l(k,(f,w))}$
                        $\Pr_l(m,(f,w)) = Temp\_prl$
                        $Temp(m) = Temp(m-\theta) * \left( 1 - \frac{\Pr_l(m,(f,w))}{\sum_{k=m+\omega}^{k=s_l^+ - \omega} \Pr_l(k,(f,w))} \right)$
                        End if
                    Next $m$
                End if
                Next $f$
            End if
        Next $l$
    End if
```

Step 2. If $|N_h| = 1$, then go to Step 6 of Execute.

Step 3. Initialize the variable to loop through the NOS. $n_h \leftarrow 1$

Step 4. Save the state of the simulation by saving the state information in the save-state stack and incrementing the save-state counter. $H_h = \{S, L\}$. $h \leftarrow h+1$.

Step 5. Remove the $(N_{h-1})_{n_{h-1}}$ event notice from L. $L = L\backslash\{l|l = (N_{h-1})_{n_{h-1}}\}$. Event notice l is removed from the calendar. Go to Step 7 of Execute.

Step 6. Remove the first event notice from L. $L = L\backslash\{l|l = (x_1, \gamma_1, e_1, a_1, b_1, z_l)\}$. Event notice l is removed from the calendar.

Step 7. Evaluate the execution edge condition, $X_{e_1}(\vartheta_{e_1}, a_1)$. If $X_{e_1}(\vartheta_{e_1}, a_1) = FALSE$ then go to Step 15 of Execute, else go to Step 8 of Execute.

Step 8. Determine the possible new simulation clock time. $\tau' \leftarrow [\max(x_l^-, \tau^-), \min(x_l^+, \forall\, l \in L)]$.

Step 9. If t_{e_l} is a constant delay interval, determine if the constant delay time is still valid. It is still valid if $t_{e_l} = \tau' - b_l$. If t_{e_l} is a constant interval and still valid, or if t_{e_l} is an uncertain interval, go to Step 11 of Execute, else go to Step 10 of Execute.

Step 10. Set $t_{e_l} \leftarrow \tau' - b_l$. If $t_{e_l}^- \le t_{e_l}^+$, then the new t_{e_l} is valid, but the process must be started at the beginning so that t_{e_l} can be consistent throughout the process. Go to step 1 of New Process Initialization. Otherwise there is no valid constant interval for this process and the process is declared "invalid" and terminates. In that case, go to step 16 of Execute.

Step 11. Update the simulation clock. $\tau \leftarrow \tau'$.

Step 11a. Set $z \leftarrow z+1, V(\Theta) = V(\Theta) \cup z$

If $z_l \neq 0$, then $y \leftarrow y+1, E(\Theta) = E(\Theta) \cup \{y_l, y\}, \psi(\Theta) = \psi(\Theta) \cup \{z_l, z\}$

If $n_{h-1} = 1$, then go to Step 11b, else go to Step 12.

Step 11b. This step is created to calculate the probability of executing first for the current event l.

$counter\_f = 0$

If $z = 1$ then

If the current event l is the first event to execute, it is assume that it is executed at time [0,0] and there is only one edge emanating from the vertex that executed this event. The probability of condition (1,1), the delay time and expected execution time of l is assigned. Example: node 1 in the Simple Queuing System QDES model

$Arc\_P_z = 1$

$c(1,1) = 0$

$d(1) = 1$

$\Pr(1,1) = 1$

$Delay(1,1) = 0$

$Timestamp(1,1) = 0$

Else if $z \neq 1, x_l = [0,0], |N_h| = 1$ then

If the current event l is not the first event to execute, it's execution time is [0,0] and there is only one event in the NOS, then the algorithm will not go into the loop that calculates the conditional execution time probability. Example: node 2 and 3 in the Simple Queuing System QDES model

```
        $Arc\_P_z = 1$
        $counter\_f = counter\_f + 1$
        $c(counter\_f, w+1) = 0$
        $d(w+1) = z$
        $\Pr(counter\_f, w+1) = 1$
        $Delay(counter\_f, w+1) = 0$
        $Timestamp(counter\_f, w+1) = 0$

Else if $z \neq 1, x_l \neq [0,0], |N_h| = 1$ then
If there is only one element in the NOS, the execution time for the current event l is not equal to [0,0] and this is not the first
event to execute in the simulation. Example: node 5, 6, 23, 24, 26 in the Simple Queuing System QDES model.

        For each f where $c(f, w)$ and $d(w)$ exists and $d(w) = z - 1$
        For each condition f from the previous node (z-1), given that the current node is equal to z.
                For $r = s_l^- $ to $s_l^+ - \theta$
                This for-loop creates new condition, increments the counter for the number of conditions and then calculates
                the probability of the newly created condition. The expected delay time and the expected execution time for l
                will also be determined here. The pointer(cursor) and rvalue for the newly created condition are assigned.
                Since this is the only event in NOS, the conditional probability of executing first is equal to its conditional
                scheduled execution probability.

                        $counter\_f = counter\_f + 1$
                        $c(counter\_f, w+1) = r$
                        $\Pr(counter\_f, w+1) = \Pr_l(rr(f, w)) * \Pr(f, w)$
                        $Delay(counter\_f, w+1) = r - c(f, w)$
                        $Timestamp(counter\_f, w+1) = r + \frac{\theta}{2}$
                        $cursor(counter\_f, w+1) = f$
                        $rvalue(counter\_f, w+1) = r$
                        $Cond\_Exe\_First_l(rr, (f, w)) = \Pr_l(rr, (f, w))$

                        If $Delay(counter\_f, w+1) = 0$ and $\gamma \neq 1$ then
                        If the current event l has the same expected execution time with its predecessor, then we will
                        determine to see if how many consecutive events have the same expected execution time so that the
                        expected execution time can be adjusted to the right value.
                                $count = 1$
                                $same = true$

                                Do until $same = false$
                                        If $c(f, w - count + 1) = c(f, w - count)$ then
                                                $count = count + 1$
                                        Else
                                                $same = false$
                                        End if
                                End do
                                $countsametime(counter\_f, w+1) = count$
                        End if
                Next r
        Next f
        $d(w+1) = z$
```

```
For $counter\_f = 1, counter\_f{+}{+}$
        $temp\_p = cursor(counter\_f, w+1)$

        If $countsametime(counter\_f, w+1) = 0$ then
        If none of the consecutive events have the same expected execution time, then we only need to copy the c-matrix, timestamp, delay values from all previous condition's variable values. Adjustment of variable values is not needed.

                For all $qq = 1$ to $w$
                        $c(counter\_f, qq) = c(temp\_p, qq)$
                        $\Pr(counter\_f, qq) = \Pr(temp\_p, qq)$
                        $Timestamp(counter\_f, qq) = Timestamp(temp\_p, qq)$
                        $Delay(counter\_f, qq) = Delay(temp\_p, qq)$
                Next $qq$
        Else
                For cc = 1 to $w - countsametime(counter\_f, w+1)$
                This for loop is created to copy the timestamp, delay time for condition counter_f starting from the first element until the element before adjusting the appropriate timestamp and delay values.

                        $c(counter\_f, cc) = c(temp\_p, cc)$
                        $\Pr(counter\_f, cc) = \Pr(temp\_p, cc)$
                        $Timestamp\,(counter\_f, cc) = Timestamp\,(temp\_p, cc)$
                        $Delay\,(counter\_f, cc) = Delay\,(temp\_p, cc)$
                Next $cc$

                For $bb = 0$ to $countsametime(counter\_f, w+1)$
                This for loop is created to adjust the timestamp and delay values.

                        $c(counter\_f, w - bb + 1) = c(temp\_p, w - bb + 1)$
                        $\Pr(counter\_f, w - bb + 1) = \Pr(temp\_p, w - bb + 1)$
                        $Timestamp\,(counter\_f, w - bb + 1)$
                        $= rvalue(counter\_f, w+1) + \left(\frac{countsametime(counter\_f, w+1) - bb + 1}{countsametime(counter\_f, w+1) + 2}\right) * \theta$

                        If $bb \neq 0$ and $bb \neq countsametime(counter\_f, w+1)$ then
                                $Delay(counter\_f, w - bb + 2)$
                                $= Timestamp(counter\_f, w - bb + 2) - Timestamp(counter\_f, w - bb + 1)$
                        Else if $bb = countsametime(counter\_f, w+1)$ then
                                $Delay(counter\_f, w - bb + 2)$
                                $= Timestamp(counter\_f, w - bb + 2) - Timestamp(counter\_f, w - bb + 1)$
                                $Delay(counter\_f, w - bb + 1)$
                                $= Timestamp(counter\_f, w - bb + 1) - Timestamp(counter\_f, w - bb)$
                        End if
                Next $bb$
        End if
Next counter_f

For $r2 = s_l^-$ to $s_l^+ - \theta$
```

```
        This for-loop calculates the probability of executing first in the interval r2 and its associated probability distribution
        for the current event l.

                $PExe\_First_l(r2) = \sum_{\forall (f,w)} \left( \Pr_l(r2,(f,w)) * \Pr(f,w) \right)$
                $Norm\_Exe\_First_z(r2) = PExe\_First_l(r2)$
                $Arc\_P_l = 1$
        Next r2
Else
        For each f where $c(f,w)$ and $d(w)$ exists and $d(w) = z-1$
                $ts = [ts^-, ts^+] = [s_l^-, \min\{s_l^+, s_{l_1}^+, s_{l_2}^+, s_{l_3}^+, ...\}], \forall l_1, l_2, l_3, ... \in \{(N_{h-1})_{n_{h-1}} \mid l_1, l_2, l_3, .. \neq l\}$
                ts is the time interval in which the current event l could execute first. Event l can execute first as early
                as its lesser endpoint of scheduled execution time and by latest the minimum of all the events' greater
                endpoint scheduled execution time.

                For $r =$ $ts^-$ to $ts^+ - \theta$ step $\theta$
                        $ii = 0$
                        For all events in $l' \in \{(N_{h-1})_{n_{h-1}} \mid l' \neq l\}$
                        This for loop is created to calculate the conditional probability of the current event l executing first
                                in the interval r, under the condition f.

                        This procedure will generate combinations of the binary elements, i.e. {0,1}. The sequence length
                        of the binary combinations is set to $|N_h| - 1$. This means that if we have three NOS events in the
                        event calendar, the binary combinations that will be generated are {00,01,10,11}. Let the three
                        NOS events equal to {A,B,C}, where A is the current event l. The first number from the binary
                        combination represents the indicator for the first event from the set of L\{A}.

                        Assume that the first number represents the indicator for event B. Then the second number from the
                        binary combination represents the indicator for L\{A,B}, i.e. C. In this case, the procedure will
                        generate indicator (ii,jj) for ii=1,2 and jj=1,2,3,4.

                        The binary combinations of {00,01,10,11} can be represented by {(indicator(1,1),indicator(2,1)),
                        (indicator(1,2),indicator(2,2)), (indicator(1,3),indicator(2,3)), (indicator(1,4),indicator(2,4)),}
                        When the indicator value of an event l' is set to 0, it is assume that l' will not be executed in the
                        same interval as the current event l and that l' will be executed (by earliest) after the current
                        interval r. The probability of l' not being in the same interval as the current event l and is executed
                        after l is calculated and stored in the variable, q(ii,jj).

                                $ii = ii + 1$
                                For $jj = 1$ to $2^{|N_h|-1}$
                                        If $\left\lfloor \frac{\left\lceil \frac{jj}{2^{ii-1}} \right\rceil}{2} \right\rfloor = \frac{\left\lceil \frac{jj}{2^{ii-1}} \right\rceil}{2}$ then
                                                $indicator(ii, jj) = 1$
                                                If $s_{l'} \cap [r, r+\theta] = \varnothing$ then
                                                        $q(ii, jj) = 0$
                                                Else
                                                        $q(ii, jj) = \Pr_{l'}(r,(f,w))$
                                                End if
                                        Else
```

```
                    $indicator(ii, jj) = 0$
                    $\lambda = [\max\{s_{l'}^{-}, r+\theta\}, s_{l'}^{+}]$
                    *The time interval for event l' such that l' will not be in the same interval as l will be calculated and stored in the variable $\lambda$.*
                    $q(ii, jj) = \sum_{k=\lambda^{-}}^{k=\lambda^{+}-\theta} \Pr_{l'}(k,(f,w))$
                End if
            Next $jj$
        Next $l'$
        $Cond\_Exe\_First_l(r,(f,w)) = \Pr_l(r,(f,w)) * \sum_{\forall jj} \left[ \frac{1}{\left[\sum_{\forall ii} indicator(ii, jj)\right]+1} * \prod_{\forall ii} q(ii, jj) \right]$
    Next $r$
    $Sum\_Cond\_Exe\_First_l(f,w) = \sum_{\forall r} \left( Cond\_Exe\_First_l(r,(f,w)) \right)$
    For $pp =$ $ts^{-}$ to $ts^{+} - \theta$ step $\theta$
        If $Cond\_Exe\_First_l(pp,(f,w)) \neq 0$ then
            $counter\_f = counter\_f + 1$
            $c(counter\_f, w+1) = pp$
            $\Pr(counter\_f, w+1) = \frac{Cond\_Exe\_First_l(pp,(f,w)) * \Pr(f,w)}{Sum\_Cond\_Exe\_First_l(f,w)}$
            $Delay(counter\_f, w+1) = pp - c(f,w)$
            $Timestamp(counter\_f, w+1) = pp + \frac{\theta}{2}$
            $cursor(counter\_f, w+1) = f$
            $rvalue(counter\_f, w+1) = pp$
            $countsametime(counter\_f, w+1) = 0$

            If $Delay(counter\_f, w+1) = 0$ and $\gamma \neq 1$ then
                $count = 1$
                $same = true$
                Do until $same = false$
                    If $c(f, w-count+1) = c(f, w-count)$ then
                        $count = count + 1$
                    Else
                        $same = false$
                    End if
                End do
                $countsamettime(counter\_f, w+1) = count$
            End if
        End if
    Next $pp$
    $d(w+1) = z$
Next $f$

For $counter\_f = 1, counter\_f ++$
```

```
    $temp\_p = cursor(counter\_f, w+1)$
    If $countsamet\,ime(counter\_f, w+1) = 0$ then
        For all $qq = 1$ to $w$
            $c(counter\_f, qq) = c(temp\_p, qq)$
            $\Pr(counter\_f, qq) = \Pr(temp\_p, qq)$
            $Timestamp(counter\_f, qq) = Timestamp(temp\_p, qq)$
            $Delay(counter\_f, qq) = Delay(temp\_p, qq)$
        Next $qq$
    Else
        For $cc$ = 1 to $w - countsamet\,ime(counter\_f, w+1)$
            $c(counter\_f, cc) = c(temp\_p, cc)$
            $\Pr(counter\_f, cc) = \Pr(temp\_p, cc)$
            $Timestamp\,(counter\_f, cc) = Timestamp\,(temp\_p, cc)$
            $Delay\,(counter\_f, cc) = Delay\,(temp\_p, cc)$
        Next $cc$
        For $bb = 0$ to $countsamet\,ime(counter\_f, w+1)$
            $c(counter\_f, w - bb + 1) = c(temp\_p, w - bb + 1)$
            $\Pr(counter\_f, w - bb + 1) = \Pr(temp\_p, w - bb + 1)$
            $Timestamp\,(counter\_f, w - bb + 1)$
            $= rvalue(counter\_f, w+1) + \left( \frac{countsamet\,ime(counter\_f, w+1) - bb + 1}{countsamet\,ime(counter\_f, w+1) + 2} \right) * \theta$

            If $bb \neq 0$ and $bb \neq countsamet\,ime(counter\_f, w+1)$ then
                $Delay(counter\_f, w - bb + 2)$
                $= Timestamp(counter\_f, w - bb + 2) - Timestamp(counter\_f, w - bb + 1)$
            Else if $bb = countsamet\,ime(counter\_f, w+1)$ then
                $Delay(counter\_f, w - bb + 2)$
                $= Timestamp(counter\_f, w - bb + 2) - Timestamp(counter\_f, w - bb + 1)$
                $Delay(counter\_f, w - bb + 1)$
                $= Timestamp(counter\_f, w - bb + 1) - Timestamp(counter\_f, w - bb)$
            End if
        Next $bb$
    End if
Next counter_f

For all $l$
    For $rr = x_l^- $ to $x_l^+ - \theta$
        $PExe\_First_l(rr) = \sum_{\forall (f,w)} \big( Cond\_Exe\_First_l(rr, (f,w)) * \Pr(f,w) \big)$
    Next $rr$

    $Arc\_P_z = \sum_{\forall rr} PExe\_First_l(rr)$

    For $kk = x_l^-$ to $x_l^+ - \theta$
        $Norm\_Exe\_First_z(kk) = \frac{PExe\_First_l(kk)}{Arc\_P_l}$
    Next $kk$
```

```
        Next $l$
End if
```

Step 12. Assign the attributes to the parameters of the vertex. $P_{v_1} \leftarrow a_1$. If Y is the i^{th} state variable in the vertex parameter list, i.e. $(P_v)_i = Y$, then $Y \leftarrow (a_1)_i$.

Step 13. Evaluate the state change. $S_{v_1} \leftarrow f_{v_l}(S)$.

Step 14. Schedule further events. For each edge, e_{1j}, emanating from v_1, if $C_{e_{1j}}(\Phi_{e_{1j}}) = TRUE$ then

```
    Evaluate $A_{e_{1j}}$ and assign the attribute value of the new event notice, $k$, $a_k \leftarrow A_{e_{1j}}$.
    Generate the inter-event time, $b_k = f(t_{e_{1j}})$, and generate the inter-event distribution $\bar{b}_k$
    If $\hat{b}_k$ is uniformly distributed, then
            $p = \frac{\theta}{b_k^+ - b_k^-}$
    End if

    For $j = b_k^-$ to $b_k^+ - \theta$ increment by $\theta$
            $\bar{b}_k(j) = p$
    End for

    Schedule the event notice where $L = L \cup \{(\tau + b_k, \gamma_{e_{1j}}, v_j, e_{1j}, a_k, b_k, z,)\}$.
```

Step 15. If any of the following conditions are satisfied:

- $\tau > T_{stop}$.
- The simulation stopping condition, ω, evaluates TRUE.
- L is empty.

then the simulation has reached the end of the process. Go to Step 17 of Execute.

Step 16. Go to Step 1 of Execute.

Step 17. If $h = 1$, then terminate the simulation.

Step 18. Restore the last saved system state off the saved-state stack: $h = h - 1$, $L = (L \mid L \in H_h)$, $S = (S \mid S \in H_h)$.

Step 18a. Calculate thread probability

$thread = thread + 1$

$$Threadpr(thread) = \prod_{\forall d(w)} Arc\_P_{d(w)}$$

Step 19. Increment $n_{h-1} \leftarrow n_{h-1} + 1$.

Step 20. If $n_{h-1} \leq |N_{h-1}|$ then go to Step 5 of Execute.

Step 21. Go to Step 17 of Execute.

CHAPTER 5: QDES OUTPUT STATISTICS

5.1 Introduction

A key concept in discrete event simulation modeling is that of a system state description. A system can be characterized by a set of variables with each combination of variable values representing a unique state or condition of the system. The manipulation of the variable values and the record of the timing whenever these variables change values not only provide a means to simulate the movement of the system from state to state, but also allow the computation of statistics for these variables. In the first section of this chapter, the computation of time-persistent and tally statistics in a RDES model are described. In the following section, the computation of both types of statistics for variables of interest for a QDES model is presented. The next section demonstrates the probabilistic approach to calculating statistics in QDES on the Simple Queuing Model.

5.2 Regular Discrete Event Simulation (RDES) Output Data Analysis

There are two main types of statistics that can be collected from an RDES model, namely time-persistent and tally statistics. Time-persistent statistics give the time-weighted values of different variables in the simulation. It is a statistics for which the amount of time a value is observed is used to "weight" the computation. A good example is server utilization. Tally statistics are collected one observation at a time without regard to the amount of time between observations [Ingalls 2002]. An example is the average waiting time.

There are different options available for analyzing RDES experiments depending on whether the simulation is a terminating simulation or non-terminating simulation. Terminating simulation is one for which there is a natural event E that specifies the length of each simulation run or replication [Law 2007]. The event E is specified before any replications are made and it often occurs either at a time point beyond which no useful information is obtained, or when the system is "cleaned out" [Law 2007]. For example, if we are interested in determining the first time at which the queue has at least 10 customers, then E is the event where the queue length reaches 10 for the first time. The QDES model of the Simple Queuing System is an example of a terminating simulation and the terminating condition is when the number of customers exiting the system reaches 2. The initial condition of the system, i.e., the condition under which the system starts could have a large impact on the performance measure since we are interested in the behavior of the system over a finite time horizon [Nakayama 2006].

A non-terminating simulation, also known as steady-state simulation, is one for which there is no natural event E to specify the length of a run and we are interested in the behavior of the system in the long run when it is operating "normally" [Law 2007]. In this case, the impact of the initial condition on behavior of the system over an infinite horizon becomes negligible after a sufficiently long time has elapsed [Nakayama 2006].

In the case of a terminating simulation, statistical analysis could be performed on the output data by making independent replications of the simulation model each terminated by the event E. Using the unbiased point estimator formula for mean and variance, we could obtain a point estimate and $100(1-\alpha)$ percent confidence interval for mean μ [Law 2007]. It is important to ensure that the replications are identically distributed by starting each simulation with the same initial condition and using the same dynamics to govern the evolution of the system [Nakayama 2006].

In the case of a steady-state simulation, two techniques commonly used to analyze output date are the method of batch means and independent replications. The method of batch means involves only one very long simulation run which is suitably subdivided into an initial transient period and *n* batches [Banks, et al. 2005]. Each batch is then treated as an independent replication of the simulation. No observations are made during the transient period as it is treated as a warm-up period. The determination of a warm-up period could be addressed using the Welch's graphical approach [Law 2007]. A sample mean and variance could be formed across the batches.

The method of independent replications is more commonly used for systems with short transient period. It requires independent replications of the simulation from different initial random seeds of the simulator's random number generator. For each independent replications of the simulation run, its transient period is removed [Banks et. al. 2005]. A sample mean and variance could be formed across the replications.

5.3 Qualitative Discrete Event Simulation (QDES) Output Data Analysis

In the previous chapter, we examined a qualitative and probabilistic approach to the modeling of a discrete event system. Instead of collecting sample data by making independent replications of the discrete event system, QDES evaluates all possible scenarios of the discrete event system and calculates the probability of executing each event and scenario as the simulation progresses. The probabilistic approach to analyzing the output data of a QDES model will be examined in this section. In this section, we consider a potentially more informative way of analyzing the output from a QDES model and computing the two different types of statistics.

5.3.1 Time-Persistent Statistics

Time persistent statistics of a RDES model are calculated by observing the amount of time a value, which will be used to "weight" the computation. As an example, let's say we are interested in finding the time-weighted number of customers waiting in line for service. Let *Q(t)* be the number of customers in queue at time t. The value of *Q(t)* is observed for the total amount of time *T*. The time-weighted number of customers waiting in line for service is equal to $\frac{\int_o^T Q(t)dt}{T}$. If we are to plot the graph of *Q(t)* versus *t*, the time-weighted number of customers waiting in line for service could be computed by evaluating the area under the curve of *Q(t)*. However, most RDES simulation packages do not wait until the end of the simulation to compute the value of $\frac{\int_o^T Q(t)dt}{T}$. The statistics are updated when the value that is being tracked changes. For example, let's say that a RDES model has been running for an hour and the average number waiting is 1.2306. For the next 20 seconds, the number of customers waiting in line for service has been 2. If we were to convert the time unit to seconds, the new time-weighted number waiting equals to $\frac{(1.2306 * 3600 + 2 * 20)}{3620} \cong 1.234851$ [Ingalls 2002].

Time persistent statistics of a QDES model can be calculated either at the end of the simulation or whenever the value that is being tracked changes. The key idea is to capture the delay between consecutive events to compute the statistics. In an RDES model, the exact delay between two consecutive events can be easily computed. Unlike in a RDES model, the QDES model's event execution times are represented in time intervals. Even so, the computation of two consecutive events’ delay time is not difficult.

Recall that in Section 4.4.2, we obtained the conditional probability of an event executing first in a time interval, given the conditions of all previous events' execution times. All of the time intervals are represented in one time unit interval, which is the base time unit for the Simple Queuing System QDES model. For example, we are interested in knowing the average server utilization of the Simple Queuing System. First, we need to obtain the conditional probability for the last event in each thread, which contains information on the node number of all events in a given thread, the events' execution times and their probability of occurrence. The information of each event's execution time is used directly in computing the delay times between consecutive events. The node number of each event is needed to get the value of the variable of interest, since all values of the variable of interest are stored in S_{v_1} (refer to the probability algorithm in Section 4.8.3) in which the variable S_{v_1} is directly tied to node v_1. The delay time for two consecutive events is computed, this value is then multiplied with the value of variable S_{v_1} such that v_1 is the node for the first consecutive event. The result is an estimate for how long the variable of interest is equal to the value of S_{v_1}, which is equal to the area under the curve of $\mathrm{S}_{v_1}(t)$ where t is the delay time. The multiplication of the delay time with the value of variable S_{v_1} is computed and accumulated, starting from the first event in a thread and it continues until the simulation reaches the last event in the thread. The statistics for the variable of interest for a given thread are computed at the end of the thread where in the accumulated multiplication of the delay time with the value of variable S_{v_1} is divided by the average execution time of the last event in the thread. This completes the process of calculating time-persistent statistics for a thread. When this process of calculating time-persistent statistics is repeated for all the threads, the average value for the time-persistent statistics is computed by summing the product of thread probability with its associated statistics for the variable of interest. In the next section, the method of computing the delay time of consecutive events will be discussed, followed by an example to demonstrate the method discussed.

5.3.2 Delay Time of Consecutive Events in a Thread

The method of calculating the delay time of two consecutive events in a thread is straightforward for the case when the events' execution times are not equal. Assuming that the base time unit, θ equals to 1. Let's say that an event's execution time is equal to [8,9] and the event following it has an execution time of [9,10] (which does not equal to [8,9]). QDES assumes that the first event could execute any time in [8,9] and the second event could execute any time in [9,10]. In other words, the event's execution times are uniformly distributed in the interval $[t^-, t^+]$ where $(t^+ - t^-)$ equals to the base time unit. So, the average execution time for the first event is equal to $\frac{(8+9)}{2} = 8.5$ and the average execution time for the second event is equal to $\frac{(9+10)}{2} = 9.5$. In this case, the delay time between the two events is determined by subtracting the midpoints of the events' execution times, which is equal to $(9.5 - 8.5) = 1$.

For the case when the consecutive events' execution times are equal, the average execution times are not equal to their midpoints. Since the order of event execution has been determined (or more likely assumed), the first event will have a smaller average execution time compared to the next event in sequence, even though the execution time intervals are equal. Clearly, the average delay time of these two consecutive events in this case does not equal to zero.

Using the same method of calculating the probability of executing first for a NOS, the following shows the calculation of the probability of executing first with its associated expected value for two events (X and Y), each with an execution times of [0,1] with the base time unit, θ equals to 1.

<u>Question:</u> Given that two events X and Y are uniformly distributed on the interval of [0,1]. What is the probability that the execution time of X is less than or equal to Y (in other words, X executed before Y)? What are the expected value of the execution times of X and Y?

Intuitively, since both X and Y are equally likely to occur anywhere on the interval of [0,1], the probability of the execution time for X being less than or equal to Y would be 0.5.

	[0,0.5]	[0.5,1]
Event X	0.5	0.5
	[0,0.5]	[0.5,1]
Event Y	0.5	0.5

Let's divide the interval of [0,1] into two equal portions.

The probability of X executed before Y in the interval of $0 \le X \le 0.5$ is computed as below.

$$\Pr(0 \le X \le 0.5, X \le Y) = \tfrac{1}{2}\left(\tfrac{1}{2} * 0.5 + 1 * 0.5\right) = 0.375$$

Given that *X* executed before *Y* in the interval of $0 \le X \le 0.5$, the expected value of the execution time for X and Y are computed as below.

$$E(X) = \frac{0+0.5}{2} = 0.25 \text{ and } E(Y) = \frac{0+1}{2} = 0.5$$

The same calculation is repeated for the second interval split [0.5, 1]. The probability of *X* executed before *Y* in the interval of $0.5 \le X \le 1$ is computed as below.

$$\Pr(0.5 \le X \le 1, X \le Y) = \tfrac{1}{2}\left(\tfrac{1}{2} * 0.5\right) = 0.125$$

Given that *X* executed before *Y* in the interval of $0.5 \le X \le 1$, the expected value of the execution time for X and Y are computed as below.

$$E(X) = \frac{0.5+1}{2} = 0.75 \text{ and } E(Y) = \frac{0.5+1}{2} = 0.75$$

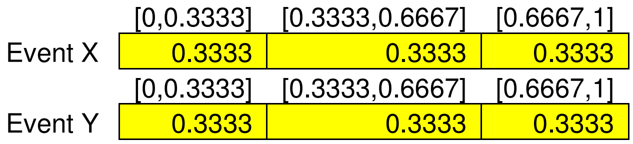
Notice that the probability of the execution time for $X \le Y$ equals to 0.5, as we anticipated.

$$\Pr(X \le Y) = \Pr(0 \le X \le 0.5, X \le Y) + \Pr(0.5 \le X \le 1, X \le Y) = 0.375 + 0.125 = 0.5$$

The expected values of the execution time for *X* and *Y* are given as below.

$$\begin{aligned} E(X) &= 0.25 * \Pr(0 \le X \le 0.5, X \le Y) + 0.75 * \Pr(0.5 \le X \le 1, X \le Y) \\ &= 0.25 * \frac{0.375}{0.5} + 0.75 * \frac{0.125}{0.5} = 0.1875 + 0.1875 \\ &= 0.375 \end{aligned}$$

$$\begin{aligned} E(Y) &= 0.5 * \Pr(0 \le X \le 0.5, X \le Y) + 0.75 * \Pr(0.5 \le X \le 1, X \le Y) \\ &= 0.5 * \frac{0.375}{0.5} + 0.75 * \frac{0.125}{0.5} = 0.375 + 0.1875 \\ &= 0.5625 \end{aligned}$$



	[0,0.3333]	[0.3333,0.6667]	[0.6667,1]
Event X	0.3333	0.3333	0.3333
	[0,0.3333]	[0.3333,0.6667]	[0.6667,1]
Event Y	0.3333	0.3333	0.3333

Now, let's divide the interval of [0,1] into three equal portions.

The probability of X executed before Y in the interval of $0 \le X \le \frac{1}{3}$ is computed as below.

$$\Pr(0 \le X \le \frac{1}{3}, X \le Y) = \frac{1}{3}\left(\frac{1}{2} * \frac{1}{3} + 1 * \frac{2}{3}\right) = \frac{5}{18}$$

Given that X executed before Y in the interval of $0 \le X \le \frac{1}{3}$, the expected value of the execution time for X and Y are computed as below.

$$E(X) = \frac{0 + \frac{1}{3}}{2} = 0.16667 \text{ and } E(Y) = \frac{0+1}{2} = 0.5$$

The probability of X executed before Y in the interval of $\frac{1}{3} \le X \le \frac{2}{3}$ is computed as below.

$$\Pr(\frac{1}{3} \le X \le \frac{2}{3}, X \le Y) = \frac{1}{3}\left(\frac{1}{2} * \frac{1}{3} + 1 * \frac{1}{3}\right) = \frac{1}{6}$$

Given that X executed before Y in the interval of $\frac{1}{3} \le X \le \frac{2}{3}$, the expected value of the execution time for X and Y are computed as below.

$$E(X) = \frac{\frac{1}{3} + \frac{2}{3}}{2} = 0.5 \text{ and } E(Y) = \frac{\frac{1}{3} + 1}{2} = 0.6667$$

The probability of *X* executed before *Y* in the interval of $\frac{2}{3} \le X \le 1$ is computed as below.

$$\Pr(\frac{2}{3} \le X \le 1, X \le Y) = \tfrac{1}{3}\left(\tfrac{1}{2} * \tfrac{1}{3}\right) = \frac{1}{18}$$

Given that *X* executed before *Y* in the interval of $\frac{2}{3} \le X \le 1$, the expected value of the execution time for X and Y are computed as below.

$$E(X) = \frac{\frac{2}{3}+1}{2} = 0.83333 \text{ and } E(Y) = \frac{\frac{2}{3}+1}{2} = 0.83333$$

Notice that the probability of $X \le Y$ equals to 0.5, as we anticipated.

$$\Pr(X \le Y) = \Pr(0 \le X \le \frac{1}{3}, X \le Y) + \Pr(\frac{1}{3} \le X \le \frac{2}{3}, X \le Y) + \Pr(\frac{2}{3} \le X \le 1, X \le Y)$$
$$= \frac{5}{18} + \frac{1}{6} + \frac{1}{18} = 0.5$$

$$E(X) = 0.1667 * \Pr(0 \le X \le \frac{1}{3}, X \le Y) + 0.5 * \Pr(\frac{1}{3} \le X \le \frac{2}{3}, X \le Y) + 0.83333 * \Pr(\frac{2}{3} \le X \le 1, X \le Y)$$
$$= 0.1667 * \frac{\tfrac{5}{18}}{0.5} + 0.5 * \frac{\tfrac{1}{6}}{0.5} + 0.83333 * \frac{\tfrac{1}{18}}{0.5}$$
$$= 0.3518$$

$$E(X) = 0.5 * \Pr(0 \le X \le \frac{1}{3}, X \le Y) + 0.6667 * \Pr(\frac{1}{3} \le X \le \frac{2}{3}, X \le Y) + 0.83333 * \Pr(\frac{2}{3} \le X \le 1, X \le Y)$$
$$= 0.5 * \frac{\tfrac{5}{18}}{0.5} + 0.6667 * \frac{\tfrac{1}{6}}{0.5} + 0.83333 * \frac{\tfrac{1}{18}}{0.5}$$
$$= 0.5926$$

Using MS Excel to repeat the calculations for the number of interval splits equal to 4 to 1000, we could see the progression of the expected value of the execution time for the first event *X* and the second event *Y* in the following table.

Number of interval splits in [0,1]	Expected value of the execution time for X
2	0.375000000000
3	0.351851851852
4	0.343750000000
5	0.340000000000
998	0.333333500669
999	0.333333500334
1000	0.333333500000

Table 19. The progression of the expected value for the execution time of X

From Table 19, we could see that the expected value of the execution time for event X is converging to 0.3333… as the number of interval splits increases. The formula for calculating the expected value of the execution time for the first event could be derived, following that the consecutive events' execution times are equal and thus the probability of a given consecutive event executing in any interval split is equal as well.

Let there be *e* number of consecutive events such that the execution time of each event is equal to [0,1]. Assume that the base time unit is equal to 1.

Let *n*= the number of interval splits in [0,1]

For *e*=2, the probability of a consecutive event *l* executing first in the i^{th} interval split is derived as below.

P(a consecutive event *l* executing first in the i^{th} interval split)

=P(*l* in the i^{th} interval split)*[1*P(the event following *l* in sequence, *l′* executing after the i^{th} interval split)+ $\frac{1}{2}$ *P(*l′* executing in the i^{th} interval split)]

$$=\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)+\frac{1}{2}\left(\frac{1}{n}\right)\right]$$

In general for *e*=2, the probability of a consecutive event *l* executing first in the i^{th} interval split is equal to the multiplication of the probability of event *l* being in the i^{th} interval split (which equals to $\left(\frac{1}{n}\right)$) and the summation of the probability for the event following *l* in sequence, *l'* executing in all possible combinations of interval splits that may affect the probability of *l* executing first in the i^{th} interval split. There are only two such combinations that would affect the probability of *l* executing first in the i^{th} interval split. The first combination is when *l'* executes in the i^{th} interval (which is in the same interval as *l*) with a probability of $\left(\frac{1}{n}\right)$. In this case, the probability of *l* executing first in the i^{th} interval split is equal to $\frac{1}{2}$. Thus, we multiply $\left(\frac{1}{n}\right)$with $\frac{1}{2}$. The other combination is when *l'* executes after the i^{th} interval (which is not in the same interval as *l*) with the probability of $\left(\frac{n-i}{n}\right)$. In this case, *l* would definitely execute first in the i^{th} interval split with a probability of 1.

If the interval [0,1] is split to *n* number of portions, the first interval is equal to $\left[0,\frac{1}{n}\right]$, the second interval is equal to$\left[\frac{1}{n},\frac{2}{n}\right]$ and so on. The i^{th} interval is equal to $\left[\frac{i-1}{n},\frac{i}{n}\right]$. So, the average execution time for an event *l* executing first in the i^{th} interval split is equal to the midpoint of the i^{th} interval, which equals to $\frac{\left(\frac{i-1}{n}+\frac{i}{n}\right)}{2}=\frac{2i-1}{2n}$. In order to calculate the expected value of the timing of *l* executing first, the probability of *l* executing first would have to be normalized. The normalized probability of *l* executing first in the i^{th} interval is then multiplied with the expected value of *l* executing first in the i^{th} interval, summing over *i=1* to *n* to get the expected value of the timing of an event *l* executing first, as shown below.

E(timing of *l* executing first in the i^{th} interval split)

$$=\sum_{i=1}^{n}\left[\frac{\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)+\frac{1}{2}\left(\frac{1}{n}\right)\right]}{\sum_{i=1}^{n}\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)+\frac{1}{2}\left(\frac{1}{n}\right)\right]}\left(\frac{2i-1}{2n}\right)\right]$$

Using Mathematica to compute the above formula shows that as *n* approaches infinity, it is equal to $\frac{1}{3}$. This formula could be extended for $e \geq 3$, as shown below.

For $e \geq 3$,

P(event *l* executing first in the i^{th} interval split)

$$=\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)^{e-1}+\sum_{j=1}^{e-2}\left(C_j^{e-1}\left(\frac{1}{j+1}\right)\left(\frac{1}{n}\right)^{j}\left(\frac{n-i}{n}\right)^{e-1-j}\right)+\frac{1}{e}\left(\frac{1}{n}\right)^{e-1}\right]$$

In general for $e \geq 3$, the probability of a consecutive event *l* executing first in the i^{th} interval split is equal to the multiplication of the probability of event *l* being in the i^{th} interval split (which equals to $\left(\frac{1}{n}\right)$) and the summation of the probability for the events following *l* in sequence (there is a total of *e-1* events except *l*) executing in all possible combinations of interval splits that may affect the probability of *l* executing first in the i^{th}

interval split. There is a total of *(e-1)!* such combinations that would affect the probability of *l* executing first in the i^{th} interval split.

The first combination is when all the events other than *l* executes in the i^{th} interval (which is in the same interval as *l*) with a probability of $\left(\frac{1}{n}\right)^{e-1}$, since there is a total of *e-1* events. In this case, the probability of *l* executing first in the i^{th} interval split is equal to $\frac{1}{e}$. Thus, we multiply $\left(\frac{1}{n}\right)^{e-1}$ with $\frac{1}{e}$. The second combination is when all events not including *l,* executes after the i^{th} interval (which is not in the same interval as *l*) with the probability of $\left(\frac{n-i}{n}\right)^{e-1}$. In this case, *l* would definitely execute first in the i^{th} interval split with a probability of 1.

With the exclusion of the combinations for which all events other than *l* executes in the i^{th} interval and also not in the i^{th} interval (the first and second combinations), all other combinations of events whether each event will execute in the i^{th} interval or not are considered. The summation of the probabilities for all such combinations are represented in the middle term $\sum_{j=1}^{e-2}\left(C_j^{e-1}\left(\frac{1}{j+1}\right)\left(\frac{1}{n}\right)^j\left(\frac{n-i}{n}\right)^{e-1-j}\right)$ where $\left(\frac{1}{n}\right)^j$ is the probability for *j* out of *e-1* events (other than *l*) executing in the i^{th} interval, $\left(\frac{n-i}{n}\right)^{e-1-j}$ is the probability for the remaining *e-1-j* events executing after the i^{th} interval, $\left(\frac{1}{j+1}\right)$ is the probability of *l* executing first with such combination of events, C_j^{e-1} is the combination of *e-1* choose *j* events. As an example, if *e*=4, let's name these four events A, B, C and D. We want to find the expected value of the first event execution time. Assume that A executes first and we are currently considering the i^{th} interval of n interval splits, the first combination in this case is when B, C and D execute in the i^{th} interval, the second combination is when B,C and D not execute in the i^{th} interval. All other combinations are considered and its associated probability is represented in the middle term $\sum_{j=1}^{e-2}\left(C_j^{e-1}\left(\frac{1}{j+1}\right)\left(\frac{1}{n}\right)^j\left(\frac{n-i}{n}\right)^{e-1-j}\right)$, for which the summation of j=1 considers the combinations of 1,2 and 3 from Table 20, in other words, the combination of only 1 event executing in the i^{th} interval. On the other hand, the summation of j=2 considers the combinations of 4,5 and 6 from Table 20, which is the combination of 2 events executing in the i^{th} interval. The number 0 in the B, C and D columns represents the combination of the column event not executing in the i^{th} interval and 1 means otherwise.

Combination	B	C	D
1	0	0	1
2	0	1	0
3	1	0	0
4	1	1	0
5	0	1	1
6	1	0	1

Table 20. Combinations of B,C and D events executing in the i^{th} interval

The expected value of the execution time of l executing first is then equal to summation over i=1 to n of the product of normalized probability of l executing first in the i^{th} interval with the expected value of l executing first in the i^{th} interval.

E(timing of an event *l* executing first)

$$= \sum_{i=1}^{n} \left\{ \frac{\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)^{e-1} + \sum_{j=1}^{e-2}\left(C_j^{e-1}\left(\frac{1}{j+1}\right)\left(\frac{1}{n}\right)^{j}\left(\frac{n-i}{n}\right)^{e-1-j}\right) + \frac{1}{e}\left(\frac{1}{n}\right)^{e-1}\right]}{\sum_{i=1}^{n}\left\{\left(\frac{1}{n}\right)\left[\left(\frac{n-i}{n}\right)^{e-1} + \sum_{j=1}^{e-2}\left(C_j^{e-1}\left(\frac{1}{j+1}\right)\left(\frac{1}{n}\right)^{j}\left(\frac{n-i}{n}\right)^{e-1-j}\right) + \frac{1}{e}\left(\frac{1}{n}\right)^{e-1}\right]\right\}} \left(\frac{2i-1}{2n}\right) \right\}$$

where $\left(C_j^{e-1}\right)$ is the combination of *e-1* choose *j*

Using Mathematica to compute the above formula for the case of $n \to \infty$, the expected value of the first event execution time converges to $\frac{1}{e+1}$. If we are considering the time interval of [4,6], the expected value of the first event execution time is equal to $4+\left[\left(\frac{1}{e+1}\right)(6-4)\right]$.

E(timing of an event *l* executing first in the interval of [0,1]) = $\frac{1}{e+1}$

Given that the expected value of the first event execution time is equal to $\frac{1}{e+1}$ with the time interval under consideration equals to [0,1], the expected value of the second event execution time is between $\frac{1}{e+1}$ and 1. The formula of the expected value $\frac{1}{e+1}$ could be applied to calculate the expected value of the second event execution time. Consider now that the first event is executed and we know the expected value of its execution time is equal to $\frac{1}{e+1}$, there are now $e-1$ events remain to execute in the interval $\left[\frac{1}{e+1},1\right]$. The expected value of the next (or second) event execution time is equal to $\left[\frac{1}{e+1}+\left(\frac{1}{(e-1)+1}\right)\left(1-\frac{1}{e+1}\right)\right]=\left[\frac{1}{e+1}+\left(\frac{1}{e}\right)\left(\frac{e}{e+1}\right)\right]=\left(\frac{2}{e+1}\right)$. With the same concept, the expected value of the j^{th} event execution time is equal to $\frac{j}{e+1}$ when the time interval under consideration is equal to [0,1]. If all the events are scheduled to occur in the time interval of $[a,b]$, the expected value of the j^{th} event execution time is equal to $a+\left(\frac{j}{e+1}\right)*(b-a)$.

E(j^{th}) event execution time in the interval of $[a,b]$) = $a+\left(\frac{j}{e+1}\right)(b-a)$

5.3.3 Server Utilization and Number Waiting of the Simple Queuing System QDES Model

Server utilization is the ratio between the time a server is in use and the total time. So, utilization will be between 0 and 1. The average server utilization and average number waiting for the Simple Queuing System have been manually calculated using MS Excel. In this section, the mechanics of calculating time-persistent statistics will be discussed, using the Simple Queuing System as an example to show how the two time-persistent statistics are calculated.

In the Simple Queuing System QDES model, the variables that are being tracked include the queue length, *Q*, the status of the bank teller, *S* and number of customers that have exited, *E*. The status of the server is busy if *S=0*. If *S=1*, then the bank teller is idle. Server utilization is being tracked by converting the variable values of *S* and store the new values in a new variable, *S'*. When the server is busy, *S'=1* and if idle, *S'=0*. The variable *S'* stores the server utilization values of the Simple Queuing System in every node of the QDES model (see Figure 24).

For illustration purpose, the calculation of the server utilization for thread 8 and the number waiting for thread 4 will be discussed in details. The results of both statistics for all the threads and for the QDES model will be given towards the end of the discussion. The conditional probability for the last event in thread 8 (Node 31) and 4 (Node 13) are shown in Table 21 and Table 22. The "Condition" column in each table consists of the conditions that are generated for the event. For example, the condition of "**4,3**" in Table 21 indicates that Leave (Node 5) and Start (Node 6) are executed in the time interval with the lesser endpoint of 4, Enter (Node 4) is executed in the time interval with the lesser endpoint of 3. The probability of 0.09375 for the condition of "**4,3**" and under the "**8**" column is the conditional probability of LEAVE[8,12] executing in the time interval with the lesser endpoint of 8. As for the condition of "**4,4**" in Table 22, the first number indicates that Leave (Node 22) is executed in the time interval with the lesser endpoint of 4 and the second number indicates that Enter (Node 23) and Start (Node 24) are executed in the time interval with the lesser endpoint of 4. This naming convention of the condition is only for the ease of referencing and relating between the events in the node with its associated execution time. The lesser endpoint of the execution time for the last event is indicated in the column heading after the "Condition" column. Note that the events with zero execution time are omitted from the condition. This is because the delay between any two events for which both events have zero execution time, is equal to zero.

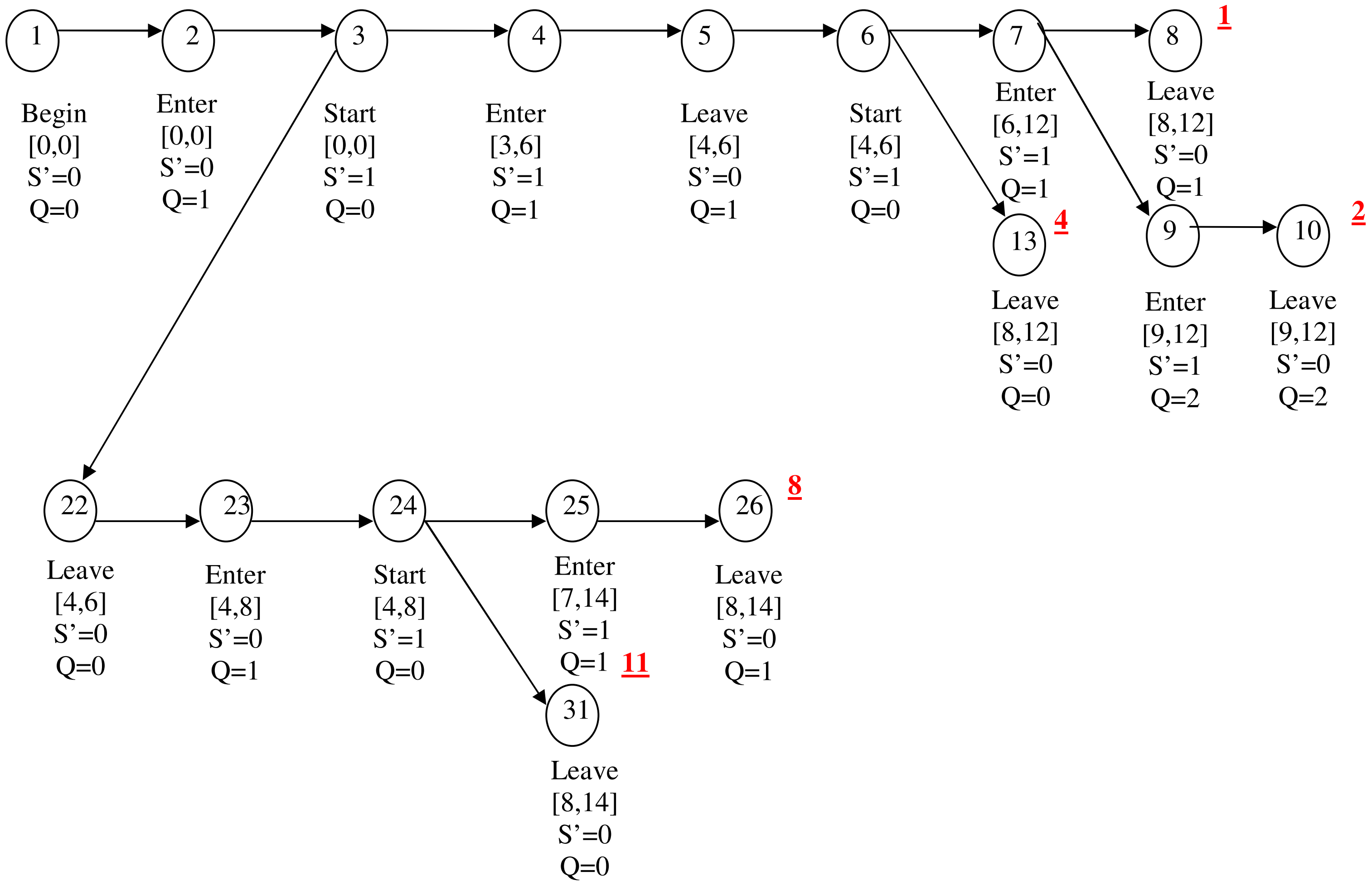


Figure 24. The values of S and Q

Condition	8	9	10	11
4,3	0.09375	0.125	0.03125	0
5,3	0	0.117647	0.117647	0.014706
4,4	0.041667	0.0625	0.020833	0
5,4	0	0.09375	0.125	0.03125
5,5	0	0.041667	0.0625	0.020833

Table 21. Conditional probability for node 13 (Thread 4)

Condition	8	9	10	11	12	13
4,4	0.027777778	0.041667	0.013889	0	0	0
5,4	0	0.055556	0.083333	0.027778	0	0
6,4	0	0	0.055556	0.083333	0.027777778	0
7,4	0	0	0	0.055556	0.083333333	0.027778
5,5	0	0.027778	0.041667	0.013889	0	0
6,5	0	0	0.055556	0.083333	0.027777778	0
7,5	0	0	0	0.055556	0.083333333	0.027778

Table 22. Conditional probability for node 31 (Thread 8)

Now that we have the conditional probability of the last event in each thread, the next task is to compute the average statistics for non-zero cells in each table, for which the conditional probability is not equal to zero.

For each non-zero cell in Table 21, the average number waiting for that cell will be computed. The delay for all the consecutive events from node 3 to 13 of thread 4 will be calculated. The delay time is then multiplied with the value of variable Q from the immediate preceding event. The multiplication of the variable value and the delay time is accumulated from node to node in the thread until it reaches the last node. The final accumulated value is then divided by the expected execution time for the last event in the thread. As an example, the average number waiting for the execution time with the lesser endpoint of 6 and condition “4,3” is equal to $\frac{(3.5-0)*0+(4.5-3.5)*1+(4.5-4.5)*1+(8.5-4.5)*0}{8.5}=0.117647$. The average number waiting for the remaining non-zero cells are computed the same way and they are shown in Table 23.

Condition	8	9	10	11
4,3	0.117647	0.105263	0.095238	
5,3		0.210526	0.190476	0.173913
4,4	0.039216	0.035088	0.031746	
5,4		0.105263	0.095238	0.086957
5,5		0.035088	0.031746	0.028986

Table 23. Average number waiting for thread 4

For each non-zero cell in Table 22, the average server utilization is computed the same way as computing the average number waiting. As an example, the average server utilization for the execution time with the lesser endpoint of 8 and condition "**4,4**" is equal to $\dfrac{\left(4\tfrac{1}{3}-0\right)*1+\left(4\tfrac{2}{3}-4\tfrac{1}{3}\right)*0+\left(4\tfrac{2}{3}-4\tfrac{2}{3}\right)*0+\left(8.5-4\tfrac{2}{3}\right)*1}{8.5}=0.960784$.

The average server utilization for the remaining non-zero cells are computed and shown in Table 24.

Condition	8	9	10	11	12	13
4,4	0.960784	0.964912	0.968254			
5,4		0.894737	0.904762	0.913043		
6,4			0.809524	0.826087	0.84	
7,4				0.73913	0.76	0.777778
5,5		0.964912	0.968254	0.971014		
6,5			0.904762	0.913043	0.92	
7,5				0.826087	0.84	0.851852

Table 24. Average server utilization for thread 8

The average number waiting for thread 4 is equal to the summation of the product of Table 21 and Table 23, which equals to 0.109927. On the other hand, the average server utilization for thread 8 is equal to the summation of the product of Table 22 and Table 24, which equals to 0.866053. The average number waiting and server utilization for each thread are computed and the results are shown in Table 25.

Thread	Server Utilization	Number Waiting
1	1.000000	0.262974
2	1.000000	0.547770
4	1.000000	0.109927
8	0.871161	0.103025
11	0.866053	0.000000

Table 25. The average number waiting and server utilization all the threads

The average server utilization and number waiting for each thread is then multiplied with the thread probability in order to get the final value for the time-persistent statistics. The final average server utilization is equal to 0.920858 and the final average number waiting is equal to 0.106693. The final average values for these two time-persistent statistics are shown in Table 26. An Arena simulation model is created for the Simple Queuing System in order to acquire the statistics results from simulation and compare them to the calculated version from the QDES model. The Arena simulation model was iterated for a total of 10,000 runs and the average number waiting and server utilization from the Arena simulation model are shown in Table 26. Table 26 shows that the calculated version of the time-persistent statistics is close to those obtained from the Arena simulation model.

Result	Server Utilization	Number Waiting
Calculated Statistics	0.920858	0.106693
Arena Simulation Statistics (Half Width for 95% confidence interval)	0.9196 (<0.00)	0.1049 (<0.00)
Difference	0.001258	0.001793

Table 26. Comparison of Arena simulation statistics and calculated statistics from QDES

5.3.4 Tally Statistics

Tally statistics are collected one observation at a time without regard to the amount of time between observations [Ingalls 2002]. Tally statistics in an RDES are collected by tracking the variable of interest for each entity in the model. For example, we wish to tally the time waiting in a queue. This means tracking each entity in the system from the time the entity entered the system to the time it is served. The total amount of waiting time for all the entities that completed the queue wait is then divided by the number of entities that completed the queue wait. The average waiting time of entities in queue = $\frac{\sum_{i=1}^{N} W_i}{N}$, where N=number of entity completed queue wait and W_i = waiting time for the i^{th} entity.

Since there is no entity modeling in QDES, the amount of time for the variable of interest are tracked by defining a pair of events, namely the begin event and end event that will trigger the statistics collector to start and stop tracking time. In the Simple Queuing System QDES model, let's say we want to find the average waiting time of customer in queue, we can define the ENTER event as the begin event and the START event as the end event, since ENTER represents the event when a customer enter the system and START represents the event when the service starts. By capturing the time between ENTER and START events, we can calculate the waiting time for each customer.

The waiting time of a customer in queue and the total time in system for the Simple Queuing System have been manually computed using MS Excel. The begin event and end event for the waiting time tally statistics are defined to be the ENTER and START events. While the begin event and end event for the total time in system tally statistics are defined to be the ENTER and LEAVE events. The Simple Queuing System has a First Come First Serve (FCFS) policy to handle customers, i.e., the customers are attended in the order that they arrive. In this case, the statistics collector will identify the first begin event and first end event as the first pair of events, the second begin event and second end event as the second pair of events and so on, in order to track the amount of time between these pairs of events. The delay time between each pair of events is computed. The i^{th} delay time is equal to the difference between the average execution time for the i^{th} begin event and the average execution time for the i^{th} end event.

After the delay time between begin and end events for a tally statistics has been determined, the conditional probability of the last node in each thread is obtained. The average waiting time for each non-zero conditional probability in a given time interval of the last event in a thread can now be computed. Let's assume that there are *N* number of pairs of begin and end events in a thread. Given the condition of all the events' execution time for the thread (which is obtained from the conditional probability in a given time interval of the last event in a thread), we can sum the i^{th} delay time over i=1 to *N* and by dividing this sum by *N*, we get the average waiting time of customer in queue for condition. This process is repeated for all the non-zero conditional probabilities in a given time interval of the last event in each thread. The average waiting time of a customer in queue for a given thread is computed by summing the product of the average waiting time of customer in queue for condition with its associated conditional probability. To obtain the average waiting time of customer in queue for the QDES model, we can sum over all threads for the multiplication of the average waiting time of a customer in queue for a given thread with its associated thread probability. In the next section, the mechanics of calculating tally statistics for a QDES model will be demonstrated by calculating the total time in system and waiting time of the Simple Queuing System.

5.3.5 Total Time in System and Waiting Time of the Simple Queuing System QDES Model

The total time in system and waiting time of the Simple Queuing System QDES model will be calculated for all the threads. In this section, for illustration purposes, only the calculation for thread 4 will be shown. The calculations for the remaining threads follow the same method.

The conditional probability for the last event in thread 4 is obtained and shown in Table 27, which is the same as Table 21. Recall that the condition of "**4,3**" indicates that Leave (Node 5) and Start (Node 6) are executed in the time interval with the lesser endpoint of 4 and Enter (Node 4) is executed in the time interval with the lesser endpoint of 3. The probability of 0.09375 for the condition of "**4,3**" and under the [8,9] column is the conditional probability of LEAVE[8,12] executing in the time interval with the lesser endpoint of 8 (refer to Figure 25 in the next page for thread 4's event sequence).

Condition	**[8,9]**	**[9,10]**	**[10,11]**	**[11,12]**
4,3	0.09375	0.125	0.03125	0
5,3	0	0.117647	0.117647	0.014706
4,4	0.041667	0.0625	0.020833	0
5,4	0	0.09375	0.125	0.03125
5,5	0	0.041667	0.0625	0.020833

Table 27. Conditional probability for node 13

For the Simple Queuing System QDES model, the beginning and ending events for the total time in system are defined as ENTER and LEAVE events. On the other hand, the beginning and ending events for waiting time are defined as ENTER and START events. There are two pairs of ENTER-LEAVE events and thus we need to determine the sum of the delay time for the two pairs of events. The first pair of ENTER-LEAVE events is shown in Figure 25 and the second pair is shown in Figure 26.

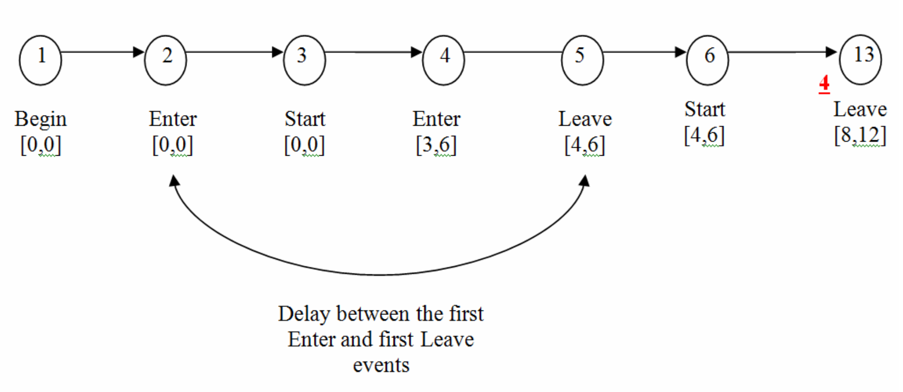


Figure 25. Delay between the first ENTER and first LEAVE events

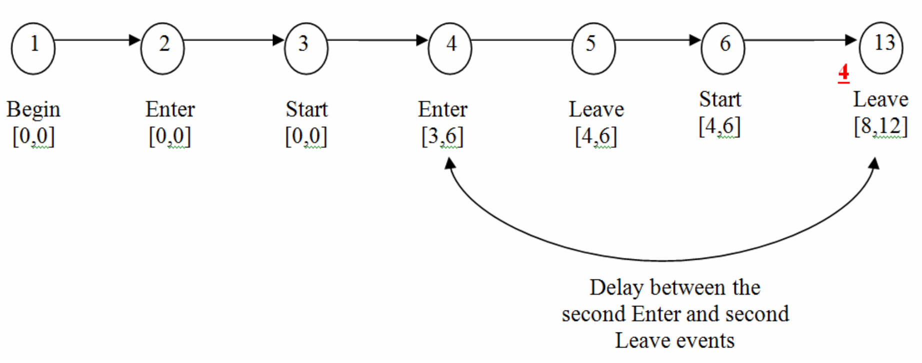


Figure 26. Delay between the second ENTER and second LEAVE events

For each non-zero cells in Table 27, the total time in system will be calculated where the timing of all previous events are obtained from the condition column and the timing of the last event in the thread is obtained from its associated timing column. As an example, let's calculate the total time in system for the cell with the probability of 0.09375 for condition of "**4,3**" and under the [8,9] column. The delay time between the first pair of events ENTER-LEAVE is equal to $(4.5-0)=4.5$. While the delay time between the second pair of events ENTER-LEAVE is equal to $(8.5-3.5)=5$. The sum of delay times from the two pairs of ENTER-LEAVE events is equal to $4.5+5=9.5$.

This sum is then divided by the number of pairs of events, which is 2 and the result is equal to $\frac{9.5}{2} = 4.75$. The total time in system for the remaining non-zero cells in Table 27 are computed the same way and the results are shown in the following table.

Condition	[8,9]	[9,10]	[10,11]	[11,12]
4,3	4.7500	5.2500	5.7500	
5,3		5.7500	6.2500	6.7500
4,4	4.4167	4.9167	5.4167	
5,4		5.2500	5.7500	6.2500
5,5		4.9167	5.4167	5.9167

Table 28. The total time in system for all non-zero execution time conditional probability in node 13

The total time in system for thread 4 is obtained by multiplying the total time in system for the non-zero execution time conditional probability in node 13 with its associated conditional probability, which is the sum of the product of Table 27 and Table 28. Thus, the total time in system for thread 4 is equal to 5.469363.

Let's determine the delay time between the pairs of ENTER-START events in order to calculate the waiting time for thread 4. Figure 27 show that there are two pairs of ENTER-START events in thread 4. The first pair of events has zero execution times, so the delay time between the events is equal to zero as well.

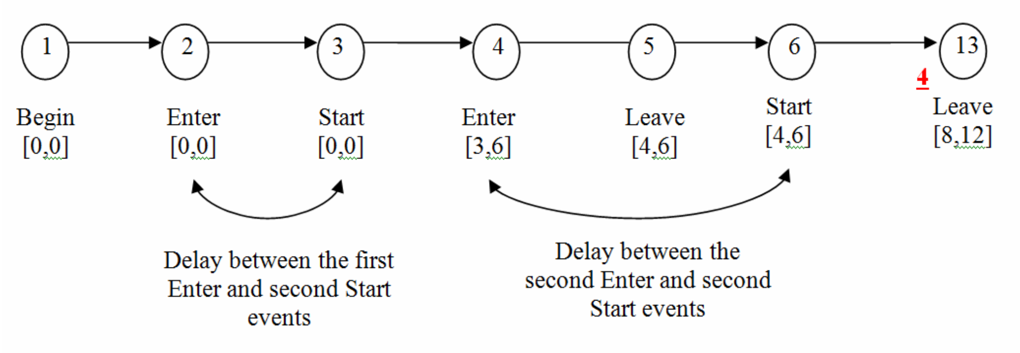

1
2
3
4
5
6
13
Begin
[0,0]
Enter
[0,0]
Start
[0,0]
Enter
[3,6]
Leave
[4,6]
Start
[4,6]
4
Leave
[8,12]
Delay between the first Enter and second Start events
Delay between the second Enter and second Start events

Figure 27. Delay between the first and second pairs of ENTER and START events

As an example, the waiting time for the cell with the probability of 0.09375 for condition of "**4,3**" and under the "**8**" column Table 27 is computed as $\frac{4.5-3.5}{2}=0.5$. The waiting time for all other non-zero cells in Table 27 is calculated and shown in the following table.

Condition	8	9	10	11
4,3	0.5000	0.5000	0.5000	
5,3		1.0000	1.0000	1.0000
4,4	0.1667	0.1667	0.1667	
5,4		0.5000	0.5000	0.5000
5,5		0.1667	0.1667	0.1667

Table 29. The waiting time for all non-zero execution time conditional probability in node 13

The waiting time for thread 4 is computed by summing the product of Table 27 (the conditional probability of node 13) and Table 29 (the waiting time for all non-zero conditional probability of node 13), which equals to 0.541667. The total time in system and waiting time for all threads are computed and shown in the following table.

Thread	Waiting Time	Total Time in System
1	0.541667	5.683388
2	0.775043	6.259168
4	0.541667	5.469363
8	0	5.076389
11	0	4.847222

Table 30. Waiting time and total time in system for all threads

The average waiting time and total time in system for the Simple Queuing System are calculated by summing the product of tally statistics with their associated thread probabilities. The resulting tally statistics are shown in Table 31. Tally statistics are also collected from the Arena simulation model for the Simple Queuing System for average waiting time and total time in system. Based on the results from the 10,000 iterations of the Arena simulation model, the average waiting time is 0.2176. Compared to the calculated version of 0.217860 (refer to Table 31), the difference is 0.000260 which is close to the statistics obtained from Arena simulation. Table 31 also shows that the calculated version of total time in system, which equals to 5.205121 is close to the statistics obtained from Arena simulation, with a difference of -0.008179.

Results	Waiting Time	Total Time in System
Calculated Statistics	0.217860	5.205121
Arena Simulation Statistics (Half width for 95% confidence interval)	0.2176 (<0.01)	5.2133 (<0.01)
Difference	0.000260	-0.008179

Table 31. Average waiting time and total time in system for the Simple Queuing System

From Table 25 and Table 31, the server utilization for thread 1, 2 and 4 is equal to 1. This indicates that the server is busy all the time and on average, customers wait for a total of 0.541667, 0.775043 and 0.541667 time units, respectively. The probability of these scenarios happening is 0.4 altogether.

On the other hand, statistics results for thread 8 and 11 shows that in the scenarios for these threads, the server is less busy and customers did not have to wait for service (waiting times for both cases are equal to zero). These scenarios have a higher probability of occurrence which is 0.6 and they are favorable scenarios as service levels are quite high at 0.871161 and 0.866053 and zero customer waiting times. But since the waiting times for threads 1,2 and 4 are not too long for each scenario (it may seem long from other modeler's point of view) and the probability of these scenarios happening is 0.4, a single server for this system seems to be sufficient to serve customers based on the results above and the QDES model of only two customers exiting the system. The results from Arena simulation model of the Simple Queuing System may not be able to provide insights into how the system performs at the same level of detail.

One of the advantage of using QDES model is that from a single run of the model, we can see how a system performs under each scenario, which scenarios are favorable, what makes it favorable and what is the probability of each scenario happening.

5.4 Probability and Statistics Analysis Algorithm

The statistical analysis method for a QDES model has been discussed in previous sections. This section describes the procedure for executing the statistical analysis for a QSGM output data. The statistical analysis procedure can either be built into the probability algorithm discussed in Section 4.8.3 or implemented as a post-process procedure. Both methods require some information collection during the execution of the probability algorithm. We opt for a built-in statistical analysis procedure into the probability algorithm because the framework of the QSGM and probability algorithm has been carefully structured and the structure is able to support new features of the statistical analysis procedure that will be added and discussed in this section. The new algorithm, added with statistical analysis features will be named the QDES Probability and Statistical Analysis Algorithm.

The definition of L is an ordered set, L $=\{(x_1,\gamma_1,v_1,e_1,a_1,b_1,z_1),(x_2,\gamma_2,v_2,e_2,a_2,b_2,z_2),...\}$ where x_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, a_i are the values of the edge attributes, b_i are the times at which events are scheduled for the i^{th} event notice, and z_i represents the vertex of the event that scheduled the i^{th} event notice. All executed events will be assigned a vertex number, z and an edge number, y that represents the vertex and the edge that has been executed for an event. The scheduled execution time for an event is denoted as s_l.

The definitions for the variables that have been introduced in Section 4.8.2 Execution of the QDES Algorithm will be included in this section for the ease of referencing.

f : is the condition number that will be generated at each node.

w: is the element number for the f^{th} condition. It represents the tier or level of the nodes in a probability tree diagram from QSGM output.

$c(f,w)$: is the lesser endpoint of the timing of the event that is executed under the f^{th} condition and it is the w^{th} element in the order of the event sequence.

$d(w)$: is the vertex number of the event that is executed for the w^{th} element in the order of the event sequence

$\Pr(f,w)$: is the probability of occurrence for the f^{th} condition and w^{th} element. It is obtained by normalizing the conditional probability of event l executing first in the interval with the lesser endpoint pp.

$\Pr_l(r,(f,w))$: is the probability of scheduled execution time for event l.

$Cond\_Exe\_First_l(r,(f,w))$: is the conditional probability for event l to execute first in the interval with lesser endpoint r. The calculation of this condition execution probability is based on the f^{th} condition and w^{th} element.

$Sum\_Cond\_Exe\_First_l(f,w)$: is the sum of the conditional probability from the $f^{\,th}$ condition and w^{th} element for event l, $Cond\_Exe\_First_l(r,(f,w)), \forall r$.

$PExe\_First_l(r)$: is the probability for event *l* to execute first in the interval with lesser endpoint *r*, which is equal to $\sum_{\forall(f,w)}\left(Cond\_Exe\_First_l(r,(f,w)) * \Pr(f,w)\right)$

$Norm\_Exe\_First_l(r)$: the normalized probability for event *l* to execute first in the interval with lesser endpoint *r*, which is equal to $\frac{PExe\_First_l(r)}{Arc\_P_l}$.

Arc_P_z: the sum of $PExe\_First_l(r), \forall r$, i.e. the probability of branching towards the execution of event *l*.

$Threadpr(i)$: is the thread probability for thread number *i*

$thread$: is the variable that label the thread number and is also used as an index for the variable $Threadpr(thread)$

$Timestamp(f,w)$: is the expected execution time for an event from the f^{th} condition and w^{th} element.

$Delay(f,w)$: is the expected delay time between the event from the f^{th} condition and w^{th} element with the event from the same f^{th} condition and $(w\text{-}1)^{th}$ element.

$cursor(f,w)$: is the pointer to the condition which is the predecessor for *c(f,w)*. It is defined to handle copying old values of *c(f,w)*'s from preceding events to the newly created *c(f,w)*.

$rvalue(f,w)$: is the lesser endpoint of the current time interval. It is saved using the index of *(f,w)* the expected execution time (which is denoted as $Timestamp(f,w)$) for consecutive events that have the same execution times can be calculated.

$countsametime(f,w)$: is defined to count the number of consecutive events that have the same execution time, starting from the event associated with the f^{th} condition and w^{th}

element, decrementing the values of w (or going up the tiers of the probability tree diagram).

All the statistics that are to be calculated will be given a name and stored in the variable $stat\_name$. The type of the statistics can be either time-persistent or tally. The type of statistics can be defined at the beginning of the algorithm and stored in the variable $stat\_name.type$.

For each tally statistics, the amount of time for the variable of interest are tracked by defining a pair of events, namely the begin event and end event that will trigger the statistics collector to start and stop tracking time. The begin and end event can also be defined and stored at the beginning of the algorithm in the variables $stat\_name.startevent$ and $stat\_name.stopevent$, respectively. The expected timing of the begin event for tally statistics $stat\_name$ will be stored in $stat\_name.startTime(f,w)$, while the expected timing of the end event will be recorded and stored in $stat\_name.stopTime(f,w)$. If it has been determined in Step 11b (refer to the algorithm in Page 126) that the current event l is the begin event for $stat\_name,$ the variable $stat\_name.startTime(f,w)$ will be assigned the value that is equal to the negative value of the current event l 's expected execution time, thus $stat\_name.startTime(f,w) = -Timestamp(f,w)$. If the current event l is the end event for $stat\_name,$ the variable $stat\_name.stopTime(f,w)$ will be assigned the value of the current event l 's expected execution time, thus $stat\_name.stopTime(f,w) = Timestamp(f,w)$.

In Step 18a (refer to the algorithm in Page 126), the simulation is at the end of a thread, the last saved system is restored off the saved-state stack. The expected timing of begin and end events that are stored in variables $stat\_name.startTime(f,w)$ and $stat\_name.stopTime(f,w)$ during Step 11b will then be stacked onto the $stat\_name.startlist_f$ list and $stat\_name.stoplist_f$ list, respectively. This is to ensure that the expected timing of begin and end events are stacked according to order the execution of each begin and end event. For a system that follows the First Come First Serve (FCFS) policy, the i^{th} delay time between each observation is obtained by adding the i^{th} $stat\_name.startlist_f$ and the i^{th} $stat\_name.stoplist_f$. Since the number of begin and end events may be different, in order to get the number of completed observations for each tally statistics, the number of end events will be counted and stored in $countlist(f)$. The tally statistics for $stat\_name$ under condition f will be stored in variable $stat\_name.value(f)$ which is determined by summing all the i^{th} delay time between each observation and divide the sum by the number of observations, $countlist(f)$. The formula is shown as below.

$$stat\_name.value(f) = \frac{\sum_{i=1}^{countlist(f)} \left[\left(stat\_name.stoplist_f(i) + stat\_name.startlist_f(i) \right) \right]}{countlist(f)}.$$

When the algorithm has finished looping through all conditions, we would have obtained all the $stat\_name.value(f)$ values. The thread statistics can now be calculated and stored in variable $stat\_name.statistics(thread)$. The thread statistics are equal to the sum over all conditions of the product of $stat\_name.value(f)$ and probability of condition *f*. The formula for thread statistics is shown as below.

$$stat\_name.statistics(thread) = \sum_{\forall f} stat\_name.value(f) * \Pr(f, w+1)$$

The final value for each statistics is calculated just before the simulation is terminated. In Step 17, if $h = 1$ (i.e. if this is the last thread), the simulation will go through the for-loop that calculates the final statistics for each $stat\_name$ and store in the variable $stat\_name.finalstatistics$. The formula for calculating each final statistics is

$$stat\_name.finalstatistics = \sum_{\forall thread} stat\_name.statistics(thread) * Threadpr(thread).$$

The definitions for the variables that have been discussed are given as follows.

$stat\_name$: is the name of the statistics to be collected

$stat\_name.type$: is the type of the statistics to be collected, which could be either tally or time-persistent

$stat\_name.startevent$: is the name of the event that sets the start time of an event-based trigger.

$stat\_name.startTime(f, w)$: is the expected execution time for the event from the fth condition and wth element that sets the start time of an event-based trigger

$stat\_name.stopevent$: is the name of the event that sets the stop time of a event-based trigger

$stat\_name.stopTime(f, w)$: is the expected execution time for event from the fth condition and wth element that sets the stop time of an event-based trigger

$stat\_name.startlist_f$:is the list that contains all the start times for $stat\_name$

$stat\_name.stoplist_f$: is the list that contains all the stop times for $stat\_name$

$countlist(f)$: is the counter for the number of end events. The counter is not saved specifically for each tally statistics because the value of this counter will be calculated and referenced immediately in the same loop that calculates the tally statistics for $stat\_name$ under the fth condition

$stat\_name.value(f)$: is the calculated statistics for $stat\_name$ under the fth condition.

$stat\_name.statistics(i)$: is the weighted statistics for thread i over the probability of the condition f for $stat\_name$. The formula for this weighted statistics is given as follows.

$$stat\_name.statistics(i) = \sum_{\forall f} stat\_name.value(f) * \Pr(f, w+1)$$

$stat\_name.finalstatistics$: is the weighted statistics over the thread probability. This is the final value for statistics $stat\_name$ and the formula is given as follows.

$$stat\_name.finalstatistics = \sum_{\forall thread} stat\_name.statistics(thread) * Threadpr(thread)$$

With these definitions, the execution of the Probability and Statistical Analysis Algorithm is carried out as follows. The comments for a specific statement in the algorithm are in *Italic* fonts and it is placed in the next line following the statement.

QDES Probability and Statistical Analysis Algorithm

Run Initialization

Initialize the saved state set and counter. $H = \varnothing, h \leftarrow 1, y \leftarrow 0, z \leftarrow 0 \;\; f = 0, w = 0, counter\_f = 0, thread = 0$

New Process Initialization

Step 1. Initialize the global simulation clock. $\tau \leftarrow [0,0]$.

Step 2. Insert one event notice into the event calendar: For simplicity, we will assume that the event notice is executed at time [0,0]. $L = L \cup \{([0,0], x_1.\gamma_1, e_1, a_1, b_1, z_l)\}$

Execute (execution of the model implementation)

Step 1. Determine the NOS, the set Q is the set of all event notices that could be executed next without regard to priority.

$Q = \{l \mid x_l^- \le (\min(x_l^+) \;\forall\, l \in L) \;\forall\, l \in L\}$

$N_h = \{q \mid \gamma_q = (\min(\gamma_q) \forall q \in Q) \forall q \in Q\}$

Step 1a. This step is created to calculate the conditional probability for all the events in the event calendar, and for all the conditions that were generated by the immediate preceding event.

```
If $z \neq 0$ then
    For all $l \in L$
    Loop through all the events in the event calendar

        If $x_l \neq [0,0], |N_h| = 1$ then
        If the current event l's execution time is not equal to [0,0] and there is only one event in the NOS

            For each f where $c(f,w)$ and $d(w)$ exists and $d(w) = z - 1$
            For each condition f from the previous node (z-1), given that the current node is equal to z. This for loop is created to find the conditional scheduled execution time probability for all the events in the event calendar and for each condition f.

                If $x_{z_l} = [0,0]$ then
                If the event that scheduled l has zero execution time

                    $s_l = [\max\{c(f,w), b_l^-\}, b_l^+]$
                    The scheduled execution time for event l =[max{lesser endpoint of previous event's execution time, lesser endpoint of the delay interval}, lesser endpoint of the delay interval

                    For $i = s_l^-$ to $s_l^+ - \theta$ step $\theta$
                        $\Pr_l(i,(f,w)) = \dfrac{\overline{b}_l(i)}{\sum_{k=i}^{k=b_l^+ - \theta} \overline{b}_l(k)}$
                    Next $i$
                Else
                    $g = c(f,w)$ such that $d(w) = z_l$
                    This step is created to find the location(or the w-th element) of the event that scheduled l (under the f-th condition) and then assign its lesser endpoint of execution time to g.

                    $s_l = [g + b_l^-, g + b_l^+]$

                    For $j = g + b_l^-$ to $g + b_l^+ - \omega$ step $\omega$
                        If $j = g + b_l^-$ then
                            $\Pr_l(j,(f,w)) = \left[\dfrac{\overline{b}_l(j-g)}{2}\right]$
            Else if $j = g + b_l^+ - \omega$ then
                    $\Pr_1 ( j,(f,w) )= \left[ \frac{\bar{b}_l(j-g-\omega)}{2} \right]$
            Else
                    $\Pr_1 ( j,(f,w) )= \left[ \frac{\bar{b}_l(j-g)+\bar{b}_l(j-g-\omega)}{2} \right]$
            End if
        Next $j$
End if

If $s_l^- \le c(f,w)$ then
If the lesser endpoint of the scheduled execution time for event l is less than the
lesser endpoint of previous event's execution time, i.e. the scheduled execution
time for the current event l intersects with the previous event's execution time.
        $s_l = [c(f,w), s_l^+]$
        The scheduled execution time for the current event l has to be at least
        greater than or equal to the lesser endpoint of previous event's
        execution time.

        For $m = s_l^-$ to $s_l^+ - \theta$ step $\theta$
        This for-loop refines the conditional scheduled execution time
        probability calculation, taking into consideration of the effect that the
        scheduled execution time for the current event l intersects with the
        previous event's execution time.

            If $s_l^- = s_l^+ - \theta$ then
            If $s_l$ is a one-time-unit interval, the conditional probability
            is equal to 1.

                    $\Pr_l(m,(f,w)) = 1$
            Else if $m = s_l^-$ and $s_l^- \ne s_l^+ - \theta$ then
            If this is the first interval (first loop)

                    $Temp\_prl =$
                    $\frac{\frac{1}{2} * \Pr_l(m,(f,w))}{\frac{1}{2} * \Pr_l(m,(f,w)) + \sum_{k=m+\omega}^{k=s_l^+-\omega} \Pr_l(k,(f,w))}$
                    $\Pr_l(m,(f,w)) = Temp\_prl$
                    $Temp(m) = 1 - \Pr_l(m,(f,w))$
            Else if $m = s_l^+ - \theta$ and $s_l^- \ne s_l^+ - \theta$ then
            If this is the last interval (last loop)
                    $\Pr_1 ( m,(f,w) )= Temp(m-\theta)$
            Else

                $Temp\_prl = Temp(m-\theta) * \frac{\Pr_l(m,(f,w))}{\sum_{k=m+\omega}^{k=s_l^+-\omega} \Pr_l(k,(f,w))}$
                $\Pr_l(m,(f,w)) = Temp\_prl$
```

```
                                        $Temp(m) = Temp(m-\theta) * \left( 1 - \frac{\Pr_l(m,(f,w))}{\sum_{k=m+\omega}^{k=s_l^+ - \omega} \Pr_l(k,(f,w))} \right)$
                                  End if
                              Next $m$
                          End if
                      Next $f$
                  End if
              Next $l$
End if
```

Step 2. If $|N_h| = 1$, then go to Step 6 of Execute.

Step 3. Initialize the variable to loop through the NOS. $n_h \leftarrow 1$

Step 4. Save the state of the simulation by saving the state information in the save-state stack and incrementing the save-state counter. $H_h = \{S, L\}$. $h \leftarrow h+1$.

Step 5. Remove the $(N_{h-1})_{n_{h-1}}$ event notice from L. $L = L \backslash \{l | l = (N_{h-1})_{n_{h-1}} \}$. Event notice l is removed from the calendar. Go to Step 7 of Execute.

Step 6. Remove the first event notice from L. $L = L \backslash \{l | l = (x_1, \gamma_1, e_1, a_1, b_1, z_l)\}$. Event notice l is removed from the calendar.

Step 7. Evaluate the execution edge condition, $X_{e_1}(\vartheta_{e_1}, a_1)$. If $X_{e_1}(\vartheta_{e_1}, a_1) = FALSE$ then go to Step 15 of Execute, else go to Step 8 of Execute.

Step 8. Determine the possible new simulation clock time. $\tau' \leftarrow [\max(x_l^-, \tau^-), \min(x_l^+, \forall\, l \in L)]$.

Step 9. If t_{e_l} is a constant delay interval, determine if the constant delay time is still valid. It is still valid if $t_{e_l} = \tau' - b_l$. If t_{e_l} is a constant interval and still valid, or if t_{e_l} is an uncertain interval, go to Step 11 of Execute, else go to Step 10 of Execute.

Step 10. Set $t_{e_l} \leftarrow \tau' - b_l$. If $t_{e_l}^- \le t_{e_l}^+$, then the new t_{e_l} is valid, but the process must be started at the beginning so that t_{e_l} can be consistent throughout the process. Go to step 1 of New Process Initialization. Otherwise there is no valid constant interval for this process and the process is declared "invalid" and terminates. In that case, go to step 16 of Execute.

Step 11. Update the simulation clock. $\tau \leftarrow \tau'$.

Step 11a. Set $z \leftarrow z+1, V(\Theta) = V(\Theta) \cup z$

If $z_l \neq 0$, then $y \leftarrow y+1, E(\Theta) = E(\Theta) \cup \{y_l, y\}, \psi(\Theta) = \psi(\Theta) \cup \{z_l, z\}$

If $n_{h-1} = 1$, then go to Step 11b, else go to Step 12.

Step 11b. This step is created to calculate the probability of executing first for the current event *l*.

```
$counter\_f = 0$
If $z = 1$ then
*If the current event l is the first event to execute, it is assume that it is executed at time [0,0] and there is only one edge emanating from the vertex that executed this event. The probability of condition (1,1), the delay time and expected execution time of l is assigned. Example: node 1 in the Simple Queuing System QDES model*
        $Arc\_P_z = 1$
        $c(1,1) = 0$
        $d(1) = 1$
        $\Pr(1,1) = 1$
        $Delay(1,1) = 0$
        $Timestamp(1,1) = 0$

Else if $z \neq 1, x_l = [0,0], | N_h | = 1$ then
```

```
If the current event l is not the first event to execute, it's execution time is [0,0] and there is only one event in the NOS,
then the algorithm will not go into the loop that calculates the conditional execution time probability. Example: node 2
and 3 in the Simple Queuing System QDES model

        $Arc\_P_z = 1$
        $counter\_f = counter\_f + 1$
        $c(counter\_f, w+1) = 0$
        $d(w+1) = z$
        $\Pr(counter\_f, w+1) = 1$
        $Delay(counter\_f, w+1) = 0$
        $Timestamp(counter\_f, w+1) = 0$

        For all $stat\_name \mid stat\_name.type = timepersistent$
        This for loop is created to record the start time or stop time for each stat_name, after it has been determined
        that the current event l is the begin or end event for statistics stat_name
                If $v_l = stat\_name.startevent$ then
                        $stat\_name.startTime(counter\_f, w+1) = -Timestamp(counter\_f, w+1)$
                Else if $v_l = stat\_name.endevent$ then
                        $stat\_name.stopTime(counter\_f, w+1) = Timestamp(counter\_f, w+1)$
                End if
        Next $stat\_name$

Else if $z \neq 1, x_l \neq [0,0], |N_h| = 1$ then
If there is only one element in the NOS, the execution time for the current event l is not equal to [0,0] and this is not the
first event to execute in the simulation. Example: node 5, 6, 23, 24, 26 in the Simple Queuing System QDES model.

For each f where $c(f,w)$ and $d(w)$ exists and $d(w) = z-1$
For each condition f from the previous node (z-1), given that the current node is equal to z.
                For $r = s_l^-$ to $s_l^+ - \theta$
                This for-loop creates new condition, increments the counter for the number of conditions and then
                calculates the probability of the newly created condition. The expected delay time and the expected
                execution time for l will also be determined here. The pointer(cursor) and rvalue for the newly
                created condition are assigned. Since this is the only event in NOS, the conditional probability of
                executing first is equal to its conditional scheduled execution probability.

                        $counter\_f = counter\_f + 1$
                        $c(counter\_f, w+1) = r$
                        $\Pr(counter\_f, w+1) = \Pr_l(rr(f,w)) * \Pr(f,w)$
                        $Delay(counter\_f, w+1) = r - c(f,w)$
                        $Timestamp(counter\_f, w+1) = r + \frac{\theta}{2}$
                        $cursor(counter\_f, w+1) = f$
                        $rvalue(counter\_f, w+1) = r$
                        $Cond\_Exe\_First_l(rr,(f,w)) = \Pr_l(rr,(f,w))$

                        For all $stat\_name \mid stat\_name.type = timepersistent$
                                If $v_l = stat\_name.startevent$ then
                                    $stat\_name.startTime(counter\_f, w+1) = -Timestamp(counter\_f, w+1)$
                                Else if $v_l = stat\_name.endevent$ then
                                    $stat\_name.stopTime(counter\_f, w+1) = Timestamp(counter\_f, w+1)$
                                End if
                        Next $stat\_name$
```

```
                    If $Delay(counter\_f, w+1) = 0$ and $\gamma \neq 1$ then
                    If the current event l has the same expected execution time with its predecessor, then we
                    will determine to see if how many consecutive events have the same expected execution
                    time so that the expected execution time can be adjusted to the right value.
                        $count = 1$
                        $same = true$
                        Do until $same = false$
                            If $c(f, w-count+1) = c(f, w-count)$ then
                                $count = count + 1$
                            Else
                                $same = false$
                            End if
                        End do
                        $countsametime(counter\_f, w+1) = count$
                    End if
                Next $r$
            Next $f$
            $d(w+1) = z$

            For $counter\_f = 1, counter\_f++$
                $temp\_p = cursor(counter\_f, w+1)$

                If $countsametime(counter\_f, w+1) = 0$ then
                If none of the consecutive events have the same expected execution time, then we only need to copy
                the c-matrix, timestamp, delay, start time and stop time values from all previous condition's
                variable values. Adjustment of variable values is not needed.

                    For all $qq = 1$ to $w$
                        $c(counter\_f, qq) = c(temp\_p, qq)$
                        $\Pr(counter\_f, qq) = \Pr(temp\_p, qq)$
                        $Timestamp(counter\_f, qq) = Timestamp(temp\_p, qq)$
                        $Delay(counter\_f, qq) = Delay(temp\_p, qq)$

                        If $stat\_name.startTime(counter\_f, qq) \neq \varnothing$ then
                          $stat\_name.startTime(counter\_f, qq) = stat\_name.startTime(temp\_p, qq)$
                        Else if $stat\_name.stopTime(counter\_f, qq) \neq \varnothing$ then
                          $stat\_name.stopTime(counter\_f, qq) = stat\_name.stopTime(temp\_p, qq)$
                        End if
                    Next $qq$
                Else
                    For $cc$ = 1 to $w - countsametime(counter\_f, w+1)$
                    This for loop is created to copy the timestamp, delay, start time and stop time values for
                    tally statistics stat_name values for condition counter_f starting from the first element
                    until the element before adjusting the appropriate timestamp and delay values.

                        $c(counter\_f, cc) = c(temp\_p, cc)$
                        $\Pr(counter\_f, cc) = \Pr(temp\_p, cc)$
                        $Timestamp(counter\_f, cc) = Timestamp(temp\_p, cc)$
                        $Delay(counter\_f, cc) = Delay(temp\_p, cc)$

                        If $stat\_name.startTime(counter\_f, cc) \neq \varnothing$ then
                          $stat\_name.startTime(counter\_f, cc) = stat\_name.startTime(temp\_p, cc)$
                        Else if $stat\_name.stopTime(counter\_f, cc) \neq \varnothing$ then
                          $stat\_name.stopTime(counter\_f, cc) = stat\_name.stopTime(temp\_p, cc)$
                End if

            Next $cc$

            For $bb = 0$ to $countsametime(counter\_f, w+1)$
            This for loop is created to adjust the timestamp and delay values.

                $c(counter\_f, w-bb+1) = c(temp\_p, w-bb+1)$
                $\Pr(counter\_f, w-bb+1) = \Pr(temp\_p, w-bb+1)$
                $Timestamp\,(counter\_f, w-bb+1)$
                $= rvalue(counter\_f, w+1) + \left( \frac{countsametime(counter\_f, w+1) - bb + 1}{countsametime(counter\_f, w+1) + 2} \right) * \theta$

                If $bb \neq 0$ and $bb \neq countsametime(counter\_f, w+1)$ then
                    $Delay(counter\_f, w-bb+2)$
                    $= Timestamp(counter\_f, w-bb+2) - Timestamp(counter\_f, w-bb+1)$
                Else if $bb = countsametime(counter\_f, w+1)$ then

                    $Delay(counter\_f, w-bb+2)$
                    $= Timestamp(counter\_f, w-bb+2) - Timestamp(counter\_f, w-bb+1)$
                    $Delay(counter\_f, w-bb+1)$
                    $= Timestamp(counter\_f, w-bb+1) - Timestamp(counter\_f, w-bb)$
                End if

                If $stat\_name.startTime(counter\_f, w-bb+1) \neq \emptyset$ then
                        $stat\_name.startTime(counter\_f, w-bb+1)$
                        $= -Timestamp(counter\_f, w-bb+1)$
                Else if $stat\_name.stopTime(counter\_f, w-bb+1) \neq \emptyset$ then
                        $stat\_name.startTime(counter\_f, w-bb+1)$
                        $= -Timestamp(counter\_f, w-bb+1)$
                End if

            Next $bb$
        End if
    Next counter_f

    For $r2 = s_l^-$ to $s_l^+ - \theta$
    This for-loop calculates the probability of executing first in the interval r2 and its associated probability
    distribution for the current event l.
        $PExe\_First_l(r2) = \sum_{\forall(f,w)} \left( \Pr_l(r2,(f,w)) * \Pr(f,w) \right)$
        $Norm\_Exe\_First_z(r2) = PExe\_First_l(r2)$
        $Arc\_P_l = 1$
    Next r2
Else
    For each f where $c(f,w)$ and $d(w)$ exists and $d(w) = z-1$
        $ts = [ts^-, ts^+] = [s_l^-, \min\{s_l^+, s_{l_1}^+, s_{l_2}^+, s_{l_3}^+, ...\}], \forall l_1, l_2 l_3, ... \in \{(N_{h-1})_{n_{h-1}} \mid l_1, l_2, l_3, .. \neq l\}$
        ts is the time interval in which the current event l could execute first. Event l can execute first
        earliest by its lesser endpoint of scheduled execution time and by latest the minimum of all the
        events' greater endpoint of scheduled execution time.

        For $r = ts^-$ to $ts^+ - \theta$ step $\theta$
            $ii = 0$
            For all events in $l' \in \{(N_{h-1})_{n_{h-1}} \mid l' \neq l\}$
            This for loop is created to calculate the conditional probability of the current event l
            executing first in the interval r, under the condition f.

            This procedure will generate combinations of the binary elements, i.e. {0,1}. The
            sequence length of the binary combinations is set to $\mid N_h \mid -1$. This means that if we have
            three NOS events in the event calendar, the binary combinations that will be generated
            are {00,01,10,11}. Let the three NOS events equal to {A,B,C}, where A is the current
            event l. The first number from the binary combination represents the indicator for the
            first event from the set of L\{A}.

            Assume that the first number represents the indicator for event B. Then the second
            number from the binary combination represents the indicator for L\{A,B}, i.e. C. In this
            case, the procedure will generate indicator (ii,jj) for ii=1,2 and jj=1,2,3,4.

            The binary combinations of {00,01,10,11} can be represented by
            {(indicator(1,1),indicator(2,1)), (indicator(1,2),indicator(2,2)),
            (indicator(1,3),indicator(2,3)), (indicator(1,4),indicator(2,4)),}

            When the indicator value of an event l' is set to 0, it is assume that l' will not be executed
            in the same interval as the current event l and that l' will be executed (by earliest) after
            the current interval r. The probability of l' not being in the same interval as the current
            event l and is executed after l is calculated and stored in the variable, q(ii,jj).

                $ii = ii + 1$
                For $jj = 1$ to $2^{|N_h|-1}$
                    If $\left\lfloor \frac{\left\lceil \frac{jj}{2^{ii-1}} \right\rceil}{2} \right\rfloor = \frac{\left\lceil \frac{jj}{2^{ii-1}} \right\rceil}{2}$ then
                        $indicator(ii, jj) = 1$
                        If $s_{l'} \cap [r, r+\theta] = \varnothing$ then
                            $q(ii, jj) = 0$
                        Else
                            $q(ii, jj) = \Pr_{l'}(r, (f, w))$
                        End if
                    Else
                        $indicator(ii, jj) = 0$
                        $\lambda = [\max\{s_{l'}^{-}, r+\theta\}, s_{l'}^{+}]$
                        The time interval for event l' such that l' will not be in the
                        same interval as l will be calculated and stored in the
                        variable $\lambda$.
                        $q(ii, jj) = \sum_{k=\lambda^{-}}^{k=\lambda^{+}-\theta} \Pr_{l'}(k, (f, w))$
                    End if
                Next jj
            Next l'

            $Cond\_Exe\_First_l(r, (f, w)) = \Pr_l(r, (f, w)) * \sum_{\forall jj} \left[ \frac{1}{\left[ \sum_{\forall ii} indicator(ii, jj) \right] + 1} * \prod_{\forall ii} q(ii, jj) \right]$

        Next r
        $Sum\_Cond\_Exe\_First_l(f,w) = \sum_{\forall r}\left(Cond\_Exe\_First_l(r,(f,w))\right)$
        For $pp =$ $ts^-$ to $ts^+ - \theta$ step $\theta$
            If $Cond\_Exe\_First_l(pp,(f,w)) \neq 0$ then
                $counter\_f = counter\_f + 1$
                $c(counter\_f, w+1) = pp$
                $\Pr(counter\_f, w+1) = \frac{Cond\_Exe\_First_l(pp,(f,w)) * \Pr(f,w)}{Sum\_Cond\_Exe\_First_l(f,w)}$
                $Delay(counter\_f, w+1) = pp - c(f,w)$
                $Timestamp(counter\_f, w+1) = pp + \frac{\theta}{2}$
                $cursor(counter\_f, w+1) = f$
                $rvalue(counter\_f, w+1) = pp$
                $countsametime(counter\_f, w+1) = 0$

                For all $stat\_name \mid stat\_name.type = timepersistent$
                    If $v_l = stat\_name.startevent$ then
                        $stat\_name.startTime(counter\_f, w+1) =$
                        $-Timestamp(counter\_f, w+1)$
                    Else if $v_l = stat\_name.endevent$ then
                        $stat\_name.stopTime(counter\_f, w+1)$
                        $= Timestamp(counter\_f, w+1)$
                    End if
                Next $stat\_name$

                If $Delay(counter\_f, w+1) = 0$ and $\gamma \neq 1$ then
                    $count = 1$
                    $same = true$
                    Do until $same = false$
                        If $c(f, w - count + 1) = c(f, w - count)$ then
                            $count = count + 1$
                        Else
                            $same = false$
                        End if
                    End do
                    $countsamet ime(counter\_f, w+1) = count$
                End if
            End if
        Next $pp$
        $d(w+1) = y$
Next $f$

For $counter\_f = 1, counter\_f++$
    $temp\_p = cursor(counter\_f, w+1)$
    If $countsamet ime(counter\_f, w+1) = 0$ then
        For all $qq = 1$ to $w$
            $c(counter\_f, qq) = c(temp\_p, qq)$
            $\Pr(counter\_f, qq) = \Pr(temp\_p, qq)$
            $Timestamp(counter\_f, qq) = Timestamp(temp\_p, qq)$
            $Delay(counter\_f, qq) = Delay(temp\_p, qq)$
                If $stat\_name.startTime(counter\_f,qq) \neq \varnothing$ then
                    $stat\_name.startTime(counter\_f,qq) = stat\_name.startTime(temp\_p,qq)$
                Else if $stat\_name.stopTime(counter\_f,qq) \neq \varnothing$ then
                    $stat\_name.stopTime(counter\_f,qq) = stat\_name.stopTime(temp\_p,qq)$
                End if
            Next $qq$
        Else
            For $cc = 1$ to $w - countsametime(counter\_f, w+1)$
                $c(counter\_f,cc) = c(temp\_p,cc)$
                $\Pr(counter\_f,cc) = \Pr(temp\_p,cc)$
                $Timestamp\,(counter\_f,cc) = Timestamp\,(temp\_p,cc)$
                $Delay\,(counter\_f,cc) = Delay\,(temp\_p,cc)$

                If $stat\_name.startTime(counter\_f,cc) \neq \varnothing$ then
                    $stat\_name.startTime(counter\_f,cc) = stat\_name.startTime(temp\_p,cc)$
                Else if $stat\_name.stopTime(counter\_f,cc) \neq \varnothing$ then
                    $stat\_name.stopTime(counter\_f,cc) = stat\_name.stopTime(temp\_p,cc)$
                End if
            Next $cc$

            For $bb = 0$ to $countsametime(counter\_f, w+1)$
                $c(counter\_f, w-bb+1) = c(temp\_p, w-bb+1)$
                $\Pr(counter\_f, w-bb+1) = \Pr(temp\_p, w-bb+1)$
                $Timestamp\,(counter\_f, w-bb+1)$
                $= rvalue(counter\_f, w+1) + \left( \frac{countsametime(counter\_f, w+1) - bb + 1}{countsametime(counter\_f, w+1) + 2} \right) * \theta$

                If $bb \neq 0$ and $bb \neq countsametime(counter\_f, w+1)$ then

                    $Delay(counter\_f, w-bb+2)$
                    $= Timestamp(counter\_f, w-bb+2) - Timestamp(counter\_f, w-bb+1)$
                Else if $bb = countsametime(counter\_f, w+1)$ then

                    $Delay(counter\_f, w-bb+2)$
                    $= Timestamp(counter\_f, w-bb+2) - Timestamp(counter\_f, w-bb+1)$
                    $Delay(counter\_f, w-bb+1)$
                    $= Timestamp(counter\_f, w-bb+1) - Timestamp(counter\_f, w-bb)$
                End if

                If $stat\_name.startTime(counter\_f, w-bb+1) \neq \varnothing$ then
                        $stat\_name.startTime(counter\_f, w-bb+1)$
                        $= -Timestamp(counter\_f, w-bb+1)$
                Else if $stat\_name.stopTime(counter\_f, w-bb+1) \neq \varnothing$ then
                        $stat\_name.stopTime(counter\_f, w-bb+1)$
                        $= -Timestamp(counter\_f, w-bb+1)$
                End if
            Next $bb$
        End if
Next counter_f

For all $l$
    For $rr = x_l^- $ to $x_l^+ - \theta$
```

```
            $PExe\_First_l(rr) = \sum_{\forall (f,w)} \big(Cond\_Exe\_First_l(rr,(f,w)) * \Pr(f,w)\big)$
        Next $rr$
        $Arc\_P_z = \sum_{\forall rr} PExe\_First_l(rr)$
        For $kk = x_l^-$ to $x_l^+ - \theta$
            $Norm\_Exe\_First_z(kk) = \frac{PExe\_First_l(kk)}{Arc\_P_l}$
        Next $kk$
    Next $l$
End if
```

Step 12. Assign the attributes to the parameters of the vertex. $P_{v_1} \leftarrow a_1$. If Y is the i^{th} state variable in the vertex parameter list, i.e. $(P_v)_i = Y$, then $Y \leftarrow (a_1)_i$.

Step 13. Evaluate the state change. $S_{v_1} \leftarrow f_{v_l}(S)$.

Step 14. Schedule further events. For each edge, e_{1j}, emanating from v_1, if $C_{e_{1j}}(\Phi_{e_{1j}}) = TRUE$ then

```
    Evaluate $A_{e_{1j}}$ and assign the attribute value of the new event notice, $k$, $a_k \leftarrow A_{e_{1j}}$.
    Generate the inter-event time, $b_k = f(t_{e_{1j}})$, and generate the inter-event distribution $\overline{b_k}$
    If $\widehat{b_k}$ is uniformly distributed, then
        $p = \frac{\theta}{b_k^+ - b_k^-}$
    End if

    For $j = b_k^-$ to $b_k^+ - \theta$ increment by $\theta$
        $\overline{b_k}(j) = p$
    End for

    Schedule the event notice where $L = L \cup \{(\tau + b_k, \gamma_{e_{1j}}, v_j, e_{1j}, a_k, b_k, z,)\}$.
```

Step 15. If any of the following conditions are satisfied:

- $\tau > T_{stop}$.
- The simulation stopping condition, ω, evaluates TRUE.
- L is empty.

then the simulation has reached the end of the process. Go to Step 17 of Execute.

Step 16. Go to Step 1 of Execute.

Step 17. If $h = 1$, calculate the calculate the final statistics for each stat_name, then terminate the simulation. The for-loop for calculating the final statistics for each stat_name is shown as below.

```
For each stat_name
    For $\forall thread$

        $stat\_name.finalstatistics = \sum_{\forall thread} stat\_name.statistics(thread) * Threadpr(thread)$

    Next thread
Next stat_name
```

Step 18. Restore the last saved system state off the saved-state stack: $h = h - 1$, $L = (L \mid L \in H_h)$, $S = (S \mid S \in H_h)$.

Step 18a. Calculate thread probability and thread statistics for $\forall stat\_name$.

$thread = thread + 1$

$Threadpr(thread) = \prod_{\forall d(w)} Arc\_P_{d(w)}$

```
For $\forall stat\_name$
    If $stat\_name.type = tally$ then
        For all $f = 1, f++$ in $c(f, w+1)$
            $countlist(f) = 0$
            For all $w = 1, w++$ in $c(f, w+1)$
                If $stat\_name.startTime(f, w+1) \neq \varnothing$ then
                    $stat\_name.startlist_f = stat\_name.startlist_f \cup stat\_name.startTime(f, w)$
                End if
                If $stat\_name.stopTime(f, w+1) \neq \varnothing$ then
                    $stat\_name.stoplist_f = stat\_name.stoplist_f \cup stat\_name.stopTime(f, w)$
                    $countlist(f) = countlist(f) + 1$
                End if
            Next $w$
            $stat\_name.value(f) = \sum_{i=1}^{countlist(f)} \left[ \frac{\left(stat\_name.stoplist_f(i) + stat\_name.startlist_f(i)\right)}{countlist(f)} \right]$
        Next $f$
        $stat\_name.statistics(thread) = \sum_{\forall f} stat\_name.value(f) * \Pr(f, w+1)$
    Else
        For all $f = 1, f++$ in $c(f, w+1)$
            $stat\_name.value(f) = \frac{\sum_{i=1}^{w+1} Delay(f, i) * S_{stat\_name.stateVariable}(i)}{Timestamp(counter\_f, w+1)}$
        Next $f$
        $stat\_name.statistics(thread) = \sum_{\forall f} stat\_name.value(f) * \Pr(f, w+1)$
    End if
Next $stat\_name$
```

Step 19. Increment $n_{h-1} \leftarrow n_{h-1} + 1$.

Step 20. If $n_{h-1} \leq | N_{h-1} |$ then go to Step 5 of Execute.

Step 21. Go to Step 17 of Execute.

CHAPTER 6: CONCLUSION AND FUTURE RESEARCH

In this chapter, we will summarize the important findings of this dissertation and outline some of the work that can be undertaken to further enhance the QDES methodology.

The probabilistic nature of our approach to calculating statistics from a QDES output opens up new chapter on creating a closed-form discrete event simulation output. The key behind the approach is that by imposing statistical distribution on the temporal intervals and using a complex set of conditional probabilities for the thread, the statistics in the QDES model is exactly represented as we have shown in our example using uniform distribution.

The probability calculation presented in Chapter 4 lays the foundation for developing a probabilistic approach to the statistical analysis in the QDES methodology. The conditional execution time probability for each event, the execution time probability distribution for each event, the arc probability, and the thread probability is calculated based on basic probability theory. The generation of all possible conditions provides the necessary timing information for all events in a thread, which allows us to calculate the statistics for any variable of interest.

One other finding in this dissertation is the calculation of the expected execution time for consecutive events in a thread with the same execution time interval. The equation for the expected execution time for consecutive events is derived and the resulting equation is used directly towards calculating statistics for a QDES model.

The delay constraint validity algorithm that is presented in Chapter 3 refines the current QDES methodology. The implementation of this algorithm in the current QDES methodology will allow for the finding of invalid threads that do not satisfy a delay constraint that exists between an event and its scheduling event, and excluding them from further analysis. The proposed method for checking delay constraint validity between an event and its scheduling event is based on a top-down search approach for which the search for delay bounds starts from the scheduled event down to the scheduling event.

6.1 Future Research

Because this dissertation creates a new sophisticated probabilistic approach to the statistical analysis of a QDES model, there is much work to be done in this area. The assumption of uniform distribution on all the delay intervals is used in this dissertation so that the probability of an event occurrence can be calculated. Other non-uniform distributions can be used to represent the delay intervals' probability distribution, even a user-defined probability distribution. This would be a good research project to examine the robustness of the QDES framework.

The following subsections list some other research topic that can be undertaken to further enhance the QDES methodology.

6.1.1 Scaling QDES Methodology

A more thorough analysis can be done in scaling the QDES methodology with the statistical analysis features to larger and more complex problems, or even industrial size problems. The efficiency of the QDES methodology with statistical analysis feature can then be analyzed and improved. An application of the QDES methodology without the statistical analysis features to solve a PERT scheduling with resources problem is presented in Ingalls and Morrice[2004]. A good research project would be to apply the

QDES methodology with statistical analysis features to the same problem. The statistics results will be able to provide more insights to project managers when monitoring and controlling a PERT schedule on an ongoing basis.

6.1.2 Delay Bounds

In the Simple Queuing System example that was presented, delay bounds do not intersect with each other. In a larger and more complex system, such situation may exist. We have put forth one search technique, other search techniques that consider different combinations of delay bounds in the case of at least one intersection exists between delay bounds are of value.

The presented search technique for delay bounds within the scheduled event and its scheduling event is based on a top down search. For example, let's say that we are currently checking the delay constraint validity for node 7 (refer to Figure 28). Node 7 is scheduled by node 2. The next step is to calculate the total elapsed time from node 2 to node 7. The delay constraint validity algorithm will start accumulating the elapsed time from node 7, working its way down to node 2. The algorithm will first calculate the elapsed time from node 6 to node 7. At node 6 (before calculating the elapsed time), the algorithm will check to see if there is a delay bound that exists for node 6.

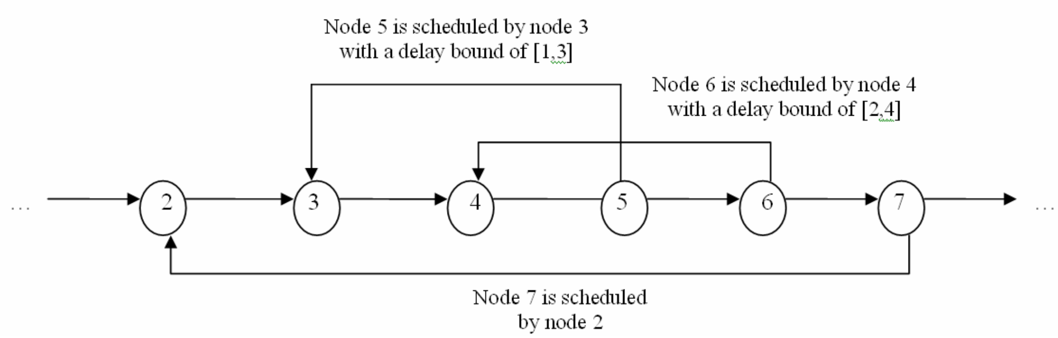

Node 5 is scheduled by node 3
with a delay bound of [1,3]
Node 6 is scheduled by node 4
with a delay bound of [2,4]
...
2
3
4
5
6
7
...
Node 7 is scheduled
by node 2

Figure 28. Delay Bound Example

Let's assume that node 6 is scheduled by node 4 with a delay bound of [2,4]. The algorithm will add [2,4] to the elapsed time from node 6 to node 7. The next node the algorithm will check for delay bound is node 3, since we have taken into consideration the elapsed time from node 4 to node 6. This means that we have skipped the delay bound for node 5. The total elapsed time for this example using the presented top down search technique for delay bound is equal to (elapsed time from node 2 to 3) + (elapsed time from node 3 to 4) + (delay bound from node 4 to 6) + (elapsed time from node 6 to 7).

Another search technique for delay bound can be a bottom up search which will consider the delay bound from node 3 to 5 instead of the delay bound from node 4 to 6. The same problem of not taking all delay bounds into consideration for the calculation of total elapsed time from the scheduled event of the current node to its scheduling event still exists. The delay bounds that exist within a specified time frame can be simple or complex depends on whether there is any intersection between delay bounds. The research in finding for a possible assignment of times on the real line to the intervals such that all the intersection relations are satisfied can be of value to the QDES methodology. This type of research may be more suitable for researchers in the area of interval algebra.

6.1.3 Application in Simulation Optimization Problems

Simulation optimization is the process of finding an optimum design of a system whose performance measure(s) are estimated via simulation [Kabirian and Olafsson 2007]. Simulation optimization methods are employed in situations where it is practically impossible or computationally expensive to obtain the closed form objective function based on decision variables. According to Kabirian and Olaffson (2007), most of the practical simulation optimization methods in the literature review have a core iterative search based strategy that evaluates available information of past searches (if any) in each iteration, propose new candidate solution(s), and simulate these candidate(s).

The coverage property of the QSGM guarantees that all possible outputs from a QDES model are characterized. Using the coverage property, the boundaries of the feasible region of the simulation output can be precisely specified. When this region is specifies, all possible outputs of the simulation are in the feasible region. With the enhanced statistical analysis feature within the QDES framework from this dissertation, all possible output will have a probability of occurrence and the statistical values for all variables of interest. These new pieces of probability and statistics information will be able to provide the contour of the feasible region. Theoretically, with the feasible space and the contour of the feasible space defined, optimization of the QDES model can be formulated. The research for development of this feasible region and the contour of this feasible region will opens up new chapters in the area of simulation optimization techniques.

6.1.4 Sensitivity Analysis

One interesting future research coming out of this dissertation is in the area of sensitivity analysis. If the input time interval of the QDES model is reduced, we expect to see the same effect in the number of threads that QDES generates. The reduction effect of input time interval on the output interval width and other variables may be subject to variances of time interval width can be the subject of ongoing research.

REFERENCES

Agrawal, N.S. 2003. Breadth-First Algorithm for Qualitative Discrete Event Simulation. Oklahoma State University, Stillwater. Master of Science.

Allen, J. F., 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832-843.

Banks, J., J. S. Carson II, B. L. Nelson, and D. M. Nicol. 2005. *Discrete-Event System Simulation,* Fourth Edition, Prentice Hall, New Jersey.

Cellier, Francois E., Qualitative Modeling and Simulation: Promise or Illusion, *Proceedings of the 1991 Winter Simulation Conference*, p.1090.

Ingalls, R.G. 1999. Qualitative Simulation Graph Methodology and Implementation. University of Texas, Austin. Doctor of Philosophy.

Ingalls, R. G., D. J. Morrice, and A. B. Whinston. 1996. Eliminating Canceling Edges from the Simulation Graph Model Methodology. In *Proceedings of the 1996 Winter Simulation Conference,* ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 825-832. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Ingalls, R. G. 2002. Introduction to Simulation. In *Proceedings of the 2002 Winter Simulation Conference*, eds. E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Ingalls, R. G., D. J. Morrice, and A. B. Whinston. 2000. The Implementation of Temporal Intervals in Qualitative Simulation Graphs. *ACM Transactions on Modeling and Computer Simulation*, 10(2): 215-240.

Ingalls, R. G. and D. J. Morrice. 2004. PERT Scheduling with Resource Constraints Using Qualitative Simulation Graphs. *Project Management Journal*, 35 (3): 5-14.

Ingalls, R. G., D. J. Morrice, E. Yücesan, A. B. Whinston. 2003. Execution Conditions: a Formalization of Event Cancellation in Simulation Graphs. *INFORMS Journal on Computing,* 15(3)

Kabirian A., Olafsson, S. 2007. Allocation of Simulation Runs for Simulation Optimization. In *Proceedings of the 2007 Winter Simulation Conference*, S.G. Henderson, B. Miller, M.-H. Hsieh, J. Shortle, J.D. Tew, and R.R. Barton, eds. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Kuipers, B. 2001. Qualitative Simulation. *Encyclopedia of Physical Science and Technology*. R.A. Meyers, Academic Press: 287-300.

Law, A.M. and McComas, M.G. Pitfalls To Avoid In The Simulation of Manufacturing System. *Industrial Engineering*, May 1989, 21 (5): 28-32.

Law, A. M. and W. D. Kelton. 1991. *Simulation Modeling and Analysis,* Second Edition, McGraw-Hill Inc., New York.

Law, A.M. and W.D. Kelton. 2000. *Simulation Modeling and Analysis*, Mc-Graw-Hill.

Law, A.M. 2007. Statistical Analysis of Simulation Output Data Analysis: The Practical State of Art. In *Proceedings of the 2007 Winter Simulation Conference*, eds.S.H. Henderson, B. Miller, M.-H.Hsieh, J. Shortle, J.D. Tew, and R.R. Barton. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Nakayama, M.K. 2006. Output Analysis for Simulations. In *Proceedings of the 2006 Winter Simulation Conference*, eds. L.F. Perrone, F.P. Wieland, J. Liu, B.G. Lawson, D.M. Nicol, and R.M. Fujimoto. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Schriber,. T.J. and Brunner, T.B. 2007. In *Proceedings of the 2007 Winter Simulation Conference*, eds. Henderson, S.G., Biller, B., Hsieh, M.-H., Shortle, J., Tew, J.D. and Barton, R.R. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Schruben, L.W. 1983. Simulation Modeling with Event Graphs. *Communications of the ACM*, 26(11), 957-963.

Schruben, L. W. and E. Yücesan. 1993. Modeling Paradigms for Discrete Event Simulation. *Operations Research Letter* 13:265-275.

Som, T. K. and R. G. Sargent. 1989. Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs. *ORSA Journal on Computing* 1(2):107-125.

Yücesan E. and L. W. Schruben. 1992. Structural and Behavioral Equivalence of Simulation Models. *ACM Transactions on Modeling and ComputerSimulation* 2 (1): 82-103.

Ziegler, B. P. and S. Chi. 1992. Symbolic Discrete Event System Specification. *IEEE Transactions on Systems, Man, and Cybernetics* 22(6):1428-1443.

APPENDICES

VITA

Yen Ping Leow

Candidate for the Degree of

Doctor of Philosophy

Thesis: USING QUALITATIVE DISCRETE EVENT SIMULATION TO CALCULATE NEAR EXACT OUTPUT STATISTICS

Major Field: Industrial Engineering and Management

Biographical:

Personal Data:
Yen Ping Leow was born in Kuala Lumpur, Malaysia on June 7, 1977. She is the second child of five children born to Swee Meen Leow and See Moy Soo.

Education:
Completed the requirements for the Doctor of Philosophy in Industrial Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2008.

Master of Science in Industrial Engineering and Management
Oklahoma State University, Stillwater, Oklahoma
December 2002

Bachelor of Science in Mathematical and Computer Science
University of Adelaide, South Australia, Australia
June 2000

Experience:
Instructor, Research Assistant, Teaching Assistant

Professional Memberships:
Alpha Pi Mu Industrial Engineering Honor Society
Institute of Industrial Engineering
Institute for Operations Research and Management Sciences

Name: Yen Ping Leow Date of Degree: July, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: USING QUALITATIVE DISCRETE EVENT SIMULATION TO CALCULATE NEAR EXACT OUTPUT STATISTICS

Pages in Study: 145 Candidate for the Degree of Doctor of Philosophy

Major Field: Industrial Engineering and Management

Scope and Method of Study:

The scope of this dissertation is to develop a statistical analysis method for the Qualitative Discrete Event Simulation (QDES) methodology. The method presented shows the calculation of the probability of occurrence and the event time distribution for each event that is generated using QDES by imposing an assumption on the distribution of the delay intervals in the simulation, in particularly uniform distribution. The resulting calculations are used in calculating the simulation time-persistent output statistics and tally statistics for variables of interests.

Findings and Conclusions:

The Qualitative Discrete Event Simulation (QDES) methodology has been advanced as a decision tool and its capability has been enhanced to provide meaningful output information. A probabilistic approach to the statistical analysis of a QDES model has been developed. The advancement in the area of reasoning with probability to produce near exact output statistics from QDES will allow modelers to make wise decisions based on a single run, instead of sampling from multiple runs as in a regular discrete event simulation model. The near exact output statistics will be able to describe the distribution of different variables, such as the queue length, customer delay, server utilization, etc., together with information for the probability of these average values occurring for each variable and the event sequences that lead to these values. The probabilistic nature of our approach to calculating statistics from a QDES output opens up new chapter on creating a closed-form discrete event simulation output. The key behind the approach is that by imposing statistical distribution on the temporal intervals and using a complex set of conditional probabilities for the thread, the statistics in the QDES model is exactly represented.

ADVISER'S APPROVAL: ____