# Poisson Simulation as an Extension of Continuous System Simulation for the Modeling of Queuing Systems

**Leif Gustafsson**
Swedish University of Agricultural Sciences
P.O. Box 7032
SE-750 07, Uppsala, Sweden
*leif.gustafsson@lt.slu.se*

Poisson simulation is an extension of continuous system simulation whereby randomness is modeled as opposed to just adding noise. This article treats how Poisson simulation can be used for modeling queuing systems. The focus is on the implementation of queues in Poisson simulation and the connections to queuing theory. This approach also has theoretical and practical implications. Dynamic and stochastic systems, especially when queues are involved, are often treated by discrete event simulation using a microscopic view in which individual entities are modeled. Poisson simulation makes it possible to handle many such systems on a macroscopic level using aggregated states. It is therefore interesting to compare these approaches. Parallel approaches can then be sketched with discrete event simulation in one branch and Poisson simulation in the other. A fundamental difference between the approaches is whether one prefers to base a model on individual, distinguishable entities or on lumped entities.

**Keywords:** Continuous system simulation, discrete event simulation, Poisson simulation, stochastic process, queuing theory

## 1. Introduction

The main objective of this study is to demonstrate how queuing systems can be modeled and simulated within the framework of continuous system simulation (CSS). There are two prerequisites for this. First, CSS is extended with a general approach to include randomness. This extended form is called Poisson simulation (PoS) [1]. Second, queuing theory provides a unified way to model waiting line processes as well as ordinary dynamic processes. The reason for this is that when the number of servers in a queuing system goes to infinity (or becomes large enough), the wait for service is eliminated, and only the service process remains.

This article starts with a short example demonstrating how randomness can be modeled within the framework of CSS, thereby producing a PoS model. Then the implementation of fundamental queuing types in Poisson simulation is given. One strength of CSS, and thus also of PoS, is that the stochastic process is expressed in terms of differential equations within a well-defined mathematical framework. Likewise, queuing theory forms the basis for understanding queuing systems. Fundamental results of queuing theory, such as Erlang's loss formula [2] and Burke's theorem [3], are presented because they constitute a basis for understanding and have powerful implications.

Finally, similarities and differences in using PoS or discrete event simulation (DES) to model stochastic and dynamic systems are discussed. It turns out that many differences between PoS and DES often become of less importance, while the fundamental difference in aggregation of the entities remains.

## 2. Poisson Simulation

PoS is an extension of CSS to handle models when the entities are still aggregated and when it is crucial to model both dynamics and stochastics in a realistic way. In its simplest form, events happen randomly and independently, implying that the number of events during a time interval, $\Delta t$, becomes Poisson distributed. The method is presented in detail in Gustafsson [1] and thus is only demonstrated here by a simple example. A number of examples of different applications are given in Gustafsson [4].

### *2.1 Example: A Model of Radioactive Decay in CSS and PoS*

A CSS model is represented by a system of ordinary differential (and algebraic) equations. Each differential equation

*SIMULATION*, Vol. 79, Issue 9, September 2003 528-541

DOI: 10.1177/003759703040234

can be written as $dx/dt = f1(x, t) - f2(x, t)$, where $f1$ is the sum of inflows and $f2$ is the sum of outflows to the state variable $x$ during the time interval $dt$. This dynamic equation, together with an initial value equation, $x(0) = xo$, completely decides the behavior of $x(t)$ over time. In a CSS language, this dynamic equation can be written as a difference equation using Euler's approximation: $x(t + \Delta t) \approx x(t) + \Delta t \cdot [f1(x, t) - f2(x, t)]$.

For the sake of simplicity, this is demonstrated with a first-order system. This system has one state variable, $x$, representing the number of radioactive atoms, and one outflow rate, $f$, describing the number of radioactive decays per time unit. Therefore, $f$ is proportional to the state value $x$ with a proportionality constant, $p$, that describes the proportion of $x$ that is expected to decay during a time unit. A deterministic computer program model can then be written as follows:

```
   x = xo      [Initialization]
   Goto *
AGAIN:
   x = x + Δt · (−f)
*  f = p · x
   time = time + Δt
   if time ≤ Tend then Goto AGAIN
```

This model gives a negative exponential solution over time. It is an excellent description of a real system when the number of atoms is very large. However, when physical experiments are performed on a small or moderate number of radioactive atoms, stochastic variations, not described by the deterministic model above, are observed.

To model the *stochastics* in a realistic way (as opposed to just adding noise), we reason thus: the average outflow is still $f = p \cdot x$ per time unit or $\Delta t \cdot f = \Delta t \cdot p \cdot x$ during the time interval $\Delta t$. Since the properties of randomness, single events, and independence are valid, the number of events during the time interval, $\Delta t$, should be Poisson distributed with the intensity $\lambda = f = p \cdot x$. Thus, the outflow during $\Delta t$ has a Poisson-distributed variation denoted: $\text{Po}(\Delta t \cdot \lambda)$. The flow rate then becomes $\text{Po}(\Delta t \cdot \lambda)/\Delta t$. Therefore, the model is reformulated as

```
   x = xo
   Goto *
AGAIN:
   x = x + Δt · (−f)
*  f = Po[Δt · p · x]/Δt [The decay is now stochastic.]
   time = time + Δt
   if time ≤ Tend then Goto AGAIN
```

This model now correctly reveals the stochastic properties with regard to the number of atoms in state variable $x$ (see Fig. 1). [Note: If $xo$ is an integer, then $x(t)$ will remain an integer because of the Poisson mechanism.] □

PoS is a general method for modeling randomness within the CSS concept. The name *Poisson simulation* is, however, somewhat narrow. State variables in a PoS model are updated by inflows and outflows. Some flows may be deterministic and some Poisson distributed, and for other flows, random distributions of any kind may be used.

Assume, for example, that the physical experiment in the example above is repeated $N$ times. However, supplying a substrate with exactly 50 radioactive atoms is not a trivial task. Say that a substrate of exactly 2000 atoms, in which 2.5% are expected to be radioactive, is used. A binomial distribution ($\text{Bi}(n, p) = \text{Bi}(2000, 0.025)$ can then be used. This can be included by stating $x(0) = \text{Bi}(2000, 0.025)$ as an initial value.

In another experiment, taking 200 balls (of which exactly 25% are special) from an urn with 500 balls has to be described by a hypergeometric distribution because of no replacement. In still another example, experience may provide an empirical distribution to use.

When modeling process times other than exponential ones, the multinomial distribution plays the central role, which will be demonstrated later on.

## 3. Implementation of Queuing Systems in Poisson Simulation

### 3.1 Queuing Notation

In this article, Kendall's [6] "A/B/C notation" is used. Here, the letter $A$ indicates the interarrival time distribution and $B$ the service pattern, as described by the probability distribution for service time. (In first and second position, $M$ stands for Markovian, $D$ for deterministic, and $G$ for general.) $C$ is the number of servers (i.e., number of parallel service channels). When $C$ is infinite, any number of entities can be held without generating a queue. Finally, queue discipline and priority may be an issue in DES but not in PoS since all entities in a state variable are identical and indistinguishable. (In PoS, priority can be modeled by separate state variables in parallel.)

In the following, the main focus will be on the *M/M/n* queuing system. This system has random exponential interarrival times, random exponential service times, and $n$ servers in the system. Special cases are when $n = 1$ and when $n = \infty$. Just set $n$ to 1 to get one server or let $n \to \infty$ to get the "self-service" case whereby no waiting before service occurs.

### 3.2 Implementation

Implementation of queuing systems in PoS is straightforward. This is demonstrated here for the *M/M/1*, *M/M/n*, and *M/M/*$\infty$ cases. The input (arrivals) is then a Poisson ($\lambda$) process, so $\text{In} = \text{Po}[\Delta t \cdot \lambda]/\Delta t$.

Let $q$ denote the actual total number of queuing and served entities in the state variable $Q$. The output (departure after a mean service time of $1/\mu$) then has the following *kernel*: $\text{Out} = \text{Po}[\Delta t \cdot \mu]/\Delta t$ in the *M/M/1* case, $\text{Out} = \text{Po}[\Delta t \cdot \mu \cdot q]/\Delta t$ for the *M/M/*$\infty$ case, and Out

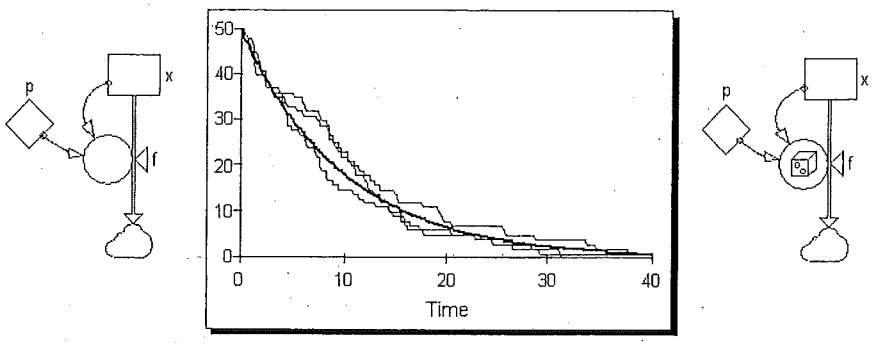


**Figure 1.** The deterministic continuous system simulation (CSS) model (left) and the stochastic Poisson simulation (PoS) model (right). Both are Powersim [5] diagrams of the codes above. The die in the outflow denotes that a random sample from some specified distribution is taken at each time step. Center: The results show how *x(t)* changes in one deterministic (bold line) and in three stochastic (thin lines) simulations of the models above. The state variable $x$ is initialized to 50 radioactive atoms, $p = 0.1$ and $\Delta t = 0.1$.

$= \text{Po}[\Delta t \cdot \mu \cdot MIN(n, q)]/\Delta t$ for the *M/M/n* case (where $MIN(n, q)$ means that at most $n$ entities can be served simultaneously, and if $q < n$, only $q$ entities are served).

However, there is a complication. The number of entities, $q$, in the state variable *may not become negative*. Since a state variable in a differential equation can take any value, CSS/PoS has no automatic mechanism protecting from negative numbers. Such a *guarding mechanism*, therefore, can be included, preventing the output from draining the state variable of more than its actual content. This can be accomplished by including another *MIN* function comparing the *kernel* (above) with the actual content. Thus, Out $= MIN[kernel, q/\Delta t]$.

The complete algorithm of a queuing system is then as follows:

$$
\begin{cases}
q(t + \Delta t) = q(t) + \Delta t \cdot (\text{In} - \text{Out}) \\
\text{In} = \text{Po}[\Delta t \cdot \lambda]/\Delta t \\
\text{Out} = \text{MIN}\{\text{Po}[\Delta t \cdot \mu], q\}/\Delta t \\
\qquad [M/M/1 \text{ case}] \\
\quad \text{or} \\
\text{Out} = \text{MIN}\{\text{Po}[\Delta t \cdot \mu \cdot \text{MIN}(n, q)], q\}/\Delta t \\
\qquad [M/M/n \text{ case}] \\
\quad \text{or} \\
\text{Out} = \text{MIN}\{\text{Po}[\Delta t \cdot \mu \cdot q], q\}/\Delta t \\
\qquad [M/M/\infty \text{ case}]
\end{cases}
$$

These formulas work nicely for small $\Delta t$. (The step size has to be adjusted to the dynamics of the queuing system.) In other words, all stochastic distributions approach their theoretical values when $\Delta t$ approaches zero.

### 3.2.1 Remarks

- In the *M/M/*$\infty$ case, the MIN guard can usually be dropped for reasonably small $\Delta t$—just like in the example of radioactive decay in section 2.
- It is often practical to increase step size, $\Delta t$, to reduce simulation time as long as dynamics are handled properly. Then several arrivals or departures may occur within a $\Delta t$, causing a problem connected to the nonnegative guarding mechanism. Assume, for example, that $Q$ is empty ($q = 0$) and that one arrival and one departure occur during $\Delta t$. If the arrival comes first, it will increase $q$ to 1, and the subsequent departure will reduce $q$ to 0. But if the departure comes first, there is nothing to take, so $q$ remains zero and the subsequent arrival will take it to 1. Both these cases have the same probability.

  However, the code given above will always add one arrival entity from the In statement, but the guard will prevent the departure (since $q = 0$ at the beginning of the time step), resulting in $q = 1$ as in the latter example. The zero state ($q = 0$) will then occur too infrequently. Alternatively, we could write the following: $\text{Out} = \text{MIN}\{\text{Po}[\Delta t \cdot \mu \cdot X], q + \text{In} \cdot \Delta t\}/\Delta t$ (where $X$ is adjusted for the *M/M/1*, *M/M/n*, or *M/M/*$\infty$ cases), resulting in $q = 0$, as in the former example. The zero state will then occur too frequently. A mechanism to handle both one arrival and one departure within $\Delta t$ could therefore be accomplished by including $\text{In} \cdot \Delta t$ with 50% probability.

  A more general mechanism, better for handling *multiple* arrivals and departures, could be achieved by the following: $\text{Out} = \text{MIN}\{\text{Po}[\Delta t \cdot \mu \cdot X], q + f(\text{In} \cdot \Delta t)\}/\Delta t$, where $f()$ is some integer-type probability function. This remains to be further investigated.
- The construction above assumes a single waiting line for all servers. If $n$ servers are to have separate waiting lines, $n$ parallel *M/M/1* queues are used.
- Note that if $q$ is initialized to an integer number, it will always remain an integer if only discrete distributions with integer outcomes, such as the Poisson distribution, are used.

In Figure 2, an *M/M/n* queuing system and some statistical devices are shown as a Powersim diagram. Below the queuing model in Figure 2, devices for statistics

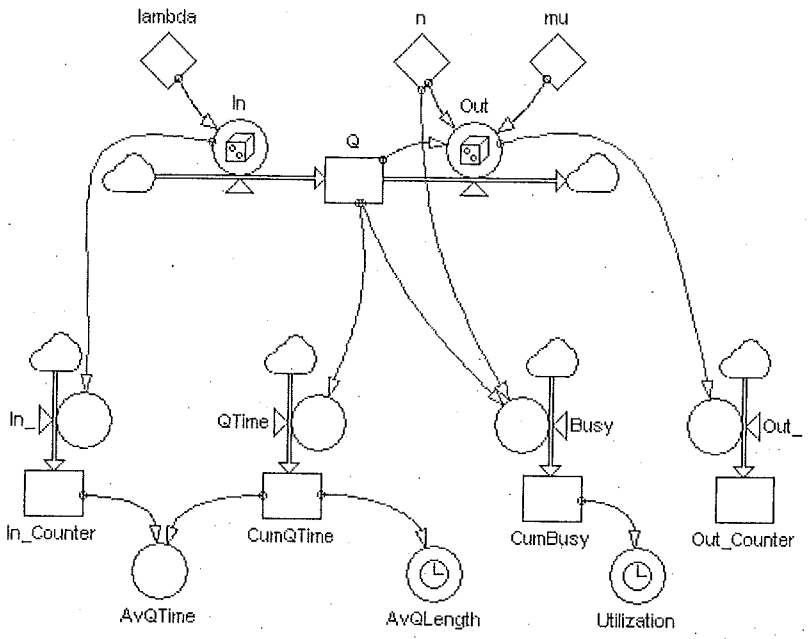


**Figure 2.** An *M/M/n* queuing system as a Powersim [5] diagram. *Q* is a state variable holding the total number of queuing and currently served entities. The arrival rate is λ (lambda), and the service rate (per server) is μ (mu). Of course, λ, μ, and also the number of servers, *n*, can be varied during the simulation. Underneath, devices for counting total numbers of arrivals and departures and for calculating total and average queue time, queue length, and server utilization during the simulation are shown. (A die in a symbol means that a random number from some specified distribution is drawn for each time step. A clock in a symbol means that time is involved.)

*within* the simulation are shown. The flows of arrivals and departures are accumulated over time in In_Counter and Out_Counter, respectively. The queuing customers are also integrated over time to CumQTime. By dividing CumQTime by In_Counter, the average time in queue, AvQTime, is obtained. By dividing CumQTime by time, the average queue length, AvQLength, is obtained. (Here, *Q* denotes both queuing and currently served entities. Alternatively, only the waiting entities [i.e., all but those currently served in *Q*] can be used in the calculations.) Server utilization is also calculated from the fraction of time the *n* servers are busy. In addition to these end results, all variables can, of course; be tabulated or displayed in a time diagram.

To estimate various statistical quantities such as mean, standard deviation, confidence intervals, or percentiles *between* simulations, the model has to be run many times. For example, to find the 5th and 95th percentiles of the average queue length at the end of a day, the model is run, say, 1000 times. Removing the simulations with the 50 smallest and the 50 largest end values gives the percentiles requested.

### 3.3 Conclusions

First, PoS can efficiently incorporate queuing systems. Second, all kinds of results can be collected during a simulation run, and statistical estimates can be obtained from the results of a number of PoS runs.

Varying intensities such as $\lambda = \lambda(t)$ cause no problems in PoS as opposed to DES. In DES, the model execution jumps to the next event of interest, which is usually scheduled in advance. Therefore, when the arrival intensity, $\lambda(t)$, varies over time, we get a problem. Say that the arrival intensity to a lunch bar is very low at 10 a.m., and thus the next arrival is scheduled to happen 3 hours later. But already at 11 o'clock, the intensity rises dramatically. Then the mechanism whereby each customer at arrival schedules his or her successor will not work. Thinning or some other mechanism then has to be used.

So far, the focus has been on Markov-type systems where the process time (as well as the arrival time) is exponential. In section 6, it will be shown that process times of any distribution can be approximated.

Although PoS was not invented for queuing studies, it is easy to include queues in such a model. This also means that PoS can handle some problems of *combined simulation*. Of course, it only uses lumped quantities, as opposed to the individual ones in DES. But it is straightforward to mix queue and nonqueue (self-service) processes in a PoS model.

## 4. Queuing Theory, Markov Models, and Poisson Simulation Models

Queuing theory is the theoretical basis for understanding the dynamics of entities waiting to be served. As mentioned above, queuing theory also applies to the self- service case. When a queuing model becomes too complex for an analytical approach, simulation has to be used.

### 4.1 Three Simple Types of Queues

The *M/M/n* queue system (including the *M/M/1* and *M/M/*$\infty$ cases) is crucial for understanding queue dynamics as well as stochastic equilibrium in steady state. It is, therefore, treated in some detail as a Markov model and as a PoS model for the cases with finite and infinite numbers of servers.

Start with a case in which entities arrive randomly, independently, and one at a time (a Poisson process). The mean arrival rate is $\lambda$ entities per time unit. The interarrival times are then exponentially distributed with a mean time between arrivals of $1/\lambda$. In queuing theory terms, the arrival process is then of Markov type.

The mean service rate *per server* is denoted by $\mu$. The service time interval at the state variable is also stochastic and exponentially distributed with the mean time $1/\mu$. The service time is thus also Markovian. This means that the probability of one server completing service of an entity during the very short time interval $\Delta t$ is $\mu \cdot \Delta t$. When the state variable holds $i$ entities, the probability of a departure during $\Delta t$, therefore, is $i \cdot \mu \cdot \Delta t$ up to the limit where $i$ exceeds the number of servers.

In Figure 3, the queuing system is presented (a) in PoS terms and (b) as a Markov model.

In Figure 3b, the same process as in 3a is modeled as a continuous-time Markov model. Here, each state $E_0, E_1, E_2, \ldots$ is described graphically, denoting 0, 1, 2, ... entities. The Markov process will go from $E_i$ to $E_{i+1}$ at the next arrival or from $E_i$ to $E_{i-1}$ at the next departure. During a short enough time interval $\Delta t$, only the probabilities of no arrival or departure, only one arrival, and only one departure have to be considered. Note that a "state variable" (often loosely called *state* in CSS and PoS) can hold any number of entities, whereas the concept *state* in queuing theory terms represents a specific number of entities. A finite or infinite number of states is therefore needed in a Markov model to represent one state variable in a Poisson model.

When dealing with *M/M/1*, *M/M/n*, and *M/M/*$\infty$ queuing systems, the analysis of a Markov process can be treated by using the theory of a *birth-and-death process* (see a fundamental book on statistics or queuing theory [7, 8]). We then regard the three cases of one arrival, one departure, and no arrival or departure during a short time interval. Thus, we have the following.

The probability of another arrival ("birth") during a short enough time interval $(t, t + \Delta t)$ is $\lambda \cdot \Delta t + o(\Delta t)$, and every transition due to an arrival has the same probability, $\lambda$.

If the system is in state $E_i$, the probability of a departure ("death") during the short enough time interval $(t, t + \Delta t)$ is $\mu_i \cdot \Delta t + o(\Delta t)$. (But $\mu_i$ is different in the *M/M/1*, *M/M/n*, and *M/M/*$\infty$ cases.)

The probability of no arrival and no departure during the interval therefore is $1 - (\lambda \cdot \Delta t + o(\Delta t)) - (\mu_i \cdot \Delta t + o(\Delta t)) = 1 - (\lambda + \mu_i)\Delta t + o(\Delta t)$.

Thus, the probability of ending up in the state $E_i$ at the end of the interval $(t, t + \Delta t)$ is the sum of these three possibilities. In probability terms, where $P_i$ represents the probability of being in state $E_i$ at the end of the time interval, we can write the following:

$$P_i(t + \Delta t) = P_i(t)[1 - (\lambda + \mu_i)\Delta t] + P_{i-1}(t)\lambda \cdot \Delta t + P_{i+1}(t)\mu_{i+1} \cdot \Delta t + o(\Delta t).$$

For $i = 0$, the second term on the right-hand side disappears. Rearranging the probabilities in terms of derivatives then gives the following set of differential equations:

$$\begin{cases} dP_0(t)/dt = -\lambda P_0(t) + \mu_1 P_1(t) \\ dP_i(t)/dt = -(\lambda + \mu_i)P_i(t) + \lambda P_{i-1}(t) \\ \qquad\qquad +\mu_{i+1}P_{i+1}(t) \qquad (i > 0) \end{cases}$$

To solve this system of equations analytically is problematic, but often the interest is focused on what happens when time approaches infinity. It can be shown that if the birth intensity is less than the death intensity, then $P_i(t)$ converges to a given value. (In the *M/M/*$\infty$ case, this is granted because the death intensity, $\mu_i$, increases with $i$, and the birth intensity, $\lambda$, remains constant for all states.)

For a stable system, therefore,

$$\lim_{t \to 0} P_i(t) = \pi_i, \quad (i = 0, 1, 2, \ldots),$$

where $\pi_i$ is independent of the initial state of the process at $t = 0$.

The next step is to calculate the equilibrium distribution $\pi = (\pi_0, \pi_1, \ldots)$ of the equilibrium probabilities $\pi_i$. Using the equilibrium condition $dP_i(t)/dt \to 0$ gives

$$\begin{cases} 0 = -\lambda\pi_0 + \mu_1\pi_1 \\ 0 = -(\lambda + \mu_i)\pi_i + \lambda\pi_{i-1} + \mu_{i+1}\pi_{i+1} \quad (i > 0). \end{cases}$$

Solving this system of equations recursively, starting from the first equation, gives

$$\begin{cases} \pi_1 = (\lambda/\mu_1) \cdot \pi_0 \\ \ldots \\ \pi_i = [(\lambda \cdot \lambda \cdot \ldots \cdot \lambda)/(\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_i)] \cdot \pi_0 \qquad (1) \\ \qquad\qquad\qquad\qquad\qquad\qquad (i = 1, 2, \ldots) \\ \ldots \end{cases}$$

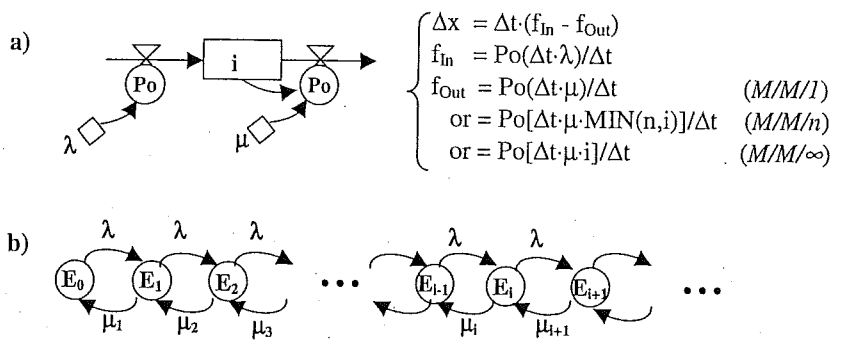


**Figure 3.** (a) The *kernel* of a queuing system as a Poisson simulation model with one state variable. For the *M/M/1*, *M/M/n*, and *M/M/*∞ cases, $f_{Out}$ differs somewhat; see also section 3.2 for guarding against outflow making the state variable negative. (b) The queuing system as a Markov model where transitions only exist between adjacent states. The service rates differ for different types of queuing systems. For an *M/M/1* queue, $\mu_i = \mu$; for an *M/M/*∞ queue, $\mu_i = i \cdot \mu$; and for an *M/M/n* queue, $\mu_i = i \cdot \mu$ for $i \leq n$ and $\mu_i = n \cdot \mu$ for $i > n$.

It remains to calculate $\pi_0$ from the requirement that the sum of the probabilities equals 1.

The *M/M/1*, *M/M/*∞, and *M/M/n* cases are treated separately in the following.

### 4.1.1 The M/M/1 Case

In this case, we have $\mu_i = \mu$ for all values of $i$. A stochastic equilibrium occurs for $\lambda < \mu$. Then, (1) gives

$$\pi_i = (\lambda/\mu)^i \cdot \pi_0 = \rho^i \pi_0 \quad (i = 0, 1, 2, \ldots).$$

Since all equilibrium probabilities have to sum up to 1, we get $1 = \pi_0 + \rho\pi_0 + \rho^2\pi_0 + \ldots + \rho^i\pi_0 + \ldots = \pi_0/(1-\rho)$, implying $\pi_0 = 1 - \rho$. Thus,

$$\pi_i = (1 - \rho)\rho^i \quad (i = 0, 1, 2, \ldots).$$

The equilibrium distribution is thus a geometric distribution for the *M/M/1* case.

### 4.1.2 The M/M/∞ Case

In this case, $\mu_i = i \cdot \mu$ because service is simultaneously given to all the $i$ entities. The equation system (1) can then be written as follows:

$$\begin{cases} \pi_1 = (\lambda/\mu) \cdot \pi_0 \\ \ldots \\ \pi_i = [(\lambda \cdot \lambda \cdot \ldots \cdot \lambda)/(\mu \cdot 2\mu \cdot \ldots \cdot i\mu)] \cdot \pi_0 \\ \qquad\qquad (i = 1, 2, \ldots). \\ \ldots \end{cases}$$

Substituting $\rho = \lambda/\mu$ gives

$$\pi_i = [\rho^i/i!] \cdot \pi_0 \quad (i = 1, 2, \ldots).$$

Now, regard a system with a finite number of states $E_0, E_1, \ldots E_K$, that is, where $\lambda_i$ and $\mu_i$ are zero for $i > K$. (This is the *M/M/n* case with so-called balking for $i > n = K$, which means that when all servers are occupied, additional entities will not enter. This complexity will not be treated here. The finite case is just treated for technical reasons, before letting $K$ approach infinity.) The result is the following:

$$\pi_i = \frac{\rho^i/i!}{1 + \rho + \rho^2/2! + \ldots \rho^K/K!} \quad (i = 1, 2, \ldots, K).$$

This is Erlang's [2] loss formula for $K$ servers in the case when only $K$ entities are allowed to enter the queuing system. (This formula was originally used in the theory of telephone traffic for the case when there are $K$ lines and further customers find the lines occupied and have to hang up.)

Multiplying numerator and denominator by $e^{-\rho}$ gives

$$\pi_i = \frac{\rho^i/i! \cdot e^{-\rho}}{1 \cdot e^{-\rho} + \rho \cdot e^{-\rho} + \rho^2/2! \cdot e^{-\rho} + \ldots + \rho^K/K! \cdot e^{-\rho}}$$

$$= \frac{p_z(i)}{p_z(0) + p_z(1) + \ldots + p_z(K)}, \quad (i = 1, 2, \ldots, K)$$

where the stochastic variable $Z \in \text{Po}(\rho)$ distribution (because the Poisson distribution has the mass function $f(i) = \rho^i/i! \cdot e^{-\rho}, i = 0, 1, 2, \ldots$). Since the denominator, representing the sum of all probabilities, is equal to 1, this formula gives the outcomes of a truncated Poisson distribution. Letting $K \to \infty$, the equilibrium probabilities $\pi_i = p_z(i)$ become Poisson distributed. Thus, the number of entities in the state variable in stochastic equilibrium is Poisson distributed.

In fact, Erlang's loss formula is also valid for *M/G/n*. Thus, the number of entities in a state variable in steady

state also has a Poisson distribution for the $M/G/\infty$ cases (i.e., it is Poisson distributed independently of the service time distribution).

This formula is of great importance for stochastic equilibrium, and it also has practical implications (e.g., for how to randomize initial values in a steady-state situation, which will be shown in section 5).

### 4.1.3 The M/M/n Case

A stochastic equilibrium occurs for $\lambda < n \cdot \mu$. In this case, $\mu_i = i \cdot \mu$ for $i \leq n$, and $\mu_i = n \cdot \mu$ for $i > n$. This is substituted into (1), which then can be solved recursively. After some calculations, the following formula is obtained:

$$\pi_i = \begin{cases} (\lambda^i/(i! \cdot \mu^i)) \cdot \pi_0 & (1 \leq i < n), \\ (\lambda^i/(n! \cdot n^{i-n} \cdot \mu^i)) \cdot \pi_0 & (i \geq n) \end{cases}$$

and

$$\pi_0 = \left[ \sum_{i=0}^{n-1} \lambda^i/(i! \cdot \mu^i) + \lambda^n/(n! \cdot \mu^n(1 - \lambda/(n \cdot \mu))) \right]^{-1}$$

In Figure 4, the stationary distributions for the three queue types are shown.

## 4.2 Networks and Burke's Theorem

A simulation model usually contains more than one block. The blocks are then linked in series and/or parallel. The fundamental case to be treated here is the serial one. In the parallel case, each serial path can be treated separately. In the case of several inflows to or outflows from a block, they can be added to a net inflow or outflow. This is also true for PoS, where Po(In1) + Po(In2) can be written Po(In1 + In2). Po(In1) here means that we draw a random value from a Poisson distribution with the parameter value In1. The discussion below is, however, valid only for feed-forward networks (i.e., networks of blocks for which entities are not allowed to revisit previously visited blocks).

In Figure 5, a PoS model with a series of state variables is shown. The arrivals are a Poisson process with the mean arrival rate $\lambda$ to the first state variable, and the mean service rates *per server* are $\mu_1, \mu_2, \ldots, \mu_m$. These service rates may be the same or different.

In the previous section, we treated stable *M/M/n* queuing systems (including *M/M/1* and $M/M/\infty$ cases). But what happens when several such systems are connected in series?

Intuitively, the structure around state variable $x_2$ is equivalent to that of $x_1$. However, the inflow ($F_0$) to $x_1$ is a Poisson process with the mean arrival rate $\lambda$, while the inflow $F_1$ to $x_2$ is the outflow from $x_1$. Although the steady-state flow through all the state variables has the average rate $\lambda$, the state variable $x_1$ is now varying stochastically, and the distribution of $x_1$ in stochastic equilibrium is different for the *M/M/1*, *M/M/n*, and $M/M/\infty$ cases (see Fig. 4). Therefore, one might expect different variations in $F_2$ than in $F_1$. It is thus necessary to find the output distribution (time between successive departures) from $x_1$, which is the input distribution (time between successive arrivals), to $x_2$. In addition, the question of dependency between the state variables has to be treated. However, paradoxically, the theory of series (or tandem) queues [3, 7, 8] has a simple and powerful answer.

Remarkably, it turns out that for a stable *M/M/n* queue (including *M/M/1* and $M/M/\infty$ cases), the steady-state output distribution is identical to the input one, that is, a Poisson process with the same departure rate ($\lambda$) as the arrival rate. It can also be shown that the output process is *independent* of the other processes in the system. Therefore, all state variables in a serial queuing system are *independent* *M/M/1*, *M/M/n*, or $M/M/\infty$ queues. This was first proven by Paul J. Burke [3] and is sometimes called *Burke's theorem*. A similar proof is given in Gross and Harris [8]; see also Kleinrock [7]. The proof of this theorem is somewhat complicated but very fundamental. In general, it opens for Jackson networks [7-9]. In our case, it extends the analytical understanding for PoS (and for DES) models of feed-forward type in stochastic equilibrium.

Another important result is that the joint probability that there are $n1$ entities at state variable 1, $n2$ at state variable 2, ... , and $nj$ at state variable $j$ is simply the product $p_{n1} \cdot p_{n2} \cdot \ldots \cdot p_{nj}$ because of independence between the state variables.

Note, however, that *M/M/1*, *M/M/n*, and $M/M/\infty$ queuing systems are the only ones that have the properties described above. This has to be kept in mind because not all PoS and DES models are of this type. In Kleinrock [7], the following is stated:

> In fact Burke's theorem tells that many multiple-server nodes may be connected ... together in a feed forward network fashion and still preserve the node-by-node decomposition. (Specifically, feedback paths are not permitted since this may destroy the Poisson nature of the feedback departure stream.) (p. 149)

In section 4.1, the stochastic equilibrium distributions for the number of entities in the state variables of *M/M/1*, *M/M/n*, and $M/M/\infty$ queuing systems were derived. These results can now be generalized to a series of queuing systems.

In the $M/M/\infty$ case, we know from above that $x_1$ has a stochastic equilibrium distribution described by the Poisson distribution with the parameter equal to $\lambda/\mu_1$. Since Burke's theorem says that a model such as the one in Figure 5 can be treated as $m$ independent queues, it means for the $M/M/\infty$ case that the equilibrium distribution of the state variable $x_i$ is Po($\lambda/\mu_i$). This will be applied in an example in section 5.

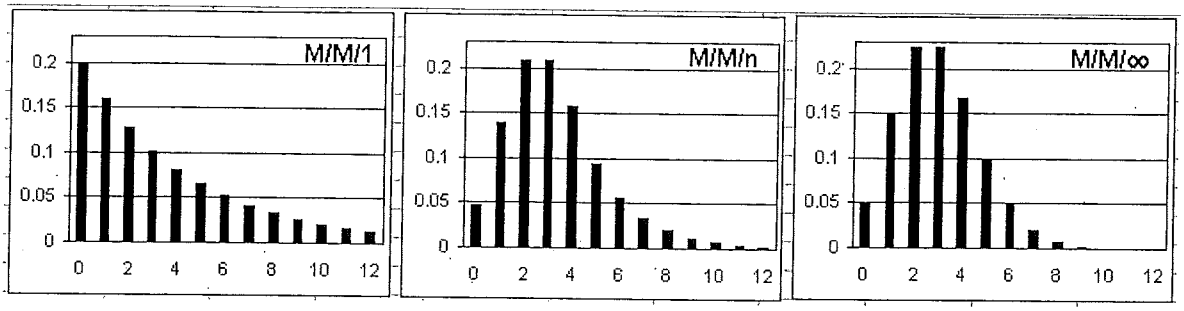


**Figure 4.** Example of stationary distributions of the state variable for the *M/M/1* case (geometric with $\lambda = 6$, $\mu = 7.5$), for the *M/M/n* case (something in between, with $\lambda = 6$, $\mu = 2$, and $n = 5$), and for the *M/M/∞* case (Poisson with $\lambda = 6$, $\mu = 2$).

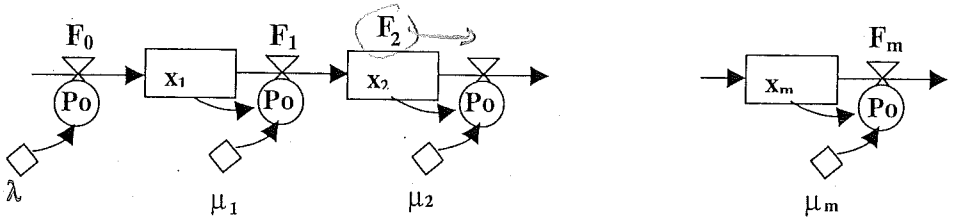


**Figure 5.** An *m*th-order system with the mean rates $\lambda$ and $\mu_1, \mu_2, \ldots, \mu_m$. (Note that $\lambda$ and $\mu_1$ to $\mu_m$ here denote parameters in a Poisson simulation model and not transition probabilities between states, as they did in Figure 3b, and the subsequent deductions in Markov terms in section 4.1.)

If Figure 5, instead, represents *M/M/1* blocks, the number of entities in the state variables $x_1, x_2, \ldots, x_m$ would all have geometric distributions in steady state.

Since the subsystems are independent and the flows are Poisson processes, one can even mix blocks of *M/M/1*, *M/M/n*, and *M/M/∞* types and still calculate the steady-state distributions for each state variable separately.

## 5. An Important Implication

### 5.1 Initial Value Problem

Choosing realistic initial values for state variables for a deterministic simulation model may seem trivial, but it often has a significant impact on the results. This problem becomes more complicated and important when the simulation model is stochastic. Using the theory developed in sections 4.1 and 4.2, this could be handled, for example, for PoS in common and fundamental cases.

### 5.2 Example: Modeling a Cohort Study

A common problem in medicine, epidemiology, biology, ecology, agriculture, and many other sciences is to understand and estimate the effects of some action called *exposure* that, via a dynamic process, causes some effects on outcome.

A dynamic model is then needed to understand and analyze the process, as well as calculate the damage or benefit from the exposure over time; a statistical treatment is also needed to estimate the variations and uncertainties. The problem with dynamic *and* stochastic systems is that stochastics excite the dynamics and dynamics change the statistical conditions. Therefore, the dynamics and the stochastics have to interact in one and the same model.

A standard approach is to compare an exposed *study cohort* with an unexposed or less exposed *reference cohort*. The results have to be interpreted in statistical terms because of finite numbers of participants in the two cohorts. To make it concrete, assume that the purpose is to study the effects of some exposure on a noncontagious disease such as cancer. Therefore, follow a study and a reference cohort randomized from the same population and where everything but the exposure is assumed to be the same. The cancer cases detected after a dynamic progress are then accumulated over a certain follow-up period.

Assume that the disease process can be described with a third-order model, as shown in Figure 6, and that the system is in a steady state when the experiment starts.

Here the hazard rate, F00, is caused by *background exposure* of the reference cohort. The study cohort also experiences the background exposure, but at a certain time

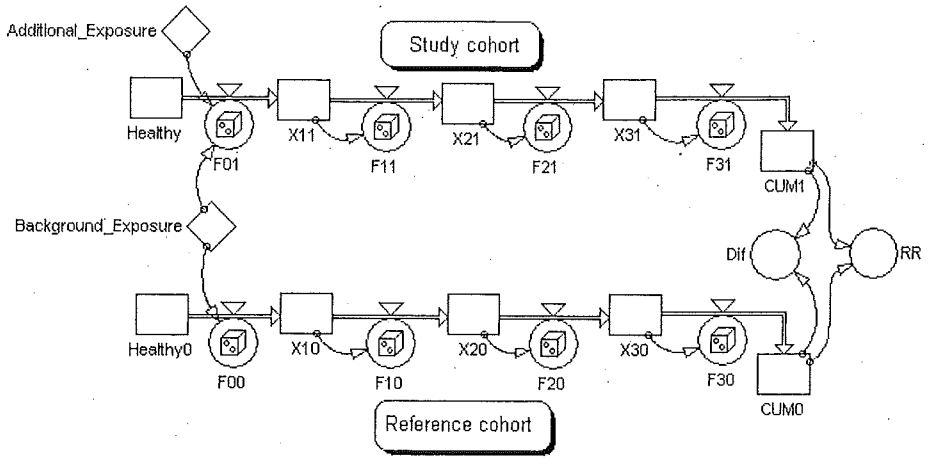


**Figure 6.** A model of the study and reference cohorts presented in Powersim [5]. (The time constants are included in the flow rates but are not shown to make the model more lucid.) Background and additional exposure are the driving forces, while the outcomes are the cumulative numbers of cancer cases from the reference cohort (CUM0) and from the study cohort (CUM1). Important measures of the study are the difference (DIF = CUM1 – CUM0) and the relative ratio (RR = CUM1/CUM0).

$t_0$, an *additional exposure* (of pulse, step, or other type) is added, causing the hazard rate F01.

The results of the study after a certain follow-up period are the cumulative numbers of cases from the reference and study cohorts. The difference (DIF) or the relative ratio (RR) are then estimates of the effects of the additional exposure.

Now to the problem. When the study starts at time $t_0$, the background exposure has long since caused disease. Therefore, the "pipeline" of state variables $X10$, $X20$, $X30$ and $X11$, $X21$, $X31$ is not empty at $t_0$. Furthermore, the randomization to study and reference cohorts implies that the numbers of cases in these state variables become stochastic variables. (Often the disease is not detectable until it surfaces [at F30 or F31], which is why it is not possible to get the initial values from experimental data. However, even if it is, we often have only one experiment and need to repeat the simulation a number of times with initial values drawn from proper statistical distributions to get correct estimates.)

Since the model will be used to interpret the experimental results, it is important that it is realistic. Setting the initial values of the state variables to zero would give a bias of the estimates. Setting them to (deterministic) steady-state values would eliminate this bias of the estimates but would give a too small variance between repeated simulations and thus too narrow confidence intervals. It is therefore important to randomize the initial values of the state variables from proper statistical distributions for each simulation run.

And now to the solution. Assuming a steady-state process up to time $t_0$ and that the cancers occur randomly and independently of each other (Poisson input), the results of section 4.1 say that $X10$ and $X11$ should be initialized by Po(steady-state value). This follows from ergodicity saying that time average (used in the deductions) equals ensemble average (of a number of simulations at $t_0$). Furthermore, Burke's theorem says that $X20$, $X30$ and $X21$, $X31$ should also be initialized in the same way. Thus, if the steady-state flow through the system is $\lambda$ and the time constant for the outflow of the $i$th state variable is $\mu_i$ (see Figs. 5 and 6), then $Xi$ should be initialized, drawing a sample from a Po($\lambda/\mu_i$) distribution.

Thus, it is often crucial to use realistic initial values in a model to get good estimates, and the theory presented above is of central importance in such cases. □

## 6. Modeling Process Times Other Than Exponential

So far, the focus has been on Markov-type systems in which the process (service) time (as well as the arrival time) is exponential. This is a field of central importance where also large theoretical support from queuing theory exists.

In the more general case, a process time may have any distribution such as uniform, triangular, normal, two peaks, or empirical. In such cases, there is no simple mechanism such as the Poisson mechanism for the exponential case. In principle, however, it is possible also to approximate the general case within the frame of CSS/PoS.

If possible, it is an advantage if a sufficiently good approximation of the process time can be obtained by the Poisson elements, described above, in series and/or parallel. But a more general approximation to produce a specific process-time distribution can be constructed in different ways. Such a structure must contain a series of states to be

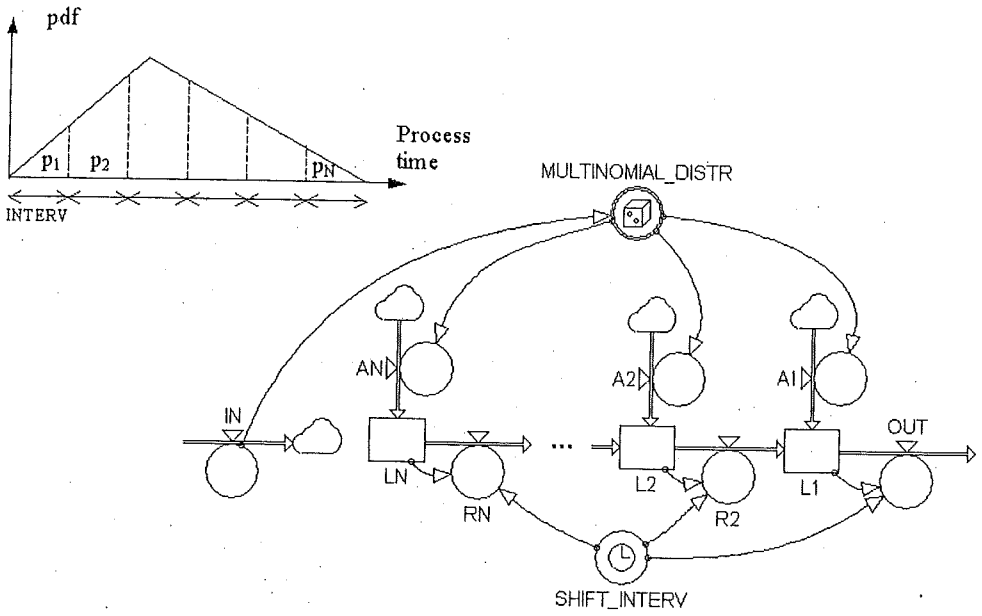


**Figure 7.** A structure in which the process time from any specified distribution can be approximated. The specified probability density function (pdf) of the process time (here triangular) is divided into $N$ successive time segments with the areas $p_1 \ldots p_N$. A multinomial random sample will then redistribute the incoming entities into the states $LN \ldots L1$, which performs the specified delay.

able to separate the processed entities that arrived at different points in time and still allow for different lengths of the process time drawn from a given distribution.

In Figure 7, one possible implementation of how a specified distribution can be approximately realized is shown. The probability density function (pdf) of the specified process-time distribution is subdivided into $N$ intervals, each of the length INTERV. The areas $p_1, p_2, \ldots, p_N$ of the pdf are estimated (where $\Sigma p_i = 1$). An arriving entity then has the probability $p_i$ to get a process time of $i \cdot$ INTERV. If $M$ entities arrive during an integration time step, they will be redistributed into the serial states $L1, L2, \ldots, LN$ according to $N$ random numbers drawn from a multinomial distribution, MULT($p_1, p_2, \ldots, p_N, M$). For each interval, INTERV (which is a multiple of the integration time step), the contents of the states $LN, LN-1, \ldots, L1$ are shifted forwards, releasing the arrival entities randomized to $Li$ after $i$ intervals.

The multinomial distribution, which delivers an $n$-tuple $a_1, a_2, \ldots, a_N$ (where $\Sigma a_i = M$) as an outcome, can be implemented as a macro or external function.

By increasing the number of states, $N$, the interval, INTERV, becomes shorter, and the accuracy of the process-time distribution will be more accurate.

The structure of Figure 7 can also be complemented with deterministic or random flows or mechanisms for customer behavior such as defection from the queue, jockeying among queues, or balking before entering a queue.

## 7. Differences and Similarities in CSS/PoS versus DES

### *7.1 Continuous System Simulation versus Discrete Event Simulation*

Two classical ways of modeling and simulating dynamic systems are CSS and DES. These are usually regarded as essentially different in almost all aspects.

For example, the worldview of CSS is a macroscopic one based on homogeneous flows of indistinguishable entities between state variables, whereas DES is based on a microscopic view where the single entities are unique.

In CSS, the continuous time of the system is modeled by very small time steps, $\Delta t$, while in DES, it is modeled by discrete events where attributes and conditions for one or several entities are affected.

The structure of CSS is that of a system of ordinary differential equations (approximated by difference equations), and the dynamics occur from integration over time. In DES, the basic structure is built on entities (actors) that compete for resources of various kinds over time. The DES mechanism behind this is a discrete event handler keeping track of the scheduled events to come. The logics are either implemented within the entities (e.g., Simula/Simulation [10], Simula/Demos [11]) or as a track-bound system with the resources as stations on which the entities travel (e.g., GPSS [12], SIMAN [13], Arena [14]).

The role of randomness also usually differs. In CSS, the macro view of many entities often eliminates the need for randomness. In DES, on the other hand, the micro view of individual entities makes randomness crucial. Therefore, the results also become of different types. In CSS, the results are typically aggregated numbers varying over a continuous time obtained from a single simulation. In DES, the results consist of statistical estimates of numbers of events, waiting times, queue lengths, utilization, and so forth based on a large number of simulations of the same model.

Due to the reasons mentioned, simulation languages for CSS and DES are so different in structure, time handling, mechanisms, result presentation, and other things that it is hard to see that they have much in common. Thus, there seems to be no obvious way to relate CSS and DES. On the other hand, both CSS and DES are used to replicate the dynamics of the same real world, although under somewhat different purposes and "worldviews."

Next, let us extend CSS to PoS and compare with DES how central aspects are handled.

### 7.2 System Aspects Modeled in Different Ways

The characteristics of a model are very much a result of how the essence of the real system is preserved in the model.

The *system* as a piece of the universe has a number of aspects such as *time*, *space*, *entities*, and their *attributes*. It also follows a certain *logic* that operates on the entities and their attributes in time and space. To describe and understand the system from a certain perspective, we have to build *theories/models*. Such a theory or model may describe aspects of nature (e.g., the laws of physics). In more aggregated terms, a physical theory lumps or aggregates into chemistry—which lumps into microbiology and into biology and ecology, and so forth—to capture something important of the system's nature. However, in physics, there is already a choice of regarding, for example, a system as a number of individual atoms or as atoms lumped (aggregated) into a unit. In Table 1, aspects of a system and of different model types are presented in these terms.

The choice of mapping the system into DES or CSS/PoS concepts is an issue of a micro (individual entities) or a macro (lumped entities) description.

The difference in time handling, using discrete events or time steps of size $\Delta t$, is of a technical nature. An input of random arrivals in DES can be generated by drawing random numbers from an exponential distribution to obtain interarrival times, whereas in PoS, the same is accomplished by drawing the random numbers from a Poisson distribution to get the number of arrivals during each time interval, $\Delta t$.

In DES, the structure of entities, attributes, and logics is very much preserved (after using Occam's razor to remove what is outside the scope). One effect of dealing with individual entities is that the temporal description can be sparse—only handling onset and end of activities in terms of events. The logics can be handled in different ways. Either they are a part of the entity, or they are implemented as a network of resources to be passed by the entities. (The former approach is more general and powerful. More about this is described in Kreutzer [15].) In both cases, the structure is a flowchart of stations to be passed by the individual entities.

In CSS/PoS, entities of the same kind with equal or similar attribute values are lumped into the same state variable; this variable only records their actual number. Thus, the way to discriminate between entities or their attributes is to place them in different state variables. The logics are that of the structure of the model (in terms of a system of equations or, equivalently, a structure of state variables and flows). The dynamic develops as changes of the number of entities in the state variables, in accordance with time integration of the differential equations (or difference equations, to be exact). Time is then handled in a quasi-continuous way. When randomness plays a minor role (e.g., because of large numbers), CSS can be used; otherwise, randomness can be modeled using PoS.

### 7.3 CSS/PoS and DES from a Queuing Theory Perspective

In queuing theory, we can study both waiting line models where entities are served by one or several servers and dynamic processes where external servers are not needed (self-service). This latter case can be handled in queuing theory by ascribing infinitely many servers [7, 8]. By extending CSS with the PoS ideas, it turns out that many problems treated by DES also can be handled in PoS. Then, in principle, a real system can be described by a dynamic and stochastic model using DES or PoS. DES models are to be used when each entity is modeled separately, and PoS is used when the entities are lumped within state variables. Dropping the randomness in PoS then gives CSS (see Fig. 8).

### 7.4 Structural Similarities for DES, CSS, and PoS

DES, CSS, and PoS models all have a structure of *blocks* that should be passed by entities in accordance with some logics.

In DES models, the fundamental block is the (server) *station* handling the activity between related start and end events (an *advance block* between seize and release in GPSS [12], some form of the *operation block* in SIMAN [13], a *process block* in Arena [14], or an *activity block* in a Simula/Demos activity diagram [11]). The inputs and outputs are arrivals and departures of individual entities (or batches of individual entities).

In CSS/PoS models, the block is a *state variable* holding a number of identical entities. In PoS, the numbers of entities are (usually) *integers*. In both CSS and PoS, each block has input(s) that come from outside the scope of the model or from other blocks and output(s) that lead to other blocks or out of the model boundaries.

**Table 1.** System, DES, and CSS/PoS models presented in terms of time, space, entities, attributes, and logics

| | System | DES | CSS/PoS |
|---|---|---|---|
| Time | Continuous | Discrete | Quasi-continuous |
| Space | Continuous | Not modeled[a] | Not modeled[b] |
| Entities | Discrete or lumped | Discrete | Lumped in state variables |
| Attributes | Discrete or continuous | Discrete or continuous | Lumped in state variables |
| Logics | "Laws of nature" | Program logics | Structure of state variables and flows[c] |

DES = discrete event simulation; CSS = continuous system simulation; PoS = Poisson simulation.
[a]Or described in terms of entities or attributes
[b]Or described in terms of number of entities in state variables
[c]Or, equivalently, a system of differential equations

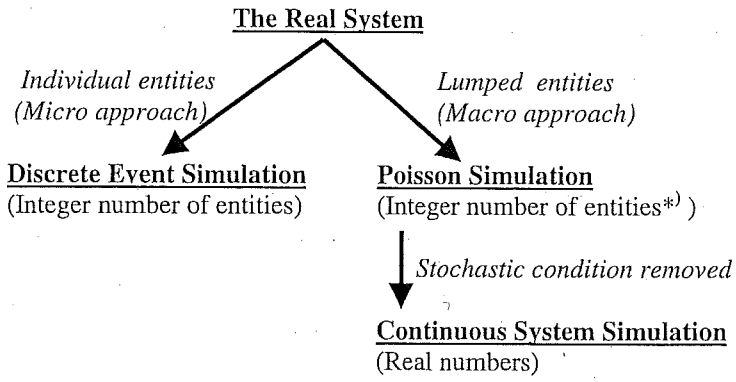


**Figure 8.** The relationships between discrete event simulation, Poisson simulation, and continuous system simulation, as seen from queuing theory.* Poisson simulation may also use continuous probability distributions, whereas the state variable contents no longer have to be integers.

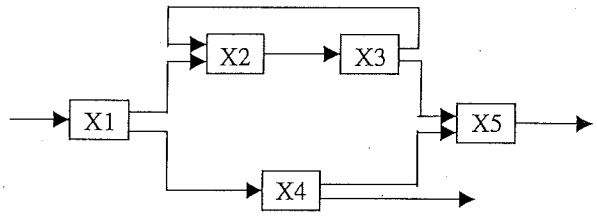


**Figure 9.** An example of a structure of interrelated blocks. There may be external inputs and outputs, and the blocks may be interconnected serially or in parallel. This is accomplished by splitting the output from a block or by summing up inputs from parallel blocks.

Such a structure of DES, CSS, or PoS models can be described as a number of linked blocks such as those in Figure 9.

In Figure 9, there is an external input to block $X1$ and external outputs from blocks $X4$ and $X5$. The output from $X1$ is split into parallel sequences $X4$ in one branch and the serial blocks $X2$ and $X3$ in the other. Part of the outputs from $X3$ and $X4$ are summed up as inputs to $X5$. Another part of the output from $X3$ is fed back to $X2$. All structures can be modeled using these concepts. The implementations in different types of simulation are, however, somewhat different. In DES, a logical condition is used for splitting, whereas in CSS/PoS, different fractions of the state variable content may lead from one state to others.

Queuing theory can be used to describe a Markov-type process when the block has a finite number of servers, so the entities may have to wait for a service, or when it has an infinite number of servers. It has been demonstrated that a queuing block with a limited or infinite number of servers can easily be implemented in PoS, as shown in section 3.2. Inversely, it is also possible to implement self-service in DES by setting the number of servers to a large enough number.

## 8. Discussion

The introduction of PoS gives a way to extend CSS to the stochastic domain by providing a method to *model* randomness, as opposed to just adding input, state, or output noise.

The power of PoS in the field of dynamic and stochastic modeling has been demonstrated [1, 4]. In this article, it is shown how *M/M/n* queues can be implemented in a PoS model, but other process-time distributions can also be handled, as shown in section 6.

In this context, queuing theory is valuable as a mathematical background for understanding fundamental queuing systems. The perspective using queuing theory also has practical implications for both the self-service and the limited resources cases. In simple cases, Erlang's loss formula [2], Burke's theorem [3], and their applications in networks have direct practical implications. In more complex cases, queuing theory may still be the prerequisite for a deeper understanding and interpretation of the results.

An example of how to randomize initial values for the state variables was shown for a model in which a study cohort and a reference cohort were differently exposed (see section 5).

Another example of a practical implication may be whether the assumption of Poisson-distributed *arrivals* is consistent with what is known about *departures*, which can be monitored a number of state variables downstream. If the process between arrivals and departures fulfills the requirements of Burke's theorem in a steady-state case, it is a necessary condition (that can be tested) that the departures are also Poisson distributed.

Many natural processes such as aging, disease, metabolism, and so forth are of the self-service type, while other processes such as treatment, transports, and so forth require external resources that, when lacking, cause queues of waiting entities. The tradition of using CSS for the former and DES for the latter case is not compulsory. When an individual description is not necessary, the lumped approach of PoS has a number of advantages.

PoS is based on differential equations that are mathematically well understood. Furthermore, a lumped model is usually much simpler to build and handle, and it usually runs many times faster than a corresponding DES model. It also requires far less data. Model fitting and optimization can be done in a fraction of the time required for a corresponding DES model. Validation is also much simplified. Finally, handling time-varying intensities of arrivals and departures in PoS is straightforward, but it requires special attention in DES.

A practical problem is that CSS languages usually lack powerful mechanisms for repeated simulation and statistical analysis. Statistical estimates require a number of runs of the model with a subsequent analysis of mean, variance, and other statistical measures of the results from these runs. This problem is, however, easy to remove. In a project, supported by a grant from the Swedish Council for Planning and Coordination of Research, a device for this has been constructed [16].

From a queuing theory perspective, important parallels between the DES and CSS/PoS worlds can be made visible. This opens up a choice of how to build a dynamic and stochastic model of a system from the purpose of the study. Besides the technical differences, there remains a fundamental difference between DES and PoS. In DES, the entities are individuals, while in PoS, they are lumped and not distinguishable within a state variable. The main focus, then, is on the merits of an individual contra a lumped description of entities. When the objective is to focus on individuals and their life stories, DES is a natural choice, while great simplifications can be obtained if a lumped PoS model is adequate.

The presented approach might open up the simultaneous use of ideas from both CSS and DES worlds within one model. In the field of combined simulation, submodels of both CSS and DES types are integrated into one model. This requires a special simulation language (e.g., DYNAMO in Simula [17] or DISCO [18] or jDisco [19]) with accompanying complications such as two types of time handling. Here PoS might be an option.

For example, in telecommunications, individual packages are sent over huge networks. Sometimes, individual packages have to be followed, but the network has to be described in lumped terms because of very large numbers. Can this be treated in terms of interfacing an individual and a lumped description? Or can it be handled by parallel descriptions, where the lumped world of the net interacts with the individual description of a limited number of packages?

The full theoretical and practical implications of the ideas presented in this article are still to be investigated.

## 9. Acknowledgments

The author wishes to acknowledge Kjell Pernestål and Mikael Sternad for valuable discussions and proposals. An ongoing project on Poisson simulation is supported through a grant from the Swedish Research Council for Environment, Agricultural Sciences, and Spatial Planning.

## 10. References

[1] Gustafsson, L. 2000. Poisson simulation—A method for generating stochastic variations in continuous system simulation. *SIMULATION* 74 (5): 264-74.

[2] Erlang, A. K. 1917. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *Electrotekniken* 13:5-13 (in Danish). (English translation in *P.O. Elec. Eng. J.* 10 (1917-1918): 189-97.)

[3] Burke, P. J. 1956. The output of a queueing system. *Operational Research* 4:699-714.

[4] Gustafsson, L. 2004. Studying dynamic and stochastic systems using Poisson simulation. In *Micro-meso-macro: Addressing complex systems couplings*, edited by H. Liljenström and U. Svedin. Singapore: World Scientific Publishing.

[5] Powersim Corporation. 1996. *Powersim reference manual.* Herndon, VA: Powersim Corporation.

[6] Kendall, D. G. 1953. Stochastic processes occurring in the theory of queues and their analysis by the method of imbedded Markov chains. *Annals of Mathematical Statistics* 24:338-54.

[7] Kleinrock, L. 1975. *Queueing systems: Vol. 1. Theory.* New York: John Wiley.

[8] Gross, D., and C. M. Harris. 1998. *Fundamentals of queueing theory*. New York: John Wiley.
[9] Jackson, J. R. 1957. Networks of waiting lines. *Operational Research* 5:518-21.
[10] Birtwistle, G. M., O. J. Dahl, B. Myrhaug, and K. Nygaard. 1973. *SIMULA BEGIN*. Philadelphia: Auerbach/Studentlitteratur.
[11] Birtwistle, G. M. 1979. *DEMOS: A system for discrete event modelling on Simula*. London: Macmillan.
[12] Bobillier, P. A., B. C. Kahan, and A. R. Probst. 1976. *Simulation with GPSS and GPSS V*. Englewood Cliffs, NJ: Prentice Hall.
[13] Pegden, C. D., R. E. Shannon, and R. P. Sadowski. 1995. *Introduction to simulation using SIMAN*. New York: McGraw-Hill.
[14] Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 1998. *Simulation with Arena*. New York: WCB/McGraw-Hill.
[15] Kreutzer, W. 1986. *System simulation: Programming styles and languages*. Reading, MA: Addison-Wesley.
[16] Gustafsson, L. 2003. Methods and tools for statistical handling of Poisson simulation. Unpublished manuscript.
[17] Hegna, H. 1974. *DYNAMO in Simula 67—A rough outline of a simple implementation: Simula information*. Oslo: Norwegian Computing Center.
[18] Helsgaun, K. 1980. DISCO—A SIMULA-based language for combined continuous and discrete simulation. *SIMULATION* 34 (7): 1-12.
[19] Helsgaun, K. 2001. *jDisco—A Java framework for combined discrete and continuous simulation*. Roskilde: Roskilde University.

***Leif Gustafsson*** *earned his Ph.D. in automatic control at Uppsala University, Sweden, in 1977. He is now a professor of systems analysis at the Swedish University of Agricultural Sciences. He is presently developing combined dynamic and stochastic methods and tools. His main interest areas are in the fields of systems analysis, modeling, simulation, and optimization. He is also working with a number of applications in agricultural and biological sciences and in epidemiology.*