Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

SIMULATION WITH STOCHASTIC PETRI-NETS

Vitali Volovoi

Independent Consultant
505 Birchington Close
Alpharetta, GA 30022, USA

ABSTRACT

This tutorial reviews the role of Stochastic Petri Nets (SPNs) in stochastic simulation. The evolution of SPNs as a component-level state-space modeling framework is discussed. SPNs are compared to both process-based approaches to discrete event simulation (DES) and to agent-based modeling (ABM). The causes for the apparent lack of commercial success of general-purpose simulation with SPNs are analyzed along with the possibility that this situation will change in the near future. In particular, the potential is explored for SPNs to serve as a useful compromise between traditional DES and the more flexible (yet lacking standard building blocks) ABM. To this end, SPNs can be considered as a middle-ground framework with natural capabilities for modeling complex interactions among the entities comprising the system.

1 INTRODUCTION

The subject of this tutorial is stochastic computer simulation in the context of Operational Research and Management Science (OR & MS). SPNs can be placed at the intersection of two important branches of science and engineering: analysis of stochastic processes by means of generation of sample paths (realizations) of the processes (Henderson and Nelson 2006) on the one hand, and a graphical mathematical modeling language called Petri nets (Petri 1962) on the other. Petri nets are popular in computer science as they provide a well-structured and visual means for analyzing complex interactions among components, such as synchronization and concurrent operations. SPNs have been around for over thirty years (Marsan 1990), but to date their implementations have been mostly limited to academic tools, or to the tools used internally by large engineering companies (such as Siemens or Total), since practicing engineers tend to find them too abstract and difficult to understand. SPNs and the related Timed Petri Nets (not to be confused with Time Petri Nets!) come in many flavors, which further complicates the ability to assess the utility of SPNs in practical simulation applications (Bowden 2000).

This tutorial aims to put the advantages and drawbacks of SPNs in the context of general simulation frameworks. First, a brief review of various approaches to stochastic simulation is provided, along with the assessment of SPNs from this high-level perspective. This is followed by a more detailed look at the evolution of the variations of SPNs and the driving forces behind their evolution. Several examples of SPN models are provided to illustrate the relevant differences and similarities between then and other modeling frameworks.

1.1 Simulation Frameworks

Historically, an important watershed in classifying simulation types in OR & MS was the choice of the underlying solution method with stochastic simulation based on sample paths (state trajectories) being contrasted with solutions of continuous differential equations. Discrete Event Simulation (DES) and System Dynamics (SD) fall under the former and the latter categories, respectively (Henderson and Nelson 2006). As will be discussed below, this dichotomy is closely mirrored within the historical development of SPNs. Another perspective consists of specifying a “world view” for the modeling approach. The following three views are generally considered (Law and Kelton 2000): process interaction, activity scanning, and event scheduling. The latter two views are effectively related to the actual algorithms implemented in modern simulation software (the "back end"). In contrast, the process-interaction view can be considered as a front end of simulation that needs to be converted to a back-end view. Event graphs (Schruben 1983) use the event-scheduling view directly for model building, which significantly simplifies the connection between the front and back ends, but also requires a high degree of modeling abstraction. The process-driven and event-driven views are combined in Kesaraju and Ciarallo (2012) by augmenting event graphs with the concepts of entities in an attempt to reduce the level of modeling abstraction. The activity-scanning world view corresponds to the listening mode in computer languages, and is relevant in the presence of exogenous events, the timing of which cannot be predicted in advance. However, in the context of stochastic simulation, candidate future events are usually endogenously generated, and as a result, the event-scheduling view generally provides signifiant computational advantages compared to the activity-scanning view.

The process-interaction view underpins the vast majority of modern DES software. Reflecting the manufacturing applications' heritage, the underlying idea is to model flows of entities through a network of servers that transform those entities (transactions) by changing their attributes. This view "implies system events by describing what happens to an entity as it encounters the system" (Nelson 2002). A certain degree of hierarchical (top-down) perspective is inherent in this view: there are transient low-level entities (jobs or transactions) and higher system-level persistent entities (such as resources) (Roeder 2004).

In contrast, in SPNs the system is comprised of individual components (entities) that can change their states, thus triggering or preventing the state changes of other entities. In other words, SPNs are based on local (peer-to-peer or bottom-up) interactions among entities, and the concepts of resources and queues are derived, rather than being fundamental properties, as they represent specific examples of coordination (interaction) among individual entities. Adopting the language of agent-based simulation, models utilize the point of view of "agent activities" rather than business processes (Bonabeau 2002), and as such can be considered a particular restricted type of agent-based models. Here an activity is interpreted as a time delay between state transitions for a given entity (agent). Indeed, referring to the table of distinct attributes of ABM and DES (Table 1 in Siebers et al. (2010)), the first six out of seven attributes of SPNs would fall under the ABM column, and the last one, related to the source of data, is arguably application-specific rather than an intrinsic attribute of the modeling framework.

Another related viewpoint considers traditional DES as primarily designed to deal with so-called "transformational" systems, where the entities are transformed (processed). These systems are contrasted with "reactive" systems, where the entities are event driven and continuously reacting to external and internal stimuli (Harel 1987); here the non-hierarchical, peer-to-peer nature of SPNs is more appropriate. Correspondingly, there are several fields where SPNs have enjoyed relative success: flexible manufacturing systems (Marsan et al. 1995); system reliability (Signoret et al. 2013); and business processes and workflow management (Aalst 1998) (the timed Petri nets employed in this area are distinct from SPNs, but the resulting models can be easily cast in terms of SPNs that are discussed in this tutorial).

There is a lack of consistency in assigning Petri nets to a world view (Roeder 2004). In accordance with Miller et al. (2004), Markov chains represent a state-based view, while Petri nets fit either the process-interaction or activity-scanning world view. However, as described below, it is logical to view Petri nets as a local, component-based equivalent of Markov chains, both providing state-based world views. This is consistent with the view of building blocks of DES as a set of state variables that characterize the modeled system and events (points in time when at least one of those state variables changes), with the assumption that no changes to the system occur between those events (Ross 2002). An analogy can be drawn between the global-local juxtaposition in stochastic state-space modeling (Markov chains vs SPNs, as discussed below) and in simulation (traditional process-focused DES vs. ABM). As commercial tools evolve to satisfy the needs of today's customers , the boundaries between the frameworks are blurred, and traditional DES frameworks allow the modeling of some agent-based behavior (Brailsford 2014) (an example discussed therein is revisited below). In particular, state-space representation is utilized for persistent entities (such as resources), albeit without standard graphical means to depict the dynamic interactions of such entities.

1.2 Markov chains

State-space diagrams provide a convenient graphical way of depicting the behavior of nondeterministic systems. Markov chains are the simplest and most popular kind of state-space diagrams, with applications ranging from their original use by Andrey Markov (for modeling the relative frequencies of vowels and consonants in Alexander Pushkin's novel *Eugene Onegin*) to evaluating web page ranks (in the PageRank algorithm invented by Google's founders) (Hayes 2013). Despite their deserved popularity, Markov chains are prone to "state-space explosion"—they scale poorly as the number of system entities increases.

Let us consider a simple example illustrating the issue (Volovoi 2013) of a household consisting of two family members (later referred to as customers), m_1 and m_2, and a car, c_1. Each customer can be in one of three states: not needing a car (N), driving a car (D), or waiting for a car (W); we assume that being a passenger in the car qualifies as state N. The inputs to this model could include the usage pattern of each customer, *e.g.,* the frequency and duration of trips, as well as the car properties, *e.g.,* the frequency of breakage and the duration of the repairs. The outputs would be the "performance" measures or metrics of this "system," such as the frequency and duration of unsatisfied demand. These measures can help to make educated decisions about changes to the system, *e.g.,* whether it makes sense to get a second car. First, let us consider a situation where the car can be in one of two states: idle (I) or "in use" (U). This "system" consists of three entities (two customers and a car), so there are $3 \times 3 \times 2 = 18$ possible permutations of the components' states that can define the state of the entire system. Not all of those permutations constitute feasible system states, so a Markov model would have only five rather than 18 states. Fig. 1, A) shows the corresponding Markov chain diagram. Here the state of each entity is represented by the corresponding capital letter (so, for example, the DNU state of the system implies that first customer is driving, the second is not needing the car, and the car is used). Introducing the possibility that the car can be broken (be in state B) extends the size of the diagram to nine states (as shown in Fig. 1, B). For a family of four with two cars, the number of possible states is 111, and for a family of five with three cars, that number is 589. If we consider a fleet of 10 cars with 20 customers, the number of states is $451,417,560,951$—or over 451 billion (Volovoi 2013).

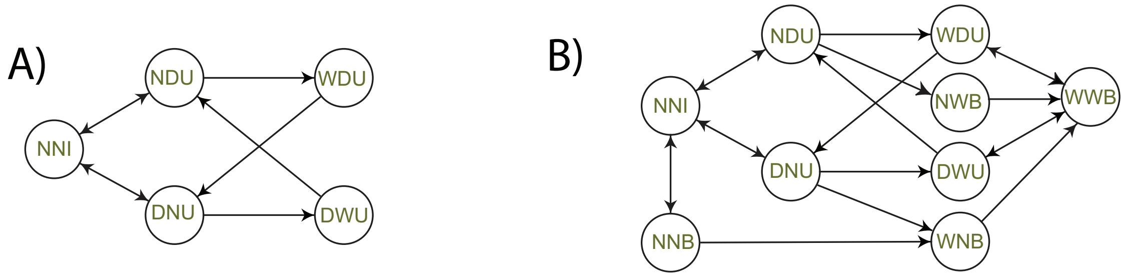

Figure 1: A) Markov chain diagram for two customers and a car, B) including the broken car possibility

This rapidly growing state-space size can be avoided using symmetry considerations if the customers or the cars are not distinguished among themselves. Indeed, for two customers and one car, the number of states reduces to three and six for Fig. 1, A) and B), respectively, as states such as *DNU* and *NDU* can be merged together. However, very often one does want to track individual performance (*e.g.,* Dad might need the car more often for longer trips, and Car A is old and breaks more often). On the other hand, the state space will be even larger if we want to distinguish which car is driven by a given customer: if there are q cars driven at any given time, there are potentially $q!$ possible combinations, although only some of those combinations might be feasible (for example, one can have a list of preferred cars for each member of the family and track how their preferences are satisfied).

2 Petri nets

The above example represents a general problem with customers generating service demand and cars representing entities (servers) that service (satisfy) the demand. The meaning of both customers and servers is domain-dependent. A partial listing of those pairs includes:

- Packets of information (customers) and computer servers or routers (servers) in computer applications
- Tasks (customers) and resources (servers) in business processes
- Callers (customers) and tellers (servers) at the call centers
- Passengers or freight (customers) and transportation and storage resources (servers) in logistics

Matching demand and supply is at the core of modeling needs with SPNs (Marsan 1990).

Noting that alphabetical notations based on the components' states were used for identifying a system state, it is natural to employ some sort of equivalent of an alphabet in a graphical description of the system. Petri nets effectively implement this idea by modeling the states of individual components rather than the explicit states of the entire system. In Petri nets, Markov chain-state diagrams are complemented by two new types of objects: first, small filled circles (called tokens) denoting individual components are introduced, each placed inside of one of the larger hollow circles that denote the potential states of those components (the latter entities are named "places" as opposed to "states" in Markov diagrams). Second, in order to model interactions among components, the tokens are routed among places via intermediate stops or junctures, called transitions, which are denoted with solid rectangles. Any two places cannot be connected by an arc directly; instead they must be connected through a transition. The number of input and output arcs does not need to coincide, enabling the merging and splitting of tokens and their routes. The timing of state changes can be modeled by specifying time delays for transition "firing," an atomic (*i.e.,* indivisible) action that removes tokens from all input places for the transition and deposits tokens into its output places. Such Petri nets are timed Petri nets, or, more specifically, Stochastic Petri Nets (SPNs) (Balbo 2007, Haas 2002), when delays can be nondeterministic and follow a specified distribution. In simulation, no limitations on the associated types of distributions are needed, but historically SPNs referred to models with exponentially distributed delays only, so that they could be converted to Markov chains and solved using appropriate techniques for the underlying differential equations.

Fig. 2 depicts an SPN for two customers and one car. Using standard notations for SPNs, thin rectangles denote immediate transitions that incur no delays. We note that the SPN in Fig. 2, A) consists of three groups corresponding to each component of the system, and if there are tokens in the places "car needed" and "car available," those two tokens are merged into a single token deposited into the place "car used." This simple model reflects the fundamental feature of SPNs in modeling the coordinated behavior of system components: tokens that represent the car and the driver are merged into a single "car-driver" token while driving takes place, and split into separate tokens again when the driving is complete. The model has eight states (places), but clearly the complexity of the model is determined not only by the number of places but also by the number of transitions (eight), connecting arcs (20), and tokens (3). In fact, this model does not take into account the possibility that the car can break while driving (so all trips are completed); to include this possibility, two extra transitions are added, as shown in Fig. 2, B). Now we have ten transitions and 26 arcs (the corresponding Markov model has 23 arcs).

Such SPN models scale better than Markov chains (which explains the fact that they were originally used as pre-processors for creating Markov chain models (Trivedi 2002) to be solved using differential equations). Indeed, there would be three places required for each customer and two places for each car, so the number of places in the model with 20 customers and 10 cars would be only 80, which is certainly an improvement over 451 billion states. However, the web of connecting arcs would be so convoluted that the resulting model is still too complex to be of practical use for visually conveying system behavior. (Referring back to Fig. 2 B), one can envision 20 segments of the net similar to the two depicted at the top

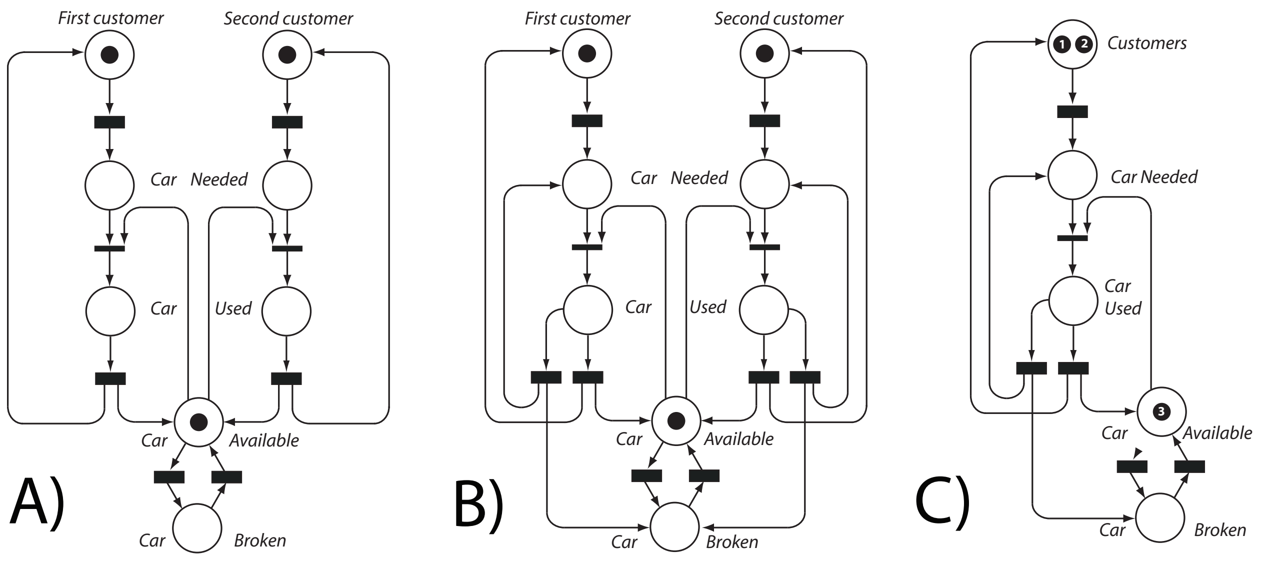

Figure 2: A) SPN diagram for two customers and one car (no breaks), B) SPN diagram with breaks, C) Colored SPN diagram equivalent to B) with integer labels (colors) directly shown inside tokens

of the net, and 10 segments similar to the one to the bottom, with each of the 10 segments at the bottom being connected to each of the 20 segments at the top.)

2.1 Using high-level extensions of Petri nets

Noting that the subnets for each customer are similar, it is tempting to use only one subnet and represent each customer by a different token within the same net. Two changes are needed to make this possible:

1. Parallel processing of tokens by transitions: Defining the SPN behavior for multiple tokens in the same place. Some SPNs use “single-server” enabling (when tokens are enabled and fired one a time), but here the multiple enabling (or “infinite-server” (Balbo 2007)) is preferable, so that each token’s eligibility for moving to a new place is assessed in parallel (*e.g.,* two customers might simultaneously want to drive a car). While it is possible to incorporate both single and multiple servers within the same framework, single servers can be easily represented using multiple servers.
2. Colors: As demonstrated by Markov chains, the state-space explosion is primarily caused by the need to account for differences in component behavior. If a token representing a component is traveling within a distinct subnet (*e.g.,* Fig. 2), we can incorporate the differences by appropriately adjusting the properties of individual transitions for each subnet. Introducing colors to Petri nets (Jensen 1993) allows the transition properties to be color-dependent, and so components can maintain their differences while the corresponding tokens travel through the same subnet. When tokens are distinguishable, another interpretation of transition firing becomes important: rather than considering the removal of the tokens from the input places and depositing tokens to output places separately, we can consider a single action of moving tokens from the input to the output places (effectively, the entities become more persistent).

The resulting net is shown in Fig. 2, C). It looks more compact and scalable; however, implementing such a model requires a fairly complex definition of what “color” means. The original concept of a token’s color (Jensen 1993) allowed for complex attributes to be assigned to tokens, and enabled the transformation of those attributes by means of complex “inscriptions” (often elaborate formulae specified for transitions). All this resulted in powerful modeling at the (substantial) expense of reduced model readability and visual transparency. This can be contrasted with the simplest and most intuitively appealing implementation of a "color" as an integer assigned to a token that can be visualized by the corresponding color (as shown in Fig. 2, C), tokens' integers can be depicted explicitly as well, which is convenient for black-and-white implementations). The merging and splitting of tokens at transitions causes some bookkeeping difficulties in terms of tracking individual tokens' identities (colors). For example, when the car is used, there is a token that represents the first customer using the car, and the following transition should "know" the past of that merged token in order to restore the original token representing the customer and to route it to the top place. For multiple cars and family drivers, the permutations will multiply—requiring a matrix of attributes, which would explicitly stipulate complex rules governing the merging of colors and then splitting them back.

2.2 Interaction modeling in SPN

Such difficulties in tracking individual tokens through transitions can be avoided by separating transition functions. Specifically, most of the time a transition has a single input and a single output. In those cases, the need for a separate node that constitutes junctures for routing tokens among places is eliminated, and places can be directly connected by arcs. When merging or splitting of tokens is required, special joint transitions with clear token transformation rules can be used. In addition, inhibitors and their opposites, enablers, provide an important mechanism for modeling interactions among different entities in the system.

Inhibitors provide a "zero-test" capability, and are known to increase the modeling power of Petri nets equivalent to that of a Touring machine (Balbo 2007, Haas 2002). Enablers are defined in the opposite way, and are effectively test arcs (Christensen and Hansen 1993). Test arcs are used in system biology modeling (Matsuno et al. 2003), where they are denoted with directed dashed arcs; the used notation used here is chosen to emphasize the fact that enablers are the opposite of inhibitors. Historically, inhibitors were viewed with a certain degree of skepticism by the Petri net community, as they make traditional analysis of structural properties (such as reachability analysis) more complex. However, the latest view of this drawback of inhibitors is not as straightforward, since the new algorithms can successfully handle inhibitors (Ciardo 2004). In addition, there is a sufficient number of applications (e.g., modeling failure and maintenance processes of complex systems) where the state space of the problem is relatively well understood, and the main utility of the modeling consists of quantitative performance evaluation of the system. It can be shown that any system that can be modeled using Petri nets with multiple inputs and outputs to a transition can also be modeled using a combination of enablers and inhibitors with direct transitions between places (in other words, the modeling power is not reduced).

The use of joints for splitting and merging tokens is restricted only to the situations where it is advantageous and with clearly defined control of token identities. This enables direct connectivity of places, and when combined with hierarchical representation, the result is a compact yet powerful modeling framework. The result is referred to as Abridged Petri Nets (APNs) (Volovoi 2013), and resembles a hybrid between traditional Markov chain-state charts and Petri nets: transitions connect places directly (similar to Markov chains), but tokens are present to represent individual components of the system (similar to Petri nets). The tokens have discrete labels (colors) as well as continuous labels (age) (Volovoi 2004).

Transitions as the hubs or junctures of tokens' movements (depicted as rectangles) provide an ingenious mechanism for modeling components' interactions in classical SPNs. The original idea of Petri nets was conceived in the context of chemical reactions (Petri and Reisig 2008), where the merging (joining) and splitting (forking) of entities is quite common, which might explain the fundamental role of transitions that provide a direct means of modeling these processes. Merging and splitting is also important in the context of informational flows. In other applications such events occur as well, but equally if not more important are the changes that occur to entities individually. In fact, there are clear competitive advantages to the modular structuring of complex systems (Simon 2002), which might explain its prominence in both natural and engineering systems leading to so-called near-decomposibility (Courtois 1977). With this consideration in mind, entities that comprise complex systems can be considered as operating mainly independently of each other, with interactions occurring relatively rarely. However, when these interactions do occur, they are critical to the system's behavior.

The APN modeling framework reflects this near-decomposability, and considers independent (parallel) behavior as the default, while explicitly focusing attention on the interactions. The joining and forking of entities represents important mechanisms for modeling a system's interactions, but, as described in the examples below, quite often the actual mechanisms are related to enabling or inhibiting state transitions, while joining and forking provide a means of modeling such mechanisms. The difference is subtle and often not noticeable when tokens are indistinguishable (as in classical SPNs), but the consequences become significant when the entities represented by tokens have distinct identities (labels, such as colors (Jensen 1993)).

When a doctor is available to see a patient, the two entities are not actually joined into a single patient-doctor entity, which is then split back into two separate entities when the visit is over. This is simply a convenient SPN representation, but it leads to complications when we want to distinguish individual patients and doctors. The label of the token representing patient-doctor would need to contain information permitting the proper way of splitting the token into two, restoring the identity and attributes of each member of the pair and routing them into appropriate places. This can lead to labeling and routing rules that are quite complex, often hindering visual understanding of the process. APN is specifically designed to avoid these complications, and the APN properties are discussed next.

2.3 Properties of APNs

1. An APN is defined as a network of places (denoted as hollow large circles) that are connected by directed arcs (transitions). Changes in the system's state are modeled by transition firing: *i.e.,* the moving go a token from the transition's input place to its output place. The combined position of APN tokens at any given moment represents the net marking, and fully specifies the modeled system.
2. Each transition has no more than a single input and single output place (a transition can also have no input place, providing a source of new tokens every time it fires, or it can have no output place, providing a sink for tokens; upon the firing of such a transition, a token is removed from the net).
3. A token has a discrete label (color) that can change when the token is fired in accordance with the policy specified by the firing transition. Tokens also have continuous labels (ages) that can change both when tokens move, and while a token stays in the same place with the progression of time (the latter property is specified by the aging transition for the place, which is not necessarily the same as the firing transition (Volovoi 2004)).
4. A transition is enabled or disabled based on the combined marking of the input places of the associated triggers (inhibitors and enablers). Inhibitors are depicted as arcs originating at a place and terminating at a transition with a hollow circle. An inhibitor of multiplicity k disables a transition at which it terminates if the number of tokens in its input place is at least k. An enabler (depicted as an arc originating at a place and terminating at a transition with a filled circle) is opposite to an inhibitor: a transition is disabled unless an enabler of multiplicity k has at least k tokens in its input place.
5. Transitions have color- and age-dependent policies that specify the delay between the moment when the token is enabled and when it is fired (for example, one can specify separate distributions for distinct colors, while age can accumulate as the cumulative distribution function of the aging transition). If a token-transition pair is enabled, a firing delay is specified based on the combination of token and transition properties. If the token stays enabled throughout the delay, the token is fired after this delay expires. If there are multiple enabled tokens in the same place, they all can participate in the firing "race" in parallel. Similarly, the same token can be involved in a race with several transitions. If a token-transition pair is disabled, the firing is preempted (however, the aging label of the token can change as a result of being enabled for a finite amount of time).

6. The delays can be deterministic (including zero delay) or follow any specified random distributions. The firing after a specified delay is "atomic": it is a single action of moving a token from an input place to an output place (tokens don't dwell between places as they do in some versions of SPNs (Bowden 2000)).
7. Joints (token junctures depicted as triangles) can connect three places, and are a specific subset of immediate transitions in regular SPNs. While joints do not increase modeling power, there are situations where the use of the merging and splitting of tokens allows for more compact models. In contrast to SPNs, these entities are relatively rare, and analogous to the batching and duplicating building blocks in process-interaction frameworks for discrete-event simulations (Law and Kelton 2000). A split joint has a single input place. A token from this place is duplicated, and two identical copies (i.e., having the same color, age, and token ID) are immediately deposited into the joint's two output places. The merge joint provides the opposite functionality by merging two tokens from its two input places and depositing a single token into the output place. A dominant input can be specified (and denoted with a thicker arc line), which stipulates which of the two tokens provides all the attributes to the merged token. One of the three merging policies can be selected (the choice is indicated by a capital letter inside the joint) stipulating which tokens can be merged: A = any two tokens; C = only two tokens of the same color; I = only tokens with the same ID.
8. The performance of the system is based on the statistical properties of the net marking. "Sensors" or "listeners" assigned to a place provide a means for evaluating relevant statistics about the number of tokens at that place (*e.g.*, the chances of crossing a threshold for a specified period of time, the number of times a threshold is crossed, the mean and the variance of the number of tokens, or the correlation with similar quantities in another place). The presence of sensor is denoted with a solid rectangle next to a place, and multiple sensors can be assigned to the same place.
9. Fusing places, commonly used in hierarchical Petri nets (see, for example, (Jensen 1993)) is employed to connect different parts of the model. Fused places appear as distinct graphical entities during model construction, but represent the same entity in simulation.

In the next section, APNs are used as a representative SPN framework for stochastic simulation. An APN tool used to generate model snapshots and performance evaluation is available from the author upon request.

3 EXAMPLES

3.1 A Family Car

Let us return to the car example. The corresponding APN diagram is depicted in Fig. 3, A). Each family member is denoted by a token, and can be in three possible states (denoted with places): the car is not needed ("Not needed"), waiting for the car ("Waiting"), and driving the car ("Driving"). In this model there is no token representing a car; instead, an inhibitor of multiplicity one is used to prevent more than one token appearing in the "Driving" place. This inhibitor stipulates that only one car can be driven at a time. There are three transitions in the model, numbered as shown in Fig. 3, A).

Next, we introduce the possibility of the car breaking from time to time. If a trip is interrupted by the car's breaking, the family member has to wait until the car is fixed, and then attempt the interrupted trip again (from the beginning).

To model this situation, we introduce a car token. Instead of creating two separate places for both possible states of the car (broken or not), we employ token colors (*i.e.,* integer labels) to create a more compact model, as depicted in Fig. 3, B). Here we take advantage of the fact that a family member (customer) might not be able to drive a car (the car is unavailable) for two distinct reasons—either when another family member is already driving the car, or when the car is broken. We combine both possibilities into a single place, "Unavailable," and introduce an immediate transition, 4, that "pushes" the token representing a family member to the "Waiting" place when the car breaks (due to the enabler of multiplicity 2). To ensure that the tokens representing cars and people don't get mixed up, we differentiate them by color: when the car token gets to the "Unavailable" place, it has color 1, as opposed to the people tokens that have color 0; and outgoing transitions from that place are color-dependent (transitions 2 and 3 are only sensitive to color 0 [people], while transition 6 is sensitive only to color 1 [car]).

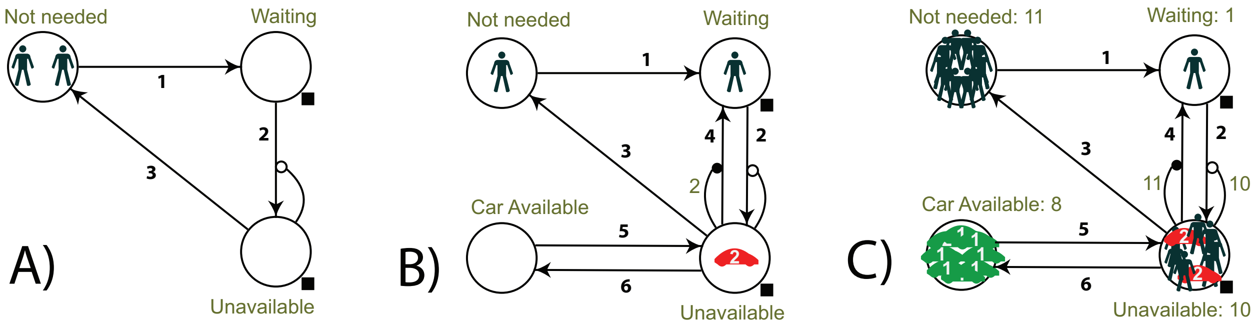

Figure 3: A) Two customers and one car, B) with car breaking; C) 20 customers and 10 cars

The model is set up for easy scalability: we simply need to add more tokens and adjust the trigger multiplicity to change the number of cars and customers. *E.g.,* for a fleet of $k = 10$ cars with $n = 20$ customers, the model is shown in Fig. 3, C) (note that the enabler multiplicity is $k+1 = 11$, and the inhibitor multiplicity is $k = 10$). At the time of the snapshot, we have two cars broken, eight cars driven, 11 customers not needing a car, and one customer waiting for a car. In this model, if a car breaks but there is another car available, the customer simply switches to another available car.

3.2 Canceling the Order

A common scenario where the job is cancelled if the total activity time exceeds a certain threshold is considered, following Jansen-Vullers et al. (2006). This publication contains a variety of workflow patterns and their feasible translation to Arena. The publication is nine years old as of this writing, so more compact constructs might have been introduced to Arena; however, given the fact that Arena is one of the leading DES tools, it is useful to compare those patterns to the equivalent implementation in SPNs . The details of the Arena model depicted in Fig. 4, A) can be obtained from Jansen-Vullers et al. (2006). For comparison, an APN model is shown in Fig. 4, B). One can observe that the APN model more directly describes the process without the need to create an artificial duplicate, as the "race" conditions are naturally implemented in SPNs. There is actually a slight difference between the two models, as the Process 1 block in Arena is represented by two different places (Queue and Service), and so in the APN model the cancellation is specific to the waiting part of the process. It can be argued that this is a more common scenario, which would require an additional check for the original of the entity to be removed in Arena (one can create an APN model that would represent the canceling when the service is in progress as well).

3.3 Doctor's Office

Next, a visit to clinic is modeled. First we follow (Robinson 2013) and consider the problem of determining whether the number of rooms is sufficient to accommodate the number of patients. Similar to (Robinson 2013), we start with the conservative assumption that all patients arrive at 9:00 AM, and evaluate the chances that all patients will be served by noon. The model is shown in Fig. 5, left; here the first two transitions are immediate, where the arrival of the patients is controlled by transition 1, and transition 2 corresponds to the transition from waiting into the room. Finally, transition 3 has a delay associated with the length of stay in a room for an individual patient. A simulation for four hours (240 minutes) is considered, with a mean value of 15 minutes and 40 rooms. The total number of patients is 200. One million simulations are used to compare exponential distribution with the lognormal distribution having

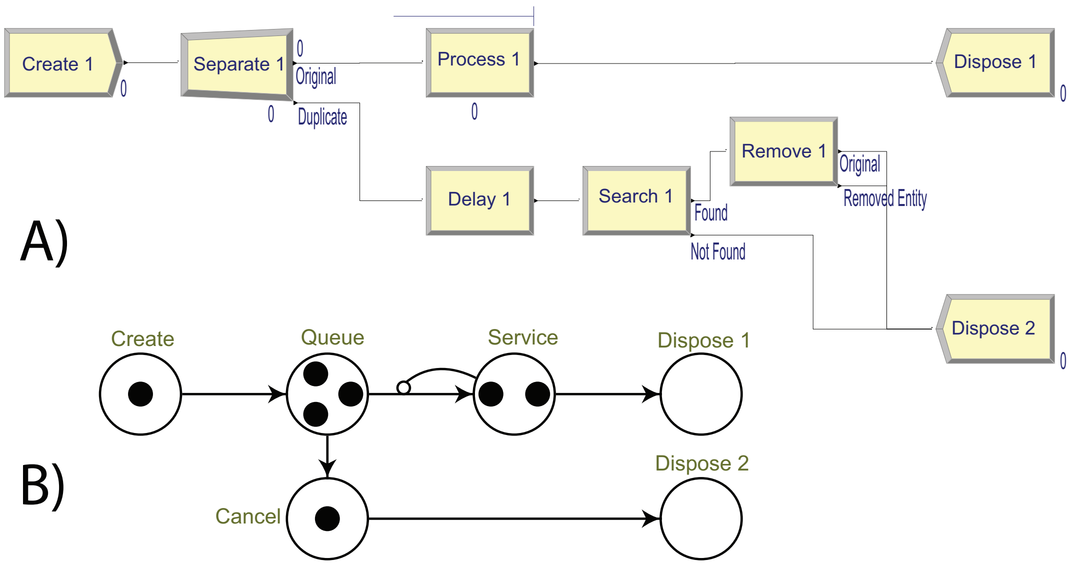

Figure 4: Arena (top) and APN (bottom) model of canceling order

the same first two moments (mean and variance). The comparison of the obtained PDFs is shown in figure

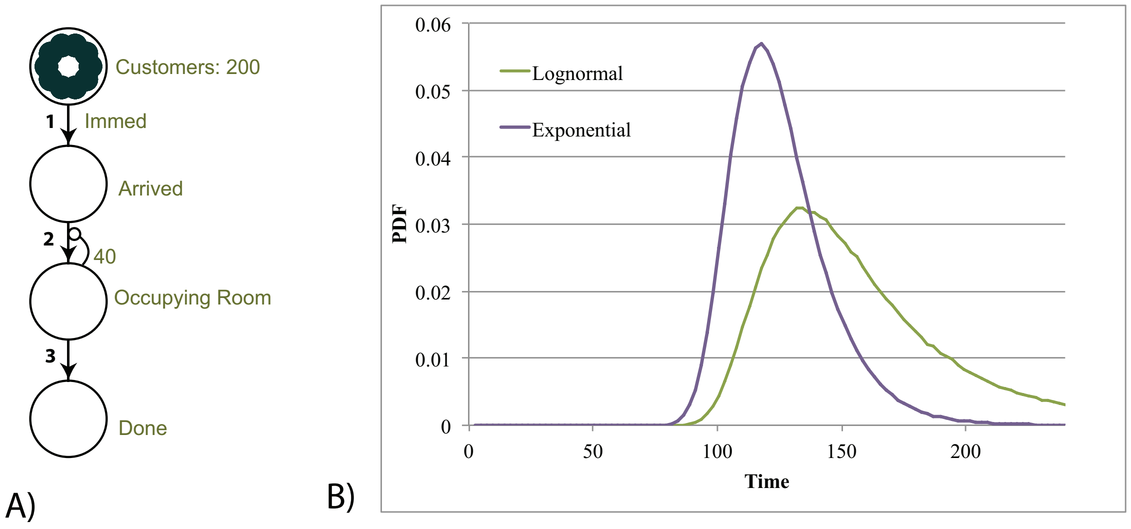

Figure 5: A) APN diagram of a simple office visit model and B) PDF of all patients served

Fig. 5, B). The “fat tail” effect is observed. In fact, if we use exponential distribution, the chances of completing the task are 98.6%; in contrast if a lognormal distribution (with matching first two moments) is used, the chances are only 77.1%. Next, let us consider a more detailed model of a doctor’s office that includes constraints on both both doctors’ and nurses’ resources in addition to constraints on the number of rooms.

The following process is modeled: the arriving patients are waiting for an available nurse and a room to get into. After they are attended by a nurse, they remain in the room and wait for a doctor; after the doctor’s visit, they are attended by a nurse again, who follows up on the doctor’s instructions before the patient is released. As a result, a nurse is needed for both pre-doctor and post-doctor tasks, and the same pool of nurses is used for both tasks (it is assumed that the nurses are interchangeable in the sense that for a given patient it is not necessary to see the same nurse at both the beginning and the end of the visit).

The corresponding APN model is shown in Fig. 6. Patients represented as tokens are initially located in the “Patients” place. In this example there are $N_1 = 100$ patients and $N_2 = 20$ rooms (the “Empty Rooms” place has 20 tokens). There are eight nurses ($N_3 = 8$) and four doctors ($N_3 = 4$). The constraint on nurse resources is enforced by an inhibitor of multiplicity 8 originating at the “Nurse” place. Similarly, the inhibitor of multiplicity 4 restricts the incoming flow in the “Doctor” place.

The patients arrive at random intervals in accordance with a known pattern based on historic demand and appointment scheduling, and proceed to the reception room. This is modeled by moving tokens into the “Waiting” place. If there is a nurse and a private room available, the nurse attends to the patient and the room becomes occupied. This is modeled by moving the patients token first to the “Admitted” place (when inhibitors allow this move), and then using the split joint to duplicate the token and deposit one into the “Nurse” place, while the copy is deposited into the occupied room. After the nurse completes the check-in procedure, the patient remains in the room waiting for the doctor (the corresponding token moves to the “Waiting Inside” place). When a doctor is available, the patients token is moved to the “Doctor” place. When the doctor is done with the patient, the patient’s token changes its color from 0 to 1 and is moved back to the “Waiting Inside” place. When a nurse is available the patient’s token is moved back to the “Nurse” place; when the Nurse completes the check-out procedure, the patients token is moved to the “Out” place. The “Out” place is an input to the merging joint that combines the token representing the patient with its duplicate located in the occupied place, and deposits the result into the “Done” place.

While not important in this model, the merging joint has an I-policy, which ensures that the occupied room and the patient tokens match (as they were originally generated by the same split action). This property to match the token by ID is important in other applications. For example, in the case of modeling a business workflow for processing a loan application, it can be useful to split a token representing an application when parts of the application are processed in parallel and then merge it based on the original token ID. Here, it is important to ensure that the two merging parts belong to the same application. An alternative means for tracking the occupied rooms can use a combination of triggers that ensures coordination of the movement of two separate sets of tokens (the movement of a single token in one group results in the movement of a single token in another). This alternative model is not shown here for brevity. Fig. 6 provides a snapshot of the model when 18 rooms are occupied: six patients are attended by a nurse, four by a doctor, and the seven patients are waiting inside. The tokens representing patients that have been already served by a doctor have color 1, which is shown in yellow in Fig. 6.

The parameters of the model can be divided into two categories: timing of individual events (such as patient arrival or time spent with a doctor) and the number of involved entities, such as demand (patients) on the one hand, and resources (rooms, nurses, and doctors) on the other. As shown below, the number of entities can be represented either as the number of the corresponding tokens, or as trigger multiplicity, depending on what is preferable. Next, both set of parameters are discussed:

Transition Parameters Transition T_1 from the “Arrival” place to the “Waiting” place determines the patient arrival pattern. The time each patient spends with a nurse during check-in is specified by the delay assigned to the transition T_2 from the “Nurse” place to the “Waiting Inside” place. Similarly, the time during check-out is described by the transition T_3 from the “Nurse” place to the “Out” place, and the time with a doctor is specified by the transition T_4 from the “Doctor” place to the “Waiting Inside” place.

Entities This example demonstrates two distinct means for describing the effect of multiple entities. Tokens are used to represent both patients and rooms. Rooms, nurses, and doctors all can be considered as resources, so similar constructions to the “Occupied Rooms” place could be used for nurses and doctors. However, in this example the use of triggers (inhibitors) is used instead to describe appropriate constraints. In general use of the triggers is more compact but less flexible, as it it is suitable only for describing a single resource. Changing the number of resources entails adjusting either the number of tokens or the multiplicity of the corresponding triggers without the need for altering the structure of the model. In addition, one can use a similar construction to that in the car example (see Fig. 3) to change the number of resources with time. Specifically, one can add another place from which tokens (of a different color) are deposited to and removed from at a specified schedule (or randomly) to model the corresponding reduction in the number of the respective resources.

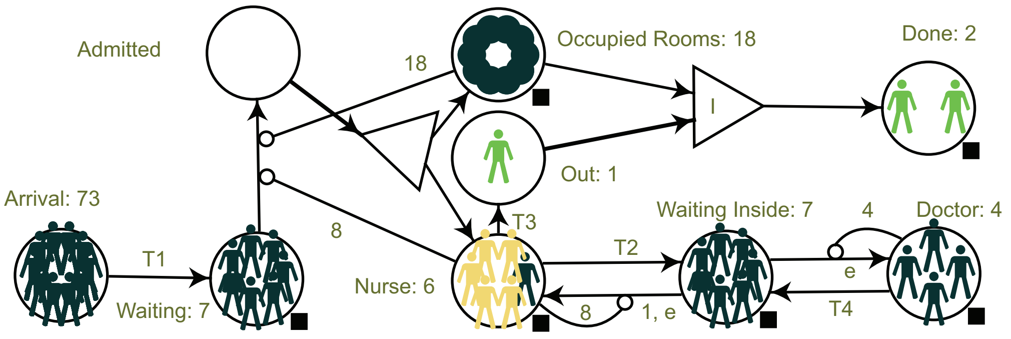

Figure 6: APN model of medical office with constrained resources: a snapshot of simulation in progress

The relevant performance measures of the system can be evaluated, such as expected (mean) waiting time for the patients, the probability of all patients being served, the number of served patients, etc. For example, let us consider the situation when patients' arrivals are uniformly distributed in the interval from 0 - 3 hours, both check-in and check-out by the nurses take a fixed 10 minutes, and the duration of a doctor's interaction with a patient varies in accordance with an exponential distribution with a mean of 10 minutes. The results of one million Monte Carlo simulations are considered for two scenarios: for Scenario A there are four doctors, eight nurses, and 20 rooms; and for scenario B an additional nurse is added, but the number of rooms is reduced to 18. On average, a patient requires exactly twice as much time with a nurse as with the doctor, so increasing the nurse-to-doctor ratio above two is not obviously beneficial, as the number of doctors remains a bottleneck of the process. Utilization as a function of time is shown in Fig. 7. One can observe that, while the average number of working nurses does not exceed eight even for scenario B, the utilization of all resources is slightly higher, which translates into a shorter wait for the patients. The total average waiting time is reduced from 43 minutes to 39 minutes. The chances of serving all patients within five hours is about 44.3% for case A and increases to 57.5% for case B.

3.4 Race Track

When discussing emergent behavior in the context of agent-based systems, the congestion pattern (a queue) that propagates in the opposite direction of the movement of individual cars is often mentioned (Bonabeau 2002, Brailsford 2014). In Brailsford (2014), the modeling of a car race around a circular track where no overtaking is allowed is discussed as an example that is difficult to model in traditional DES, but is easy to model in ABM. If all cars are driving at the same speed the initial spacing between the cars is preserved, but introducing speed variability will break the original pattern. The initial problem has a continuous spatial representation, which can be preserved in the ABM, but would require discretization in APN. Fig. 8 depicts an APN model with 12 cars and 23 places representing the track. Each place has an inhibitor for incoming transitions that prevents overtaking. The multiplicity of these inhibitors is one, but other multiplicities can be used to model multi-lane tracks with the multiplicity reflecting the number of lanes at a given location.

Depending on the perspective, the need to discretize the spatial coordinates of cars can be considered as either an extra hindrance or a gentle nudge toward a higher level of abstraction with the benefit of clarifying the essential parameters of the model. In particular, the following question can be raised about the specification of speed variability: is it persisting over time? In the simplest scenario the answer is yes—so some cars are always faster than the others, and patterns of train-like car platoons will form, with

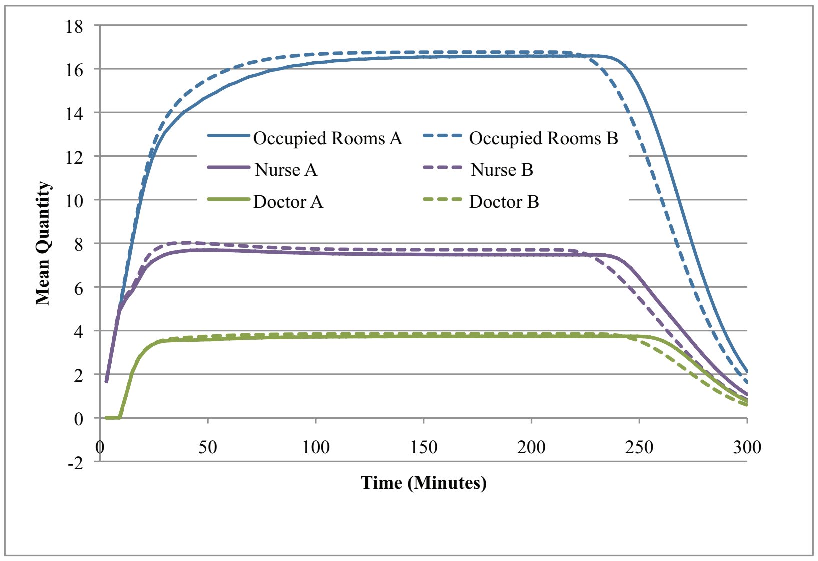

Figure 7: Utilization of resources as a function of time

the lead cars being the slowest and aggregating all the cars behind it that are faster. In APN, the easiest way to model this situation would be to assign different colors to different cars and set up a distinct traveling time for each color (as the green cars in Fig. 8). The corresponding continuous-time ABM model would be quite simple. However, one can also consider a perhaps more interesting situation, where the cars vary their speed with time in a random fashion. Interesting patterns can emerge even if all cars are similar, and the modeling of this situation in ABM requires answering the following question: how should the time variability be modeled? Even if the modeler is comfortable with auto-correlation and similar time-series concepts, it is reasonable to assume that some type of discretization would be an easier solution. This discretization has been implemented in APN already, so varying the speeds in a random fashion as cars move from a segment to segment does not require any additional modeling efforts.

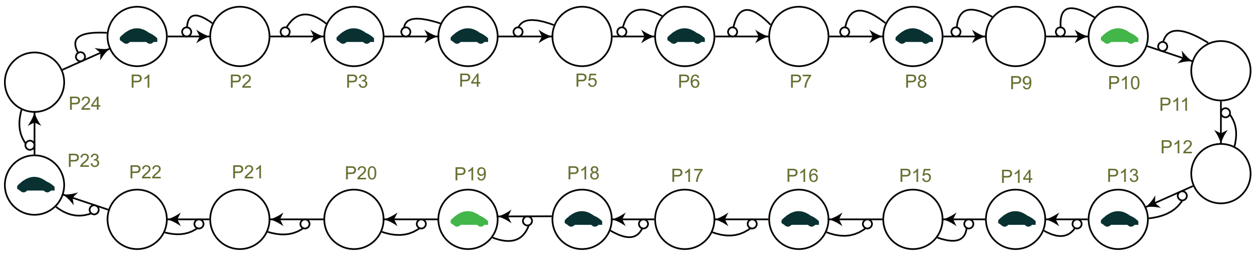

Figure 8: Race track model in APN

4 CONCLUSIONS

This tutorial is focused on Stochastic Petri Nets (SPNs) in the context of stochastic simulation. SPN modeling represents a component-based state-space world view that is distinct from both process-interaction and event-scheduling views. While still using the latter for the back-end engine of the simulation, this state-space view allows direct races among several processes impacting the same entity simultaneously, and provides a graphical means for modeling direct interactions between distinct entities that comprise the system. Traditional process-interaction implementations, such as Arena, also rely on state-space representation of some system entities (such as resources); however, the underlying dynamics lack a standard graphical representation of those state changes, and especially of interactions with other entities. State-space representation can also provide advantages over process-interaction representation in compact model initialization. A component-based view of SPNs is contrasted to global (system-level) state-space representation used for describing Markov chains, with the former providing clear advantages in reducing model size and complexity. Multiple versions of SPNs are discussed in the tutorial, including distinguishing individual tokens and assigning attributes to tokens (colors), introducing triggers (enablers and inhibitors), and streamlined SPNs that assign transition properties to connecting arcs directly (Abridged Petri Nets).

SPNs can serve as a useful compromise between traditional DES and more flexible (yet lacking standard building blocks) agent-based simulation. The graphical nature of the models and the small number of standard building blocks facilitates easy and fast model creation. The models are self-documented and therefore uniquely suited for tool-independent auditing and conveying the insights from simulation to the decision makers. These features of SPNs can be useful for lowering the effective level-of-effort barrier for simulations, independently verifying traditional DES models, and building business cases. On the other hand, the visual nature of the model logic implementation in SPNs can be less compact than its programmatic equivalent, so the advantage over traditional DES frameworks can be lost in certain cases. In particular, SPNs might be inappropriate when the processes are "transformational" rather than "reactive" and when a process-interaction world view provides efficient modeling representation. SPNs can also provide a middle-layer, "under-the-hood" level of abstraction, with a higher level of abstraction provided by domain-specific formalism, such as reliability block diagrams in system reliability, as implemented in GRIF (Signoret et al. 2013). Similar hybrid simulations might be useful for process-based DES.

REFERENCES

Aalst, W. M. P. V. D. 1998. "The Application of Petri Nets to Workflow Management". *Journal of Circuits, Systems and Computers* 8 (1): 21–66.

Balbo, G. 2007. "Introduction to Generalized Stochastic Petri Nets". In *Formal Methods for Performance Evaluation*, edited by M. Bernardo and J. Hillston, Volume 4486 of *Lecture Notes in Computer Science*, 83–131. Springer-Verlag.

Bonabeau, E. 2002, May. "Agent-Based Modeling: Methods and Techniques for Simulating Human Systems". *Proceedings of the National Academy of Sciences* 99 (3): 7280–7287.

Bowden, F. 2000. "A Brief Survey and Synthesis of the Roles of Time in Petri Nets". *Mathematical and Computer Modelling* 21:55–68.

Brailsford, S. C. 2014. "Modeling Human Behavior—an (Id)Entity Crisis?" In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 1539–1548.

Christensen, S., and N. D. Hansen. 1993. "Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs". In *Application and Theory of Petri Nets*, edited by M. Ajmone Marsan, Lecture Notes in Computer Science, 186–205. Springer Berlin Heidelberg.

Ciardo, G. 2004. "Reachability Set Generation for Petri Nets: Can Brute Force Be Smart?" In *Application and Theory of Petri Nets*, Volume 3099 of *Lecture Notes in Computer Science*, 17–34. Springer Berlin Heidelberg.

Courtois, P. J. 1977. *Decomposability: queueing and computer system applications*. New York, NY: Academic Press.

Haas, P. J. 2002. *Stochastic Petri Nets. Modelling, Stability, Simulation*. New York: Springer.

Harel, D. 1987. "Statecharts: a Visual Formalism for Complex Systems". *Science of Computer Programming* 8:231–274.

Hayes, B. 2013. "First Links in the Markov Chain". *American Scientist* 101:92–97.

Henderson, S., and B. Nelson. (Eds.) 2006. *Handbook in OR & MS: Simulation*, Volume 13. Elsevier B.V.

Jansen-Vullers, M. H., R. IJpelaar, and M. Loosschilder. 2006. "Workflow patterns modelled in Arena". Technical report, Technische Universiteit Eindhoven.

Jensen, K. 1993. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Volume 1. Berlin: Springer.

Kesaraju, V., and F. Ciarallo. 2012. "Integrated simulation combining process-driven and event-driven models". *Journal of Simulation* 6:9—20.

Law, A., and W. Kelton. 2000. *Simulation Modeling and Analysis*. 3rd ed. New York, NY: McGraw-Hill.

Marsan, M. A. 1990. *Stochastic Petri nets: An elementary introduction*, Volume 424 of *Lecture Notes in Computer Science*, 1–29. Springer.

Marsan, M. A., G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. 1995. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons.

Matsuno, H., Y. Tanaka, H. Aoshima, A. Doi, M. Matsui, and S. Miyano. 2003. "Biopathways Representation and Simulation on Hybrid Functional Petri Net". *Silico Biology* 3 (3): 389–404.

Miller, J., G. Baramidze, A. Sheth, and P. Fishwick. 2004. "Investigating ontologies for simulation modeling". In *37th Annual Simulation Symposium*, 55—63. IEEE.

Nelson, B. 2002. *Stochastic Modeling: Analysis & Simulation*. Mineola, NY: Dover Publications.

Petri, A. 1962. *Kommunikation mit Automaten*. Ph. D. thesis, Institut für Instrumentelle Mathematik, Schriften des IIM.

Petri, C., and W. Reisig. 2008. "Petri Net". *Scholarpedia* 3:6477.

Robinson, S. 2013. "Conceptual Modeling for Simulation". In *Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 377–388.

Roeder, T. M. K. 2004. *An Information Taxonomy for Discrete Event Simulations*. Ph. D. thesis, University of California, Berkeley.

Ross, S. M. 2002. *Simulation*. 3rd ed. Academic Press.

Schruben, L. 1983. "Simulation Modeling with Event Graphs". *Communications of the ACM* 26:957–963.

Siebers, P. O., C. Macal, J. Garnett, D. Buxton, and M. Pidd. 2010. "Discrete-event simulation is dead, long live agent-based simulation!" *Journal of Simulation* (4): 204–2010.

Signoret, J.-P., Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, and P. Thomas. 2013. "Make your Petri nets understandable: Reliability block diagrams driven Petri nets". *Reliability Engineering and System Safety* 113:61–75.

Simon, H. A. 2002. "Near decomposability and the speed of evolution". *Industrial and Corporate Change* 11 (3): 587—599.

Trivedi, S. K. 2002. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Second ed. John Wiley and Sons.

Volovoi, V. 2013. "Abridged Petri Nets". *ArXiv*:arXiv:1312.2865.

Volovoi, V. V. 2004. "Modeling of System Reliability Using Petri Nets with Aging Tokens". *Reliability Engineering and System Safety* 84 (2): 149–161.

AUTHOR BIOGRAPHIES

VITALI VOLOVOI is an independent consultant in the area of industrial internet, risk, reliability, and the dynamic interactions of complex systems. He collaborates with Mitek Analytics, consults for Logistics Management Institute, and serves as a member of the NASA Statistical Engineering team. He has led many projects, including turbine analysis for the Air Force, quantitative risk assessment of the Space Shuttle wiring for NASA, air transportation safety analysis for NASA, condition-based maintenance of gas turbines for Siemens, and reliability of the Hubble Space Telescope gyros for NASA. He has received two NASA Engineering and Safety Group Achievement Awards, a NASA Engineering and Safety Technical Excellence Award in 2014, and the best tutorial award at the Reliability and Maintenance Symposium (RAMS) in 2011. He has a Ph.D. in Aerospace Engineering from Georgia Tech, and a University Diploma in Mechanics and Mathematics from Moscow State University. His e-mail address is vitali@volovoi.com.