Abridged Petri Nets

Vitali Volovoi

***Abstract*—A new graphical framework, Abridged Petri Nets (APNs) is introduced for bottom-up modeling of complex stochastic systems. Drawing on selected properties of both Color and Stochastic Petri Nets (CPNs and SPNs, respectively), APNs aim at maximizing the use of dynamic components of Petri nets (tokens) in representing systems' entities. Instead of creating dedicated subnets for distinct entities, the corresponding tokens follow similar routes while maintaining their identifies. This is achieved by purely graphical means (*i.e.*, without resorting to coding) using a judicious choice of standard routing building blocks, distinguishing between one-way and two-way interactions, and employing hierarchical model building. As demonstrated by several examples from diverse domains, APNs enable clean, visual, and compact diagrams while retaining the modeling power and flexibility of SPNs. As a result, APNs facilitate the efficient performance evaluation of complex systems and effective communication with decision makers. To support quantitative performance evaluation of complex systems, APN models are analyzed using discrete-event simulations. In this context, APNs are compared to traditional process-interaction discrete-event simulations.**

***Index Terms*—Petri nets, discrete event simulation, state-space stochastic models, business workflow performance.**

I. Introduction

THIS paper introduces Abridged Petri Nets (APNs), a new graphical framework for modeling stochastic behavior of complex systems that consist of multiple interacting components. Modeling of such systems is relevant to several domains. While the underlying stochastic processes share common fundamental principles, the preferred tools are often domain-specific and their choice can be driven by legacy (inertia) considerations, as much as by the current demands of a given domain. As modeling sophistication grows and modeling scope increases (both breadth- or depth-wise), simplifications that made particular domain-specific tools initially attractive become less relevant.

These tools evolve in response to changing demand by incorporating new features that do not always provide a good fit with the original principles for those tools. Two trends are usually observed as a result: On the one hand, tools become increasingly bulkier as the new (not envisioned originally) features are "retrofitted". On the other hand, the functionality of the tools effectively converges, as those tools are increasingly expected to tackle the general dynamics of complex systems. These trends are reinforced by the ongoing consolidation of commercial vendors, as relatively small companies that used to cater to niche domains are merged with larger companies that aim at providing "cradle-to-grave" suites of modeling tools for engineering systems.

V. Volovoi is an Independent Consultant, e-mail: vitali@volovoi.com.
Version February 6, 2020.

For example, in engineering reliability and safety, Boolean-logic based frameworks (reliability block diagrams, and fault trees, respectively) provided convenient tools for estimating system-level performance in a computationally efficient manner. As technologies evolved and the impact of systems repairs and reconfigurability increased, the tools relying on those frameworks were supplemented with Monte-Carlo simulations and Markov chain modeling "under-the-hood," while keeping the user interface relatively unchanged.

A similar process can be observed in the field of discrete-event simulations (DES), where the original need for modeling relatively orderly manufacturing processes and influence of queuing network theory have led to the dominance of process-interaction DES. The popularity of these tools (see an informative overview [1]) expanded the realm of applications to services and logistics among others, leading to introduction of state-based features, especially in the context of modeling resources.

Petri nets have experienced a similar evolution in terms of incorporating the concepts of components' heterogeneity and time that were not included in the original formulation [2]. The original formulation has focused on two distinct nodes types, places (denoted with hollow circles) and transitions (denoted with rectangles) that formed a bi-partite directed graph (no two nodes of the same type can be directly connected). Tokens (denoted with small filled circles) provide a means to "mark" a place, that is to identify a state a place was in. From the state-space perspective, each place represented a component of the modeled system that can assume states enumerated by non-negative integers, and the tokens provided a visual mechanism for depicting those integers. In fact, marking often is directly denoted with a non-negative number inside the place. As a result, the tokens have been viewed originally as indistinguishable and not possessing any attributes on their own. In this setting, modeling of components with distinct properties, even if they shared some attributes, required a dedicated subnet for each component (*i.e.,* the commonality could not be exploited).

Similarly, the time delays associated with the changes in components states were deliberately excluded from the original Petri nets [2], [3]. The changes in components' state (events) were implemented by so-called "firing" of a transition that immediately removes tokens from all its input places (as identified by the origins of the directed arcs terminated at the transition), and deposits tokens into the output places (identified by the outgoing directed arcs from the transition). The lack of an explicit notion of time provides a useful level of abstraction for establishing important structural properties of the modeled system, such as reachability, boundedness, liveness (absence of dead-locks), etc. [4]. As a result, one can formally verify that a certain combination of undesired

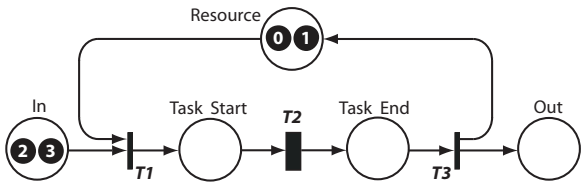

Fig. 1. Conceptual SPN resource model

components states is never reached (or ensure that desired states are always reached), which is useful, for example, in formal verification of software [5].

Historically, the development of Colored Petri nets (CPNs) and Stochastic Petri nets (SPNs) followed distinct paths. In [6] it is argued that CPNs [7] were not specifically designed to incorporate stochastic behavior. As a result, Generalized Stochastic Petri Nets (GSPNs) seem to be more suitable for modeling performance of business workflows, even though CPNs are more commonly used for this purpose. Following this logic one step further, we argue that GSPNs were originally developed for indistinguishable tokens, and as a result were not specifically designed for distinguishable [8] (colored) tokens. In particular, when dealing with the resources, existing colored GSPNs, such as those used in [6] still require dedicated subnet for each resource, or coding of complex rules ensuring the proper routings of tokens.

The basic issue is illustrated in Figure 1 where several resources (servers) are utilized to accomplish similar tasks for multiple customers. We focus on the situation when resources are distinct (for example, the costs of their utilization are different). Following standard SPN notations, we denote immediate transitions using thin bars, and transitions with the finite delays using filled rectangles. Transition $T1$ in Figure 1 provides a match between the supply and demand: for example a resource token 0 is matched with the customer 2, and the firing of the transition $T1$ results in removing those two tokens from their respective places, with a token being deposited into the "Task start" place.

This deposited token enables the $T2$ transition, initiating the task, specifying the attributes of this token *i.e.,* its color, is not trivial. This color has to inherit properties of both removed tokens, as these properties are needed subsequently when the token is split back by transition $T3$. This transition needs to deposit the resource and customer tokens into correct places. The resulting multi-dimensional colors can be specifically designed for this problem [9], but the implementation of those colors in a general setting is bound to require elaborate programmatic (coding) means. The associated knowledge barrier can be perceived as advantageous by a modeler, or convince the modeler to hide the Petri net under an additional layer of a user interface. Alternatively, one can resign to providing dedicated subnets for each resource unit, which comes at a price if there is a large number of resources.

In this context, APNs goal is to balance the needs of color and time, and provide a means to model resource allocation is similar complex synchronization patterns in a compact and graphical fashion without resorting to coding. In particular, it will be shown how the tracking of individual multiple resources can be represented within the same network. The following aspects of APN make this possible and discussed below upon formal introduction of APN:

- Tokens are viewed as persistent entities that survive individual firings and can represent system components. A frugal set of token's attributes is selected to balance modeling flexibility and the ease of specification.
- The number of inputs and outputs for transitions is deliberately restricted to make the routing of persistent tokens tractable. Specifically, transitions with one input and one output are denoted as directed arcs between places, and joints (depicted as triangles) represent the basic building blocks for splitting and merging tokens (cf. [9]).
- By default, tokens behave in parallel fashion, and interactions are explicitly specified using two main mechanisms: symmetric interactions represented by joints, and one-directional interactions represented by triggers that either enable or disable transitions. The distinction facilitates more transparent representation of causal mechanisms present in the modeled system.
- Hierarchical models are utilized that rely on fused places (places that appear as distinct graphically but represent the same place), multiple pages (with subnets connected by means of fused places), and layers (stacked pages of similar subnets with an automatic color shift).

The overarching idea is that traditional Petri net transitions with arbitrary number of inputs and outputs provide too much modeling flexibility when tokens are distinguishable and need to be routed into specific outputs. On the other hand, the majority of transitions in practical models have one input and one output, so representing such transitions with a single arc instead of two arcs and a rectangle reduces visual clutter.

One could even consider SPNs with only one input and one output transitions (with all interactions modeled using triggers). In fact, as shown below, no modeling power is lost. However, such restriction on transitions is excessive in practice, and joints facilitate more compact practical graphical implementations for certain scenarios. In particular, it allows the tracking of individual assets and cases *without providing dedicated subnet* for each tracked entity.

The paper is organized as follows: Section II provided the background on relevant modeling frameworks, including Markov chains, regular and colored SPNs, all using the same simple scenario of matching car (resource) with customers. Section III introduces the building blocks of APN. demonstrates that restricting the model to single input single output does not reduce modeling power for Petri nets with indistinguishable tokens. The choice between place vs. token for representing system component is discussed, in particular in the context of resource modeling (and analogy to the process-based model is made). Next, joints are introduced and their role in modeling matching diverse resources and customers are discussed (including a solution to the problem depicted in Figure 1). In Section IV several use cases are considered demonstrating the versatility of the proposed framework. Finally, the conclusions are presented in Section V, where the

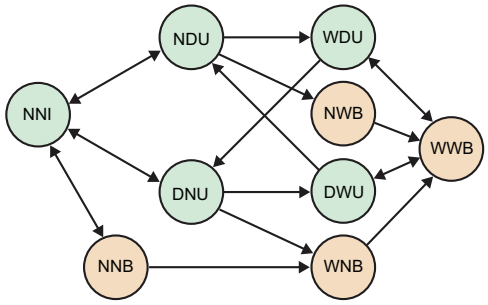

Fig. 2. Markov chain diagram for two customers and one car. Additional places colored in soft yellow introduce the possibility of car breaks

difference with traditional process-based DES are emphasized.

II. Background

A. Markov chains

Markov chains were introduced more than a century ago by Andrey Markov as a means of generating sequences of dependent random events for which a version of the law of large numbers (i.e., convergence to a steady-state) can be applied. The first-order temporal structure of this dependence (each event depends only on the immediately preceding event), usually expressed as "the future depends only on the present and not the past," is not as restrictive as it might seem. Indeed, the definition of the present can include the salient information from the past, thus expanding the state-space, while keeping the relationship linear. Such state expansion is analogous to the introduction of canonical coordinates in Hamiltonian mechanics, where the second-order differential equations of Newtonian dynamics are replaced with the first-order equations, so that the future trajectory in the phase space is uniquely defined by the present canonical coordinates.

The resulting balance of simplicity and flexibility in capturing temporal dependence has ensured broad success of Markov chains as a core of several major informational breakthroughs of the last century, including Shannon's information theory and the web page ranking algorithms developed by the founders of Google [10]. Even the original application of Markov chains by A. Markov to modeling the sequences of vowels and consonants in Alexander Pushkin's novel *Eugene Onegin*[11] remains remarkably relevant in the context of modern applications such as speech recognition and machine translation. Originally defined for discrete time, the Markov chains were extended to continuous time by Andrey Kolmogorov [12]. State transition diagrams help to visualize both discrete and continuous time Markov chains. Traditionally, the discrete version explicitly includes the probabilities of staying at the same state (depicted with self-loops), while for continuous time self-loops are usually omitted.

Despite this success, Markov chains are prone to "state-space explosion"—they scale poorly as the number of components that comprise the system increases. To demonstrate this, let us consider a simple continuous-time example corresponding to a general problem of balancing supply and demand. The problems of matching supply and demand are at the core of modeling needs with SPNs [13]. Although the SPN applications are not limited to these types of problems, they provide sufficient wealth of component interactions to illustrate modeling challenges and test various means for overcoming those challenges. As discussed in [14], specific meaning for the customers that generate the demand and the suppliers of the service that satisfy the demand greatly varies depending on the application domain.

For specificity, the example is cast in terms of a household consisting of n family members (later referred to as customers) and k cars. Three states are defined for each customer: not needing a car (N), driving a car (D), or waiting for a car (W); for simplicity, being a passenger in the car driven by another customer counts as state N. The following inputs to this model could be considered: the usage pattern of each customer, *e.g.,* trips duration and frequency, and the car properties, *e.g.,* the frequency of breaks and the duration of the repairs. The outputs from the model would estimate the "performance" of this "system," such as the frequency and duration of unsatisfied demand. As a result, the impact of changes to the system can be assessed, *e.g.,* increasing or decreasing the number of cars.

The smallest non-trivial scenario corresponds to a single car ($k = 1$) and two customers ($n = 2$). 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 define the state of the entire system. However, not all of those permutations constitute feasible system states, so the corresponding Markov model has only five (instead of 18) states. The green-colored states in Figure 2 represent the corresponding Markov chain diagram. Here the state of each entity is represented by the corresponding capital letter; for example, the DNU system state denotes the following combination of "component" states: the first customer is driving, the second is not needing the car, and the car is in use.

Extending the possible car's states to include "broken" (state B) increases the size of the diagram to nine states (the corresponding additional states are colored in soft yellow in Figure 2). For a scenario with n customers and k cars, the number of system states can be calculated as follows: at any given time we can have $0 \le m \le n$ members of the family who need cars (demand) and $0 \le l \le k$ cars that are not broken (supply). For each $m \le l$, the number of distinct states corresponds to the number of ways we can select m driving members of the family $N_1(m,l) = \binom{n}{m}\binom{l}{m}\binom{k}{l}$. If $m > l$, then $m - l$ members of the family are waiting for a car (*i.e.,* in state W), so the corresponding number of states will be $N_2(m,l) = \binom{n}{m}\binom{m}{l}\binom{k}{l}$. The total number of states will be given by the following formula:

$$\sum_{m=0}^{n}\binom{k}{l}\left[\sum_{m=0}^{l}\binom{n}{m}\binom{l}{m}+\sum_{m=l+1}^{n}\binom{n}{m}\binom{m}{l}\right] \quad (1)$$

For a family of four with two cars, the number of possible states is 115, and for a family of five with three cars, that number is 634. If we consider a fleet of 10 cars with 20 customers, the number of states is $451,805,366,885$—or over 451 billion.

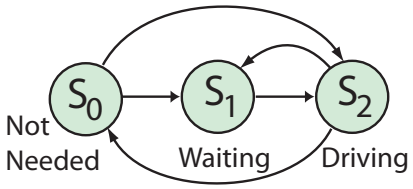

Fig. 3. Markov chain diagram for a component (customer) view

This rapid growth of the state-space size can be mitigated 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 without and with the possibility of breaks, respectively, as states such as DNU and NDU can be merged together. However, often one does want to track individual performance: the customer X might take longer trips, and car Y is old and breaks more often. On the other hand, the state space can further increase if individual pairings of cars and customers need to be to differentiated (say, customer XZ is a teenager who drives a sports car XY more carelessly than a minivan YY, and has a higher chance of breaking XY than YY, while his mother drives both cars equally carefully). Such differentiation would require $q!$ possible combinations for q cars driven at any given time.

In addition to the state-space size issue, the "global" system-level view at multi-component systems poses challenges for modeling varying with time state-transition rates. As discussed in [15], semi-Markovian processes that allow for taking into account the time spent in a given state (sojourn time) are not always sufficient due to the fact that the relevant elapsed time is often tied to a specific component, and this resolution is lost at the system level. A component-based representation of system states has a potential of address both challenges.

The following state-space diagram can be constructed (see Figure 3) for each customer. The result is a system of component models that are coupled and need to be solved simultaneously. The interrelationships about the dynamics of individual components can be quite involved when transition rates are not constant even when there are only two components. Stochastic Petri nets are specifically designed for the purpose of describing the coupled behavior of multiple components in a single model and are discussed next.

B. Stochastic Petri nets

Each state in Figure 2 is labeled using "alphabetical" principles, similar to the alphabet-based scripts where words are comprised of standard "components" (letters), thus obviating the need of distinct pictograms for each word. Petri nets effectively provide a graphical equivalent of an alphabet-based representation by modeling the states of individual components, rather than the explicit states of the entire system.

As discussed in the introduction, in Petri nets, Markov chain-state diagrams are complemented by two new types of objects: tokens and transitions. Both types of objects are briefly discussed next:

- *Tokens:* small filled circles denoting individual components are 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).
- *Transitions:* in order to model interactions among components explicitly, the tokens are routed among places via intermediate stops or junctures, called transitions, which are denoted with solid rectangles. Two places cannot be connected by an arc directly; instead they must be connected through a transition. The number of input and output arcs to a transitions does not need to coincide, enabling the merging and splitting of token routes (and therefore, effectively, the splitting and merging of tokens themselves).

The timing of state changes is represented by time delays for "firing" of transitions: an action that removes tokens from all input places for the transition and deposits tokens into its output places. Such Petri nets with time delays are called timed Petri nets, or, more specifically, Stochastic Petri Nets (SPNs) [16], [17], when delays can be nondeterministic and follow a specified distribution. 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. Here, no limitations on the associated types of distributions are considered to facilitate the broadest possible range of applications. Once the problem is posed, the relevant metrics of the modeled system can be obtained either by means of discrete event simulation or using alternative techniques for non-Markovian processes (see, for example [15]).

Figure 4 depicts an SPN for two customers and one car. The same notations are used as in Figure 1. SPN in Figure 4 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.

In other words, the pattern of matching resources (see Figure 1) is present in this problem, but for now we have only one resource, and we have dedicated networks for each customer. The resulting 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 (ten), connecting arcs (26), and tokens (3). For comparison, the corresponding Markov model has 23 arcs.

Still, 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 [18]). Indeed, let us have n customers and k cars. This leads to n subnets for each customer that would have $k+2$ places (since we would have separate place for each of k car used) and two additional places for each car. The total the number of places in the model would be $n(k+2)+2k$, so for 20 customers and 10 cars there

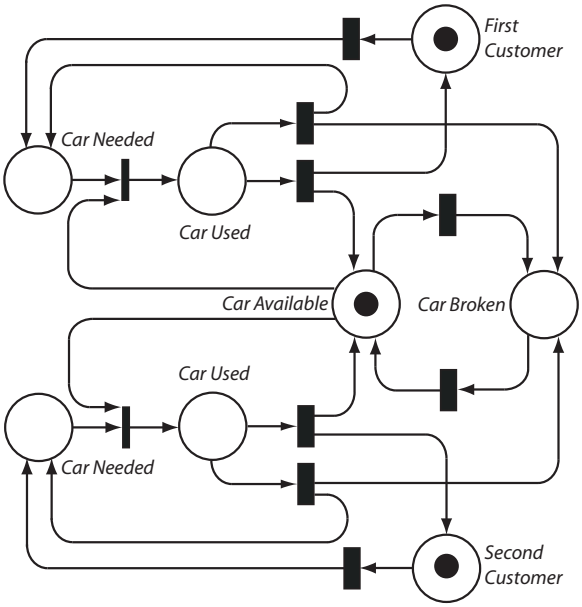

Fig. 4. Stochastic Petri Net diagram for two customers and one car, including the possibility that car can break during the trip.

would be only 260 places, which is certainly an improvement over 451 billion! However, the web of connecting arcs will be so convoluted that the resulting model is still too complex to be of practical use for conveying the system behavior visually. Referring back to Figure 4, one can envision 20 segments of the net similar to the two depicted at the top of the net, except that each of the subnets would have 10 places for car used instead of one, 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).

C. Using high-level extensions of Petri nets

Noting that the subnets for each customer are similar, it is tempting to use only one of the subnets and represent each customer by a different token within the same net. At least two modifications are needed to enable such modeling:

1) Parallel processing of tokens by transitions: we need to define the behavior of the net when there are multiple tokens in the same place. While some SPNs use "single-server" enabling (when tokens are enabled and fired one a time), here the multiple enabling (or "infinite-server," to be precise [16]) is preferable, so that each token's eligibility for moving to a new place is assessed in parallel (*e.g.,* two customers might want to drive a car simultaneously). While it is possible to incorporate both single and multiple servers within the same framework, single servers can be easily represented using multiple servers, so only multiple servers are used herein.
2) Colors: As discussed in the context of 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. 4), one can incorporate the differences in components behavior by appropriately adjusting the properties of individual transitions for each subnet. Introducing labels (colors) to Petri nets [8], [7] allows for the transition properties to be color-dependent, so that distinct components can be represented by tokens with distinct colors traveling through the same subnet.

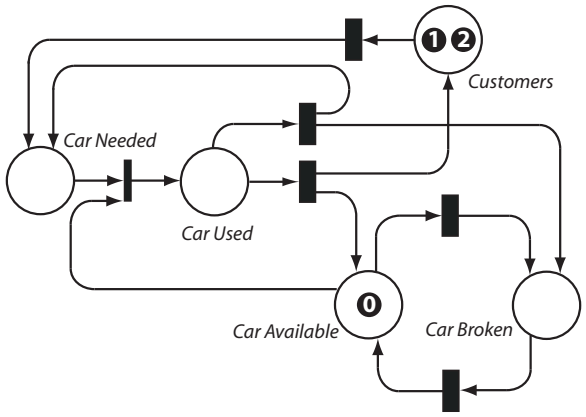

Fig. 5. Colored Stochastic Petri Net diagram for two customers and one car. Integer labels (colors) are directly shown inside each token.

The resulting network is shown in Figure 5. It looks more compact and scalable. However, just like with the resource problem discussed in the introduction, implementing such a model requires a fairly complex definition of what "color" means. The original concept of a token's color [7] allowed for complex attributes to be assigned to tokens, as the associated means of transforming those attributes by means of "inscriptions" (often elaborate formulae specified for transitions), which allowed for 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" is an integer assigned to a token that can be visualized by the corresponding color (as shown in Figure 5, tokens' integers can be depicted explicitly as well, which is convenient for black-and-white implementations). One can observe that 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 to restore the original token that represents the customer and route it to the top place. If we have 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. Next, we formally introduce APNs and demonstrate possible solutions for this challenge.

III. APN

APN inherits most of the features of the Stochastic Petri nets [16]. APN is defined as a network of places (denoted as hollow large circles) that are connected by transitions. Changes in the system's state are modeled by transition firing: *i.e.,* the removing tokens from the transition's input places to

depositing tokens to its output places. The combined position of APN tokens at any given moment represents the net marking $\mathcal{M}(t)$, and fully specifies the modeled system. Only enabled transitions can fire. A transition T_i is enabled at time t if the associated Boolean-valued "guard" condition $G_i : \mathcal{M}(t) \mapsto \{false, true\}$ is evaluated as $true$. Transitions can be timed or immediate. Timed transitions have an associated time delay that is either deterministic or is sampled from a specified distribution. Timed transitions fire after they are enabled for a specified time delay.

There are four groups of distinct APN properties that were outlined in the introduction and are described in detail next:

A. Persistent Tokens

APN treats tokens as persistent (as opposed to transient) entities that survive individual transition firings. When a transition fires tokens, two separate actions of removing tokens from the input places and depositing (potentially some other) tokens into the output places are united into a single action of moving tokens from the inputs to the outputs. The implication here is that components of the systems in APN can be represented not only by places (as in traditional SPNs) but by tokens as well.

Tokens can experience "births" and "deaths" throughout simulation as a result of firing certain transitions (as described in the transition section below), so the number of tokens is not preserved.

The following attributes (labels) are associated with a token:

- An identity number (ID), an individual non-negative integer automatically assigned to a token. The purpose of token's ID is to track individual tokens, in particular when tokens are split and merged (see the joint description in the next subsection). ID assignment for individual tokens is dictated by the joints' actions. Several tokens can have the same ID, which implies they model different aspects of the same entity, and some duplicates of an original token have been created (using split joints) at some earlier time. In addition to its current ID, a token also maintains a record of a "shadow" or "recessive" ID - the ID of a token that was most recently merged with the current one (if the ID of the merged token was different). The shadow ID provides a memory of the past token's identity, which can be useful later, similar to the concept of recessive genes.
- A color, a non-negative integer that indicates that a given token token belongs to a certain class/group of tokens. In particular, transition policies can be color dependent, and the color attribute determine the selection of appropriate transition policy. This color can change when the token is fired in accordance with the policy specified by the firing transition.
- An age: a continuous label $a \in [0, 1[$ that can change both when the token is fired, and while it stays in the same place with the progression of time. The former change is discrete, and specified by the parameter $\eta \in [0, 1[$ of the appropriate policy of the firing transition: $a \mapsto \eta a$. The latter property is specified by the aging transition for the place where the token resides, which is not necessarily the same as the firing transition. For stochastic delays, the age corresponds to the value of cumulative distribution (CDF) for the elapsed time [19], while for a fixed delay the age is simply the fraction of the elapsed time as compared to the fixed delay. The latter option provides a simple means to explicitly enforce queueing priorities, such as First-in-First-out (FIFO).

Token color is the only attribute that impacts the marking of the net, and therefore the guards, while token's age can impact time delay associated with transitions [19]. As a result, when a system component is represented by a token, its state can be characterized both by the token's color and its place. Using more traditional for Petri nets Eulerian perspective, a marking of a place is characterized by a multi-set: a union of sets of tokens of different colors occupying a given place.

The second consequence of using an atomic firing is that a transition with a single input and a single output has a very clear and simple interpretation: a token move represents the state change of the component depicted by this token; in this context, the visual object of a transition node becomes superfluous, and two places can be directly connected by an arc, as in Markov models. The practical implication of eliminating these superfluous visual objects is usually quite substantial due to the fact that majority (and often all) transitions have one input and one output. Next, APN transitions are described in more detail.

B. Transitions

APN employs only two types of transitions: direct arcs connecting two places that can have a finite delay, and joints that facilitate splitting and merging tokens (no delays are associated with those actions). Both types are described next.

Regular Transitions: A regular transition is enabled or disabled based on the combined marking of the input places of the associated triggers, inhibitors and enablers that are described in the next subsection.

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).

The delays can be deterministic 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 do not dwell between places in contrast to some versions of SPNs [20]).

Joints: Joints allow merging and splitting of tokens and depicted as equilateral triangles. A joint connects together three places, and it provides an alternative to a regular transition that connects two places. Joints actions are immediate. Split joint has one input place and two output places. When present, a single token is removed from the input place, and two identical copies of that token (that have the same IDs, colors and ages) are deposited into the output places.

Merging joint has two input places and a single output place. For merging joint we can specify which of the two tokens will be "dominant" and passes its attributes (age, color, and ID) to the merged token. If the ID of the "recessive" token is distinct from that of the dominant token, the dominant token "inherits" the ID of recessive token as a "recessed" or "shadow" ID. The dominant input path is shown with a thicker arc (the recessive token input is shown with the regular arc). There are four options for the merger joint:

- Any two tokens can be joined. This option is depicted with the letter "A" inside the joint.
- Only tokens of the same color can be joined together. This option is depicted with the letter "C" inside the joint.
- Only tokens with the same ID (that is those that originated from the same split joint at some point in the past) can be joined. This option is depicted with the letter "I" inside the joint. This is the only merging where there is no "recessive" ID to inherit for the dominant token, since both tokens have the same main ID. The main purpose of the "shadow" ID is to provide a memory about the previous mergers, so in this case the dominant token inherits the recessive ID from the recessed token.
- A recessive token retrieves its "recessed" (shadow) ID and matches it with the ID of a token at the origin place of the dominant input arc. The main ID of the recessive token is inherited as the recessive (shadow) ID of the dominant token. This option is depicted with the letter "R" inside the joint, and it is useful when matching multi-class resources, as described below.

C. Modeling Interactions (Guards)

APNs consider default token's firing as parallel (independent), and therefore focus on the coupling (interactions) in clearly defined fashion. Specifically, two mechanisms are employed: triggers (inhibitors and enablers), and merging of tokens using joints. Both of those mechanisms are discussed next.

Triggers: 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 the opposite to an inhibitor: a transition is disabled unless an enabler of multiplicity k has at least k tokens in its input place. Triggers describe a unidirectional (one-way) dependence: the marking of a place where the trigger originates influences the enabling of a transition (and therefore its firing). The unidirectionality of dependence facilitates unambiguous modeling of causality of modeled processes.

Joints: Merging joints provide a bi-directional means of dependence by merging a pair of tokens that satisfy matching criteria of the joint. However, the symmetry of component's interaction is not complete since one of the inputs is dominant (the merged token inherits most of the properties of the dominant input, except the recessive ID, as described above).

D. APN Hierarchical Properties

Hierarchical constructions for combining multiple subnets are used to model large-scale systems. Fusing places, commonly used in hierarchical Petri nets (see, for example, [7]) are employed to connect different parts of the model, including different model sub groups. Each page of the model can display several subgroups (so the same subgroup could appear in more than one page if needed). Fused places appear as distinct graphical entities during model construction, but represent the same entity in simulation. When two places belonging to different tabs are fused together, it simply means that they are separated for visual convenience and readability, but they are essentially the same place from the simulation perspective. One can visualize different pages as distinct layers, like floors in a building with elevators connecting distinct floors.

Taking this logic one step further, one can effectively utilize a third dimension (depth) and consider a stack of similar (but not necessarily identical) subnets that are positioned as a deck of cards on the same page.

An automatic generation of layers of multiple pages that contain similar but possible distinct subnets can be facilitated.

In particular, an automatic "color shift" is used to differentiate between the tokens from a given layer and ensure that the tokens enter the desired layer when needed.

IV. Examples

A. Modeling with Directly Connected Places

Next, we demonstrate that any immediate transition with multiple inputs and outputs 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). Let us consider an immediate transition with n inputs and m outputs. First we note that it is sufficient to show that we can reproduce two basic patterns: merging of two streams tokens together, and splitting a single stream into two. Indeed, then we can apply sequentially $n-1$ merging segments, thus merging n inputs into a single one, and then we can apply $m-1$ splitting segments.

Figure 6 demonstrates the equivalent APN construct for merging pattern. Here fast transitions of fixed durations (as noted in the figure) are used to ensure the desired order of the transition firing, with the basic unit of time ϵ chosen small enough to avoid a noticeable impact on the quantitative results due to the overall delay incurred after all transition fired *e.g.,* 10^{-6} of the characteristic time of simulation). Let us consider a situation when at time t there is at least one token in both places "A" and "B". At time $t+\epsilon$ a token is fired from the "B" place into the "B in" place. This enables transition at the top of the net from the "A" to "A in" place, but also inhibits (disables) transition that just fired to ensure that exactly one

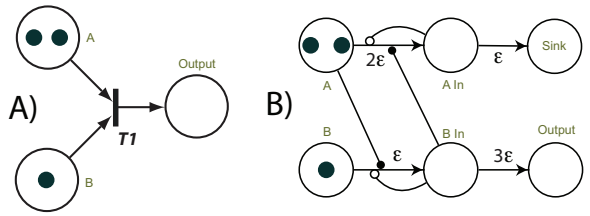

Fig. 6. Equivalency of a merging pattern in A) regular SPN and B) APN without joints

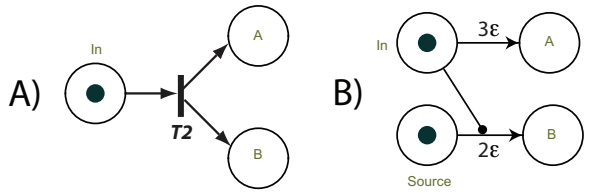

Fig. 7. Equivalency of a splitting pattern in A) regular SPN and B) APN without joints

token is removed from the "B" place in a single cycle of transactions. At time $t + 3\epsilon$ a token is fired from the "A" place into the "A in" place, also disabling the transition that just fired, thus ensuring that only one token is removed from the place. Finally at time $t+4\epsilon$ two transitions fire: the token is removed from the "A in" place into the "Sink" place (a special type of place with no outputs where tokens disappear), and the token is moved from the "B in" into the "Output" place. Then the process can be repeated if there are more pairs to be matched.

Figure 7 demonstrates the equivalent APN construct for splitting pattern, here a source type of place is utilized to create new tokens with the delay specified by the outgoing transition. Similar to the merging partner, the timing of the fast transitions is selected to ensure that each transition fires exactly once per incoming token. In contrast to the merging scenario, there is no need to specify an inhibitor for the transition that originates at the source, as source has only one token at any given time.

A natural question is related to representing timed transitions in a similar format. While for the splitting pattern, one simply need to insert a timed transition before the split, for the merging pattern (when transition $T1$ in Figure 6 has a finite delay) the situation is less straightforward. Indeed, we need to define first what the merging pattern for the infinite server firing policy for finite delay with multiple tokens. Are the pairs of tokens to be merged identified at the enabling phase and a clock is associated with each pair of tokens (one from each incoming place)? What happens if one of the tokens from this pair gets pre-empted (fired by another transition) before the clock runs down? Does the pair gets disabled, or it looks for another match at the same place? It seems that such policies become too complicated for distinguishable tokens to be represented by a simple notation as shown in Figure 6 A) when $T1$ has a finite delay.

B. Car modeling

Let us return to the car example. The corresponding APN diagram is depicted in Fig. 8A. There are two types of tokens: customers and cars (specific shapes can facilitate visualization in a software implementation, but here we use more traditional circle shapes).

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. 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 customer is already driving the car, or when the car is broken. We combine both possibilities into a single place, "Unavailable," and provide an inhibitor to transition 2 that ensures a limited capacity for that place. Furthermore, an immediate transition, 4, "pushes" the token representing a customer 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, they are differentiated by color: when the car token gets to the "Unavailable" place, it has color 0, as opposed to the customers' tokens that have color 1; outgoing transitions from that place are color-dependent (transitions 3 and 4 are only sensitive to color 1 [people], while transition 6 is sensitive only to color 0 [car]). 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).

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 $k = 2$ cars with $n = 4$ customers, the model is shown in Fig. 8B (the enabler multiplicity is $k+1 = 3$, and the inhibitor multiplicity is $k = 2$). At the time of the snapshot, we have one car broken, one car driven, two 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.

The compactness of the model is due in part to representing a resource as a place with limited capacity (by means of an inhibitor). Conveniently, this capacity does not to stay constant throughout the simulation. Instead, it effectively changes when tokens representing broken cars displace the customer tokens. Such a construction has a venerable tradition in Discrete Event Simulation when the effects of broken servers are simulated, and in general it is worthwhile to note the similarity of this representation to that of process-interaction DES [1]. However, there are important differences as well: first, the example demonstrates the ability to interrupt (cancel) a process when the car is breaking and the the broken car token pushes the customer token into the waiting place. Traditionally, process-interaction DES require more elaborate constructions to model such effects [14]. Such representation is useful when it provides the desired level of modeling resolution. As discussed in [21], focusing on the "servers" (marking of a place) rather than "transactions" (tokens)[1], can provide a coarser level of abstraction that leads to more compact models.

There are situations, however, when more explicit modeling of the resource use is required, analogous to the models

[1]Effectively assuming Eulerian rather than Lagrangian viewpoint, as elaborated in the Discussion section.

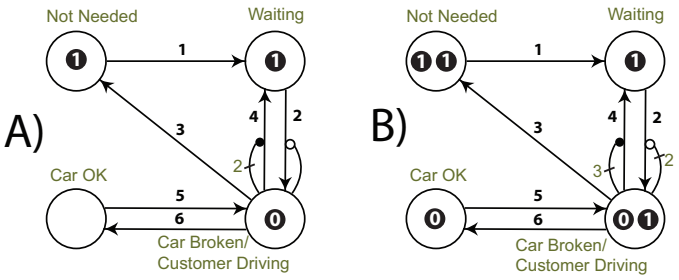

Fig. 8. APN model of matching customers with cars A) 1 car and 2 customers and B) 10 cars and 20 customers

represented in Figures 1,5, and next we discuss how these scenarios can be modeled in APNs using joints. In particular, some of the approximations can be removed (for example, we might want to consider that car breaks more often when they are actually driven, instead of a "smeared" representation of cars breaking).

Resource modeling with joints: The notional SPN model for resource allocation (Figure 1) is translated into an explicit APN model as shown in Figure 9. The explicit APN model is naturally more complex than the idealized version for SPN. This latter version can be directly reproduced in APN with transition $T1$ represented by a merging joint with policy "Any", $T2$ with a regular transition (represented as a direct arc) and $T3$ represented as a split joint. However, the resulting model would only be valid for a non-distinguishable resources, unless complex inscriptions are utilized. Otherwise, each resource requires a dedicated subnet that are fused together in the shared pool of resources place. The APN model shown in Figure 9 is described next.

When an individual customer token appears in the "In" place it is duplicated first. One of the copies is used to initiate the resource request. If the resource is available, the merging joint is fired, and the merged token is deposited into the "Resource seized" place. The resulting token inherits resource properties (this is ensured by selecting as dominant the upper input branch into the joint, as visualized by the thicker line of the dominant arc). The ID from the task token is inherited as a recessive ID for the token (it will be used later). The joint type is "Any" as here no requirement to the type of resource is specified.

This model can be further refined if only some of the resources are suitable for the task (in this case colored policy can be selected for the joint). The resulting token is duplicated next to keep a resource copy. The other copy is merged with the copy of the task token, where the recessive ID merging option "R" is utilized (since the recessive or shadow ID of that token matches the task's ID). After the task is completed and the token moves to the "Task end" place it gets duplicated with one copy used to indicate the completion of the task (the "Out" place), while the other copy is used to merge with the copy of the resource and release the resource. Here the dominant input is from the "Resource copy" place, and "R" matching policy is utilized.

Next, the explicit resource model is utilized to construct a more detailed car-customer model, as shown in Figure 10.

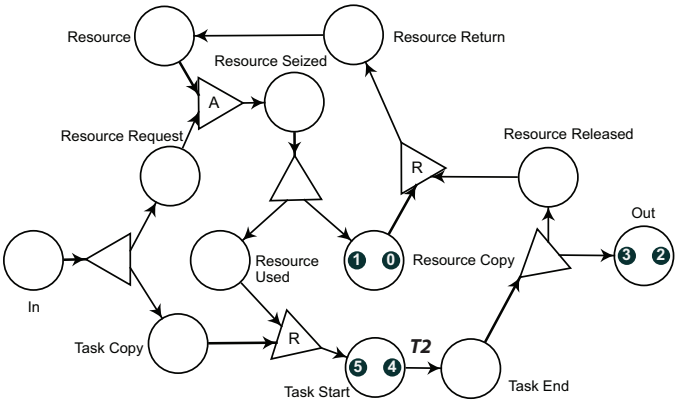

Fig. 9. APN model of explicit resource modeling

Car and customer symbols are used to represent individual tokens and the numbers inside of each token denote its ID. The model represents a refinement of the model shown in Figure 8. Several transitions are carried over: specifically, the delays associated with the demand (transition 1), the duration of each trip (transition 3), as well as the car breaking and repair (transitions 5 and 6, respectively). However the car breaking is modeled more precisely by distinguishing the delays between the breaks while the car is not used (which is represented with transition 5) and the frequency of car breaks during the trip (the corresponding distribution is assigned to transition 2, and it does not have an equivalent in Figure 8). This model reproduces most of the elements of the general resource problem (*cf.* Figure 9), with several additions: first the entire closed-loop model is shown.

To keep the general left-to-right flow of the model, we fuse three pair of places together: the "Not needed", "Broken car", and "Need car" places respectively. The key difference is the introduction of transition 2 (corresponding to the possibility of car breaking during the trip). Figure 10 shows the snapshot when such event occurred to car with ID 2. After the car token is moved to the "Car breaks" place, the car token is duplicated using a split joint with the lower branch used to deposit a car token into the "Broken cars" place.

The upper branch deposits the duplicate token into the "Car broke" place (this is the frame shown in Figure 10). The recessive ID of this car token is 6, since it was assigned when the resource (car) was originally seized[2] by a customer with that ID. As a result, the merging joint with the recessive ("R") policy is enabled, the car token is merged with the dominant token representing the customer with ID 6 and the resulting token is returned to the "Need car" place (since the customer has not completed his trip and has to repeat it). If desired, the customer token can be aged by transition 3, so that an interrupted trip (or any other service provided by the server) can be resumed not from the beginning but where it was left off (the age would be reset upon firing of the transition 3).

[2]The term "seized" is inherited from discrete event simulation where it represents the starting moment when the resource is not available for other use, so it does not imply that the car or its engine was actually seized.

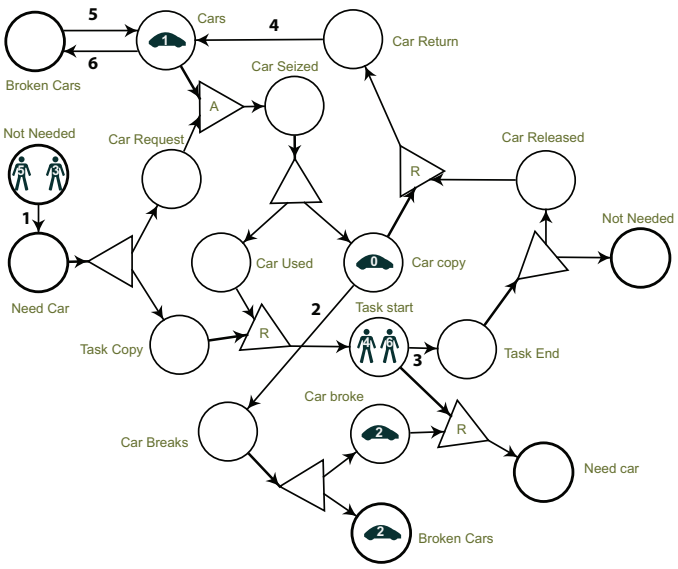

Fig. 10. Cars and Customers model in APN with explicit resource modeling

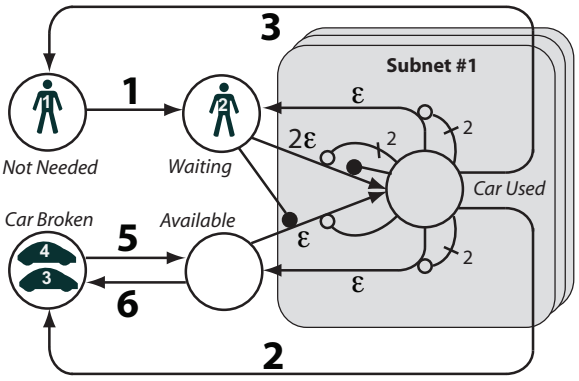

Fig. 11. An Abridged Petri Net (APN) diagram for multiple cars and customers while tracking specific pairing of customers with the service

C. Explicit matching of supply-demand pairs

To this end, it is convenient to expand the model into the third dimension (or "depth") by considering subnets stacked in layers, as depicted in Figure 11. Unless we want to assign specific properties to particular pairs of tokens (see the next section), the number of layers does not need to exceed the number of demand-supply pairs at any given time, so that for k cars and n customers, at most $\min\{k, n\}$ subnets are needed.

In a software implementation, these layers can be created automatically, and one can switch the views among layers (by bringing the layer of interest to the top of the stack) as desired. The six triggers of each subnet (see the right, shaded part of Figure 11) are arranged to match one car with one customer within each net (assuming that both the customer and car are available). Figure 11 depicts a situation where there is a customer waiting for a car, but no cars are available: Token 1 appears in 'the 'Waiting" place and enables an immediate firing of a token representing a car from the "Available" place to the "Car used" place, but there are no tokens in the "Available" place. When, for example, token 3 representing a car moves the "Available" place (the car has been repaired) it is immediately moved to the "Car Used" place, and the inhibitor from that place disables this transition, preventing more tokens (representing cars) from being fired (if there were more tokens in the "Available" place, they would not move). It must be noted that the reverse transition is also enabled momentarily, so either that transition needs to be assigned a lower priority (e.g., slower, if we are using fixed small delays), or we can make the triggers color-sensitive (so that the enabler in the transition from the "Waiting" place to the "Car Used" place is sensitive only to car tokens, while the inhibitor is sensitive only to the customer tokens—the multiplicity of the inhibitor should be equal to unity in this case).

This enables the transition for the customer to move from the "Waiting" place to "Car Used" gets enabled and fires (token 2), and similarly the inhibitor ensures that only one token moves (since the total number of tokens in the "Car Used" place cannot exceed two—that is, the multiplicity of the corresponding inhibitor). As a result, if the customer represented by Token 1 is also in the "Waiting" place, it would continue to wait for a car.

When the second car is repaired, transitions for Subnet $\#2$ are enabled, and the matching of the customer with a car takes place. At this point we have both customers driving cars. If a car breaks in the first subnet, the corresponding Token 3 moves to the "Car broken" place, then the transition from the"Car Used" place to the "Waiting" place becomes enabled and fires the token 2. When the trip ends for the second pair (the customer's Token 1 moves to the "Not needed" place, then the car's Token 4 moves back to the "Available" place , enabling the possibility for the first customer to complete the trip.

In this example, all subnets were essentially equal—they simply provided a means of accounting for each pair of tokens separately. However, one can further refine the model and allow for the possibility of differentiating among the subnets. In the considered example, we can assign a particular combination of colors to a specific subnet that might have distinct properties (for example, if a teenager drives a car, the chances of the car's breaking might increase). Next, we describe a general mechanism that facilitates the modeling of this and similar situations.

Let us consider a situation with multiple but distinct subnets, so that tokens that leave a subnet should be returned to that specific subnet (and not to any other subnet). A repair process is one example of this situation, and so is the processing of documents: when a car is sent to the shop, one hopes to receive the same car fixed (and not somebody else's fixed car). Using an automated color shift allows this situation to be modeled automatically. First, we determine the range of colors utilized in the subnet that is used as a template for multiple subnets (see Figure 12). Let us denote this range as $1 \ldots j$ without the loss of generality (recall that colors are useful only to differentiate transition policies, so they can always be shifted together).

If we want to create m subnets, then we introduce a

shifted range of colors: for k-th subnet ($1 \leq k \leq m$) the corresponding range of colors will be $jk+1 \ldots j(k+1)$. An automatic check of color changes can be implemented to verify that colors are not changed outside of the subnet, so that a token cannot be accepted by the "wrong" subnet. Figure 12 depicts subnet $\#1$ (color range $0 \ldots 2$) when the component denoted with the token 2 needs service. Token 3 returns from service Subnet $\#2$ (color range $3 \ldots 5$) when the component denoted with.

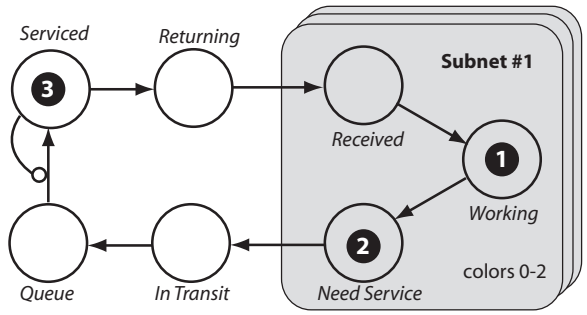

Fig. 12. A general representation of the color shift model using Abridged Petri Nets (APNs). There are multiple transitions from the outside of the subnets to the subnets, but each transition is restricted in terms of the range of colors it can transmit to ensure that tokens return to their "home" subnet

D. Customer Interruption

Here we consider the challenge of balancing the economies of scale and the risks posed by the interruptions following [22]. Let us consider a motor vehicle department, with first-in first-out (FIFO) and the possibility that some customers will abandon the queue. The corresponding model is shown in Figure 13. Transition $T1$ controls the flow of incoming customers: when it fires the customers stop arriving due to the inhibitor that disables transition $T2$. Similarly the top right of the model provides a clock for interruption and restoration of service (transitions $T5$ and $T6$, respectively). To model FIFO policy, transition $T3$ that represents the abandonment of the queue is specified as the aging transitions for tokens located in the "Waiting" place. This is combined with specifying transition $T7$ as fast fixed age-dependent transition: when the scale of the transition is ϵ, and the age is $0 \leq a < 1$, the delay is $a\epsilon$. This ensures that the "older" tokens are fired first by the $T7$ transition.

V. Discussion

In this section some of implications of the choices made in APN are discussed, following the same four general categories that was used to define APN.

A. Persistent Tokens

Representing system components in APN using not only places but also tokens is analogous to invoking two classical views in continuous mechanics. Therein, a continuous flow, for example of water in a river can be studied from the perspective of a fixed location that encounters varying quantities of the

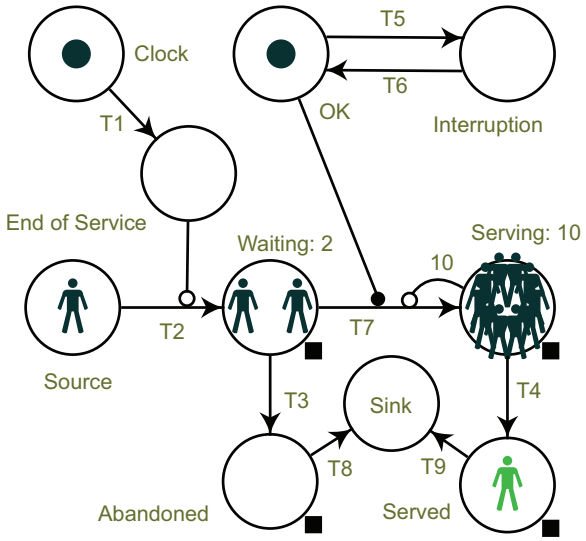

Fig. 13. A simple APN model of a queue with customer abandonment and interruption

fluid that passes by. This is referred to as Eulerian viewpoint. In contrast, the Lagrangian approach considers the flow from the point of view of a moving particle (see for example [23]). Using this analogy APN provides the Lagrangian in addition to Eulerian viewpoint that is more commonly used by Petri net formalisms[3]. As discussed in [15], both perspective can be useful in understanding complex stochastic processes.

The first benefit of merging the two separate actions into one is the removal of the ambiguity about the timing of those two actions: are removing and depositing tokens occur simultaneously, which is referred to as atomic, or can tokens dwell in the transition node, which corresponds to the so-called three-phase firing [16]? Different versions of Petri nets treat this issue differently (see the discussion on the subject in [20]).

It can be argued that the utility of separation between places and transitions can stem from assigning distinct meanings to the two types of nodes. For example, in the context of workflow a transition is associated with an activity [25], while a place with a condition of an entity. However, from the general state-space perspective, places represents the states of entities, and "activities" are only relevant as long as they result in the changes of those states. In this context, the presence of possibly multiple activities for a given entity is unambiguously represented by transitions enabled for a given token representing an entity of interest. If the activity is completed, the corresponding transition fires the affected token into a new place (as specified by the transition output).

B. Transitions

While joints do not increase modeling power when tokens are not distinguishable, there are situations where the use of the merging and splitting of tokens allows for more compact models. Joints are analogous to the batching and duplicating

[3]both Lagrangian and Eulerian viewpoints were originally introduced by Leonhard Euler, see for example [24]

building blocks in process-interaction frameworks for discrete-event simulations [26]. The relative strict restrictions on the type of transitions (as compared to transitional SPNs) is by design [4] to ensure that distinct identities of persistent tokens are tracked in a transparent and repeatable fashion.

C. Modeling Interactions

Inhibitors provide a "zero-test" capability, and are known to increase the modeling power of Petri nets equivalent to that of a Touring machine, but they don't change the modeling power of Stochastic Petri Nets [16], [17]. Enablers are defined in the opposite way, and are effectively test arcs [27]. Test arcs are used in system biology modeling [28], where they are denoted with directed dashed arcs; the 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 [29]. 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.

Importantly, inhibitors and enablers provide direct means for modeling unidirectional causal mechanisms of the modeled processes. In contrast, joints (restricted equivalent of merging and splitting common in SPNs) represent more symmetric patterns of causality. The resulting diversity of means for modeling causality enables more direct and faithful representation of the causal mechanisms of the modeled systems.

There are multiple mechanisms that either implicitly or explicitly imply interactions in traditional SPNs. Three of those mechanisms are *not* used in APN:

1) Competition for firing from the same place when "single-server" (or limited capacity) policies are used. Single-server policy if the default policy in many versions of SPNs and it provides a compact representation of simple queueing with random order. However, when the tokens are distinguishable, this policy is of limited use, and can be easily represented using the infinite server policy combined with an extra place that only allows one token at a time (by means of an inhibitor). The use of infinite server (that effectively assumes token independence) is critical for effective model of complex systems. As a result, to avoid confusion (and lacking standard distinguishing notations with the infinite server) APN does not use a single-server (or limited capacity).
2) Arc multiplicities. In traditional Petri Nets places represent components, with the state space of the component represented by the number of tokens in that place (place marking). There are situations when the state change is a "jump" so that state changes by more than one position. For example, if the old state was represented by five tokens, and the new state by three tokens, this change is represented by the outgoing transition of multiplicity two that fires two tokens simultaneously. In contrast, in APNs the main mechanism for representing system components is tokens, and they interact with transitions individually: if a place has several tokens in a place with an outgoing transition, each token has its clock with respect to the transition. The state of the token can be represented by its integer attribute (color), and upon firing through a transition we can specify the increment to that color (so if the color was five, this increment can be set to -2, resulting in the token of color three). Alternatively, if one desires to coordinate the firing of tokens - and fire two tokens at the same time, this can be modeled by providing an explicit construction with an inhibitor that allows only two tokens in a place at a time.
3) Marking dependence is effectively a "catch-all" means to represent system interactions that are not modeled otherwise. There is no standard graphical notations, so the dependence cannot be captured visually. This is fundamentally a programmatic way of capturing the dependence, and it is avoided in APN.

[4]In other words, this is a feature not a bug.

D. Hierarchical Constructions

Modularization is a fundamental mechanism for dealing with complexity. Separate model construction and performance evaluation of individual modules provides the most reliable implementation path, but it relies on the ability to assembly the outputs from individual modules into a higher-level model. This effectively replies a tree-like structure of the model and the ability to characterize the outputs from the tree "leaves" in a compact fashion. The former is not always appropriate, and the latter can be challenging as well: see, for example [30].

Multi-page models in APN provide a viable alternative to fully modularized models, and the use of stacks of layers and automatic color shifting provides additional convenience when appropriate.

VI. Conclusion

This framework can be considered as a derivative of Stochastic Petri Nets (SPNs) [13] that aims at retaining SPNs versatility in terms of modeling power, while streamlining the choice of the modeling building blocks. The visual clutter and often confusing choices that are often perceived as the major obstacle to the larger success of SPNs are reduced [20], resulting in simpler and more transparent models that can be built using only graphical interface.

The focus of the paper is on the modeling "front-end", that is the graphical interface with the user. Modern computing capabilities make Monte-Carlo simulation techniques powerful enough to render user interface to be the critical bottleneck in constructing large-scale yet error-free stochastic models. At the same time, the inherent hierarchy of the models constructed using APN has a direct impact on the underlying

event-scheduling architecture, enhancing the scalability of the models from the computational perspective as compared to process-interaction discrete event simulations. The models developed in APN where the logic is implemented visually look fundamentally different from the logic deployed in other DES, as usually, once the logic exceeds the basic constructions of the framework, the programmatic means are employed.

In some situations programmatic logic representation is more efficient, but even in those scenarios APN can provide a value as an alternative verification that the logic of the model is indeed coded up correctly (in the spirit of N-version programming [31]). For example, in the context of safety and reliability, APN can provide a useful middle layer of abstraction that is directly focused on the timing of events. This would fill an important gap between high-level static models (e.g., fault trees and reliability block diagrams) on on the one hand, and detailed spatial simulation on the other. The importance of nested complementary models of diverse levels of abstraction is invaluable to ensure that the engineering systems (e.g., Autonomous Vehicles) can be trusted to behave as intended [32].

Finally, the completeness of the visual representation serves an important self-documentation purpose: the picture of a model combined with a table of transition properties uniquely defines the model, fully independently of a specific computer implementation.

REFERENCES

[1] C. M. Jenkins and S. V. Rice, "Resource modeling in discrete-event simulation environments: A fifty-year perspective," in *Proceedings of the 2009 Winter Simulation Conference*, M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, Eds., 2009, pp. 755–766. 1, 8

[2] A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Institut für Instrumentelle Mathematik, Schriften des IIM, 1962. 1

[3] C. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, p. 6477, 2008. 1

[4] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989. 1

[5] D. A. Peled, *Software Reliability Methods*. Springer, 2001. 2

[6] C. Oliveira, R. Lima, H. Reijers, and J. Ribeiro, "Quantitative analysis of resource-constrained business processes," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 42, no. 3, pp. 669–684, May 2012. 2

[7] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Berlin: Springer, 1993, vol. 1. 2, 5, 7

[8] W. Reisig, "Petri nets with individual tokens," *Theoretical Computer Science*, vol. 41, no. 0, pp. 185 – 213, 1985. 2, 5

[9] A. K. Schömig and H. Rau, "A Petri net approach for the performance analysis of business processes," University of Würzburg Institute of Computer Science, Research Rep ort Series 116, 1995. 2

[10] P. von Hilgers and A. N. Langville, "The five greatest applications of Markov chains," in *Proceedings of the Markov Anniversary Meeting*. Boson Books, 2006. 3

[11] B. Hayes, "First links in the Markov chain," *American Scientist*, vol. 101, pp. 92–97, 2013. 3

[12] A. Kolmogoroff, "Zur Theorie der Markoffschen Ketten," *Mathematische Annalen*, vol. 112, pp. 155–160, 1936. 3

[13] M. A. Marsan, *Stochastic Petri nets: An elementary introduction*, ser. Lecture Notes in Computer Science. Springer, 1990, vol. 424, pp. 1–29. 3, 12

[14] V. Volovoi, "Tutorial: Simulation with stochastic Petri nets," in *Winter Simulation Conference*, L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, Eds., Huntington Beach, CA, December, 6—9 2015. 3, 8

[15] ——, "Correcting for non-Markovian asymptotic effects using Markovian representation," *Arxiv*, p. 1705.01070 [cs.CE], 2017. 4, 11

[16] G. Balbo, "Introduction to generalized stochastic Petri nets," in *Formal Methods for Performance Evaluation*, ser. Lecture Notes in Computer Science, M. Bernardo and J. Hillston, Eds. Springer-Verlag, 2007, vol. 4486, pp. 83–131. 4, 5, 11, 12

[17] P. J. Haas, *Stochastic Petri Nets. Modelling, Stability, Simulation*. New York: Springer, 2002. 4, 12

[18] S. K. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd ed. John Wiley and Sons, 2002. 4

[19] V. V. Volovoi, "Modeling of system reliability using Petri nets with aging tokens," *Reliability Engineering and System Safety*, vol. 84, no. 2, pp. 149–161, 2004. 6

[20] F. Bowden, "A brief survey and synthesis of the roles of time in Petri nets," *Mathematical and Computer Modelling*, vol. 21, pp. 55–68, 2000. 6, 11, 12

[21] T. M. K. Roeder, "An information taxonomy for discrete event simulations," Ph.D. dissertation, University of California, Berkeley, 2004. 8

[22] G. Pang and W. Whitt, "Service interruptions in large-scale service systems," *Management Science*, vol. 55, no. 9, pp. 1499–1512, 2009. 11

[23] H. Lamb, *Hydrodynamics*. Cambridge at the University Press, 1895. 11

[24] L. D. Landau and E. Lifshitz, *Fluid Mechanics*, 2nd ed., ser. Course of Theoretical Physics. Butterworth-Heinemann, 1987, vol. 6. 11

[25] W. van der Aalst, "The application of Petri nets to workflow management," *Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998. 11

[26] A. Law and W. Kelton, *Simulation Modeling and Analysis*, 3rd ed. New York, NY: McGraw-Hill, 2000. 12

[27] S. Christensen and N. D. Hansen, "Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs," in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, M. Ajmone Marsan, Ed. Springer Berlin Heidelberg, 1993, pp. 186–205. 12

[28] H. Matsuno, Y. Tanaka, H. Aoshima, A. Doi, M. Matsui, and S. Miyano, "Biopathways representation and simulation on hybrid functional Petri net," *Silico Biology*, vol. 3, no. 3, pp. 389–404, 2003. 12

[29] G. Ciardo, "Reachability set generation for Petri nets: Can brute force be smart?" in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3099, pp. 17–34. 12

[30] V. Volovoi and R. V. Vega, "On compact modeling of coupling effects in maintenance processes of complex systems," *International Journal of Engineering Science*, vol. 51, pp. 193–210, 2012. 12

[31] A. Avižienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *IEEE COMPSAC 77*, 1977, pp. 149–155. 13

[32] P. Koopman and M. Wagner, "Toward a framework for highly automated vehicle safety validation," in *SAE World Congress*, no. 2018-01-1071. SAE, 2018. 13