Stochastic Activity Networks Templates: Supporting Variability in Performability Models

Leonardo Montecchi, Paolo Lollini, and Andrea Bondavalli, *Member, IEEE*

***Abstract*—Model-based evaluation is extensively used to estimate the performance and reliability of dependable systems. Traditionally, these systems were small and self-contained, and the main challenge for model-based evaluation has been the efficiency of the solution process. Recently, the problem of specifying and maintaining complex models has increasingly gained attention, as modern systems are characterized by many components and complex interactions. Components share similarities, but at the same time, also exhibit variations in their behavior due to different configurations or roles in the system. From the modeling perspective, variations lead to replicating and altering a small set of base models multiple times. Variability is taken into account only informally, by defining a sample model and explaining its possible variations. In this article, we address the problem of including variability in performability models, focusing on stochastic activity networks (SANs). We introduce the formal definition of stochastic activity networks templates (SAN-T), a formalism based on SANs with the addition of variability aspects. Differently from other approaches, parameters can also affect the structure of the model, like the number of cases of activities. We apply the SAN-T formalism to the modeling of the backbone network of an environmental monitoring infrastructure. In particular, we show how existing SAN models from the literature can be generalized using the newly introduced formalism.**

***Index Terms*—Model-based evaluation, parametric models, reuse, stochastic activity networks (SANs), templates.**

I. Introduction

FORMAL methods have been extensively used to estimate the performance and reliability metrics of computer systems. They are especially useful for assessing nonfunctional properties of critical systems, for which experimental approaches are not always applicable. In fact, *model-based evaluation* [1] has the advantage of not exercising the real system, which may be dangerous, costly, or not feasible.

Traditionally, critical systems were isolated and monolithic, and the main challenge in model-based evaluation has always been the solution process, in term of efficiency of state-space generation and accuracy of results. More recently, the problem of *specifying* and *maintaining* complex models has gained attention. Modularization is an established approach in reducing the complexity in the specification of analysis models. However, one of the rising challenges consists in handling variability [2], [3] across system components.

Cyber-physical system-of-systems (CPSoSs) [4] are characterized by a large number of components and complex interactions between them. Many of these elements share similarities, but at the same time, they have a slight different behavior due to their individual configuration or role in the system. These variations lead to replicating and altering a small set of base models multiple times. Furthermore, due to dynamicity and evolution [4], changes to components configurations are introduced over time, and models need to be updated to reflect such changes. Improving variability means anticipating certain kind of changes and make them easier to be implemented [3].

In the dependability [5] and performability [6] domain, many works have proposed approaches to automatically generate formal models from design models (e.g., UML models) enriched with information on the failure/repair processes of components. The idea behind these works is that software and systems engineers can take advantage of formal models without being proficient in them, because model transformations embed the knowledge of experts in an automated "push-a-single-button" tool [7], [8].

While these approaches succeed in providing an application-specific abstraction to users of a certain domain, they are not flexible enough to relieve dependability experts from the effort of modeling complex systems. In fact, they have the following two main limitations:

1) they are tailored to the needs of system designers and not to those of formal methods experts;
2) different transformation algorithms need to be defined for different problems or classes of systems.

In this article, we address the problem of variability in performability models from the point of view of modeling experts, as opposed than targeting software and systems engineers. The focus is on specifying models considering variability, that is,

Manuscript received January 13, 2021; revised June 19, 2021 and September 5, 2021; accepted October 6, 2021. This work was supported in part by the European Union's Horizon 2020 Research and Innovation Program under the Marie Sklodowska-Curie Grant Agreement No. 823788 "ADVANCE", in part by the São Paulo Research Foundation (FAPESP) under Grant 2019/02144-6, and in part by the project POR-CREO SPACE "Smart PAssenger CEnter" funded by the Tuscany Region. Associate Editor: H. Madeira. *(Corresponding author: Leonardo Montecchi.)*

Leonardo Montecchi is with the Institute of Computing, University of Campinas, Campinas, SP 13083-852, Brazil (e-mail: leonardo@ic.unicamp.br).

Paolo Lollini and Andrea Bondavalli are with the Consorzio Interuniversitario Nazionale per l'Informatica (CINI), 00185 Roma, RM, Italy, and also with the Dipartimento di Matematica e Informatica "U. Dini", University of Firenze, 50134 Firenze, FI, Italy (e-mail: paolo.lollini@unifi.it; andrea.bondavalli@unifi.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3120979.

Digital Object Identifier 10.1109/TR.2021.3120979

anticipating and facilitating changes. We propose a new formalism based on stochastic activity networks (SANs) [9] that we call stochastic activity network templates (SAN-Ts). The idea is to leave some parts of an SAN model unspecified, and to make them depend on parameter values. Differently from what is done, for example, in the Möbius tool [10], in SAN-T models, the parameters can also affect the structure of the model, like the number of cases of an activity. The intended users of our formalism are thus experts in performability modeling, in their task of defining libraries of reusable submodels.

This article complements our recent work in [11], in which we defined a framework to improve reuse of performability models. The approach is based on the concept of *libraries of model templates* that interact using well-defined interfaces and composition rules, specified using the template models description language (TMDL). The framework proposed in [11] is not tied to a specific modeling formalism and can be applied in general to models that are composed by superposition of state variables (i.e., "state sharing"). To achieve such generality, we assumed as prerequisite the existence of the following:

1) a *template-level formalism*;
2) an *instance-level formalism*;
3) a *concretize function*, which generates instance-level models from a template-level model.

The work in this article enables the application of the TMDL framework with SANs, since we introduce here a template-level formalism based on SANs and the associated *concretize* function.

The rest of this article is organized as follows. In Section II, we introduce the background and we discuss the related work. In Section III, we present the overall idea of SAN-Ts with a running example, and then, in Section IV, we give their formal definition. In Section V, we define how concrete SAN models can be derived from an SAN-T (i.e., the *concretize* function). Then, in Section VI, we apply the formalism to the modeling of the backbone network of an environment monitoring system. In Section VII, we summarize how this article complements the TMDL framework introduced in [11]. Finally, Section VIII concludes this article.

II. Background and Related Work

A. Model-Based Evaluation

Model-based evaluation [1] consists in estimating system-level metrics through formal models, which typically include stochastic behavior. Model-based evaluation plays a key role in the assessment of critical systems and large-scale infrastructures, where exercising the real system is not feasible.

Various kinds of models can be used for this task. Approaches are typically categorized in combinatorial models and state-space models [1]. *Combinatorial models* describe which combinations of component failures lead to system failure, e.g., fault trees (FTs) [12]. These models are very popular in the industry, as they are simple to understand and they can be evaluated with well-known formulas. However, they assume independent events, and therefore, they cannot represent complex interactions between components or dynamic behavior.

On the other hand, *state-space models* explicitly represent the different states of a system and the possible transitions between them. While being more powerful, these models can quickly become very complex, leading to well-studied problems like state-space explosion and stiffness [1]. One of the most popular formalisms are stochastic Petri nets (SPNs) and their numerous extensions [13]. In particular, the work in this article is based on SANs, which can be considered a variant of SPNs [9], although adopting a different terminology (e.g., *activity* instead of *transition*).

We base our work on SANs because of their wide adoption across different domains, thanks to their flexibility and the support provided by the Möbius tool [14]. For example, recent work has employed models based on SANs to evaluate control strategies of smart grids [15], the availability of a backbone network [16], the performance of scheduling algorithms [17], performability in the railway domain [18], and the quality of experience of a distributed interactive application [19].

B. Stochastic Activity Networks (SAN)

In their semantics, SANs are similar to generalized stochastic Petri nets (GSPNs) [20] and stochastic reward nets (SRNs) [21], in which immediate transitions have priority over timed ones. Similarly to SRNs, SANs may have marking-dependent elements; however, in SANs, various firing time distributions are supported. The *input gate* and *output gate* primitives can be used to specify arbitrary complex predicates for the enabling of transitions (called activities) and for the effects of transition firings. Activities may have multiple probabilistic outcomes, called *cases*.

A formal definition of SANs was given by Sanders and Meyer in [9]. We recall the basic definitions, on which we will base later for the definition of SAN-Ts.

An activity network (AN) is an eight-tuple [9]

$$\text{AN} = (P, A, I, O, \gamma, \tau, \iota, o) \tag{1}$$

where P is a finite set of *places*; A is a finite set of *activities*; I is a finite set of *input gates*; and O is a finite set of *output gates*. The function $\gamma : A \to \mathbb{N}^+$ specifies the number of *cases* for each activity, that is, the number of possible choices upon execution of that activity. $\tau : A \to \{\textit{timed}, \textit{instantaneous}\}$ specifies the type of each activity; $\iota : I \to A$ maps input gates to activities; and $o : O \to \{(a, c) \mid a \in A \wedge c \in \{1, 2, \ldots, \gamma(a)\}\}$ maps output gates to cases of activities.

Similarly to Petri nets (PNs), places can hold *tokens*. The number of tokens in each places determines the state of the network, also called its marking. More formally, if S is a set of places ($S \subseteq P$), a *marking* of S is a mapping $\mu : S \to \mathbb{N}$. The value $\mu(p)$ is the marking of place p, i.e., the number of tokens it holds. The set of possible markings of S is the set of functions $M_S = \{\mu \mid \mu : S \to \mathbb{N}\}$.

An *input gate* is defined as a triple (G, e, f), where $G \subseteq P$ is the set of input places associated with the gate, $e : M_G \to \{0, 1\}$ is the enabling predicate of the gate, and $f : M_G \to M_G$ is the input function of the gate. An *output gate* is a pair (G, f), where

$G \subseteq P$ is the set of output places associated with the gate, and $f : M_G \to M_G$ is the output function.

An input gate $g = (G, e, f)$ *holds* in a marking μ if $e(\mu_G) = 1$. We say that an activity a is *enabled* in a marking μ if all the input gates associated with it hold. Intuitively, the behavior of the network is regulated by the following rules:

1) when an activity is enabled it can *fire*;
2) instantaneous activities have priority over timed activities;
3) when an activity fires, one of its cases is selected.

When an activity a fires in marking μ, the new marking is given by $\mu' = f_{O_n}(\dots f_{O_1}(f_{I_m}(\dots f_{I_1}(\mu))))$, where $g_i = (G_{I_i}, e_{I_i}, f_{I_i})$ is the ith input gate of the activity, and $o_j = (G_{O_j}, f_{O_j})$ is the jth output gate of the selected case. That is, all the functions of all the input gates are computed first, and then, all the functions of the output gates are computed. The complete characterization of the SANs behavior can be found in [9].

A marking in which no instantaneous activities are enabled is a *stable* marking. An activity network is *stabilizing* if, essentially, there is no marking from which it is possible to fire an infinite sequence of instantaneous activities.

Given an AN that is stabilizing in some initial marking $\mu_0 \in M_P$, an SAN is formed by defining functions C_a, F_a, and G_a for each activity a, where: $C_a \in C$ is a function specifying the probability distribution of its cases; $F_a \in F$ is a function specifying the probability distribution of its firing delay; and $G_a \in G$ is a function that describes its reactivation markings [9]. All these functions are allowed to be marking dependent.

$$\text{SAN} = ((P, A, I, O, \gamma, \tau, \iota, o), \mu_0, C, F, G)\,. \tag{2}$$

SANs have an intuitive graphical notation (see [9]). Places are represented as circles, instantaneous activities as thin bars, and timed activities as thick bars. Input gates are represented as left-pointing triangles, while output gates as right-pointing triangles. Cases are represented as small circles next to the activity; if an activity has only one case, the case is omitted from the diagram. Input arcs are considered a special case of the input gate, in which the predicate e holds when there is at least one token in the connected place, and the function f removes one token from that place. Similarly, an output arc is a special case of the output gate in which the function f simply adds one token to the connected place.

Evaluation metrics are defined using *reward structures*, and under certain conditions, the stochastic process underlying an SAN has an exact solution. Otherwise, they can be evaluated by discrete-event simulation, for example, using Möbius [10].

C. Related Work

The problem of simplifying the construction of performability models has been approached in different ways in the literature. Different variants of the original PNs formalism [22] have been defined, some of them enabling more compact and reusable specifications. For example, SRNs [21] contain primitives that allow for a compact specification of SPNs, like marking dependence, variable-cardinality arcs, priorities, etc.

In coloured Petri nets (CPNs) [23], tokens can be distinguished, by attaching information to them. More precisely, tokens can be of different data types, called color sets. Places contain a multiset of tokens of a certain color set, and transitions take into account the color (i.e., value) of tokens in their enabling and firing rules. Hierarchical CPNs support modularization by means of substitution transitions, i.e., a transition is replaced by a whole subnet in a more detailed model. The general formalization encompassing CPNs, basic PNs, and hierarchical PNs is known as high-level Petri nets (HLPNs) [24], [25].

These PNs extensions fold a complex Petri net model into a compact specification. They can address variability aspects to some extent, for example, combining different initial markings and marking-dependent properties. However, variable aspects are limited to the behavior and not to the structure of the model. Besides that, these formalisms can be used to specify concrete models that are directly executed. Using the terminology in [11], CPNs and SRNs can be seen as *instance-level formalisms*. This article focuses on specifying *template-level* models, from which different instances can be derived.

When their color sets are finite, CPNs and HLPNs can be unfolded into a regular PN [23], [26]. Unfolding expands a colored place to multiple normal places, one for each of the possible token colors it can hold. Similarly, a colored transition is expanded to multiple normal transitions, one for each combination of tokens that can enable it. In this perspective, there are some similarities with our *concretize* algorithm described in Section V. However, the general problem of unfolding a CPN is not trivial, because all the combinations of tokens that satisfy transitions guards must be enumerated [27], [28]. Our formalism adds variability to specific aspects of an SAN model, with the intent of defining reusable submodels to be composed according to the methodology in [11]. Being a *template-level formalism*, SAN-T models cannot be directly analyzed.

The work in [29] defined parametric stochastic well-formed nets (PSWNs), a parametric version of SWN. Similarly to our work, PSWNs models are only partially specified, to improve reuse and variability. Parametric behavior is given by export and import functions, which allow color sets and values to be shared between submodels. Composition is achieved by superposition of groups of transitions or places with matching labels. The work has been applied in [30] to the modeling of a fault-tolerant component adopting replica and voting, and it is implemented in the GreatSPN tool [31]. Differently from our proposal, PSWNs models can be instantiated only through composition with other submodels. They however support composition by action synchronization, while we focus on state sharing.

As mentioned earlier, SANs can also be considered a variant of SPNs [9]. In their Möbius implementation [10], they support tokens having different datatypes, including structured datatypes. SANs models can be composed using the Rep/Join state sharing formalism [32]; however, which state variables are composed, and how, must be specified manually. The Möbius implementation of SANs permits using variables, which however can only impact the behavior of the model and not its structure. In this article, we define parametric ("template") SAN models, whose structure *and* behavior can depend on parameters.

A well-established research line focuses on applying model-driven engineering (MDE) [33] techniques to automatically

derive dependability models from UML models or similar representations, e.g., see [7], [8], and [34]. However, such approaches typically provide an application-specific abstraction to users of a certain domain, and then, they automatically derive formal models defined by an expert. Instead, our approach is targeted at dependability modeling experts, and it focuses on constructing reusable models including variability.

The recent work in [35] proposed dependence-aware replication (DARep), an efficient method to replicate SAN models while still maintaining their identity, as opposed to the traditional replica operator in which instances are indistinguishable. The method uses a matrix to specify dependencies across state-variables of instances, and an efficient algorithm [36] to reflect them in the discrete-event simulator generated by Möbius.

The work on DARep is probably the most similar to our proposal. However, there are two main differences between the two approaches. First, DARep focuses on replicating identical SAN models, with alterations in the state variables shared between them. In this aspect, our approach is more general, since it allows variability in the entire structure of an SAN model (instead than on "interfaces" only), and it permits to generate SAN instances that can be composed by either Join or Rep operators. Second, the main motivation behind DARep is to improve the performance of simulation solvers [35], while we focus on the formalization of the concept of SAN-T itself, and we consider solution methods out of the scope of this article.

III. Motivating Example

In this section, we introduce an example to motivate our approach. We first provide an overview of the reference system, then we discuss the challenges in modeling it using ordinary SANs, and finally, we extract a running example to be used in the rest of this article.

A. Brazilian Environmental Data Collection System (BEDCS) System

The BEDCS, also known in Portuguese as *Sistema Brasileiro de Coleta de Dados Ambientais (SBCDA)*, is a large-scale environmental monitoring infrastructure owned by the National Institute of Space Research of Brazil (INPE) [37], [38]. Among the other applications, it supports the monitoring of the Amazon rainforest, both in Brazil as well as in other South American countries [39], [40].

The BEDCS is composed of three segments: space, ground, and application (see Fig. 1). The *application segment* features a deployment of approximately 800 automated platforms scattered throughout the country and at sea. In each of these data collection platforms (DCPs), groups of sensors collect different kinds of environmental data. The *space segment* consists of satellites, which carry a data-collecting transponder and periodically receive the data collected by the DCPs. Satellites also collect data by themselves, typically Earth images or physics-related measurement like density of electrons. Finally, the *ground segment* consists of ground stations for the reception of data from satellites, a center for remote control and tracking, and a mission center responsible for data processing and dissemination to the end users. The data are stored at the mission center, where they are processed and made available to end users through a web interface.

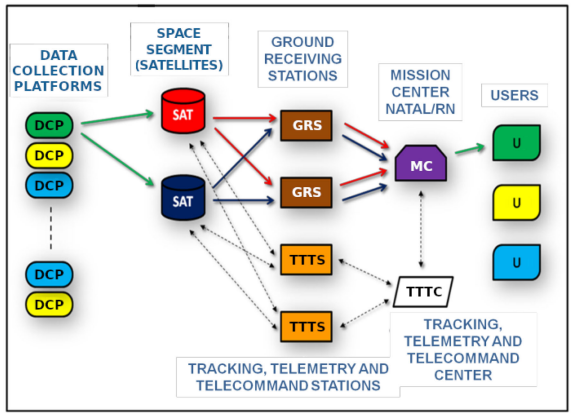

Fig. 1. Network architecture of the Brazilian Environmental Data Collection System (BEDCS). Figure adapted from [41].

Due to the large number of applications relying on it [42], the system has to fulfill strict non-functional requirements; among others: *availability* of the platform to end users; *reliability* of the data collection functionality, in order not to miss data points; and *performance* of data transmission, both DCPs to satellites and satellites to ground station. Note that low performance in data transmission may also cause data loss, in case the buffers of satellites or DCPs become full.

The system, which is in operation since the 90s, has gained importance in the Brazilian and International community over time, resulting in an increasing demand for system modernization, for the provision of new services, and for improvements in performance and dependability. In this context, model-based evaluation is a valuable tool to help engineers understanding the impact of maintenance actions, and to support informed design decisions.

The BEDCS is one of the two case studies of the ADVANCE project [43], whose objective is to define new verification and validation techniques for cyber-physical systems. Within the project, the system is being analyzed according to different points of view, supporting the INPE in its evolutive maintenance efforts.

B. Challenges

The BEDCS is a representative example of a CPSoS: It is composed of a large number of components, with complex interactions between them that possibly change over time, and it is organized as SoS architecture. That is, its constituent systems (CSs) may have different governance and ownership (e.g., satellites and sensors deployed by third parties) and they are put together to provide a higher goal that could not be provided by individual CSs alone.

Model-based evaluation is the primary evaluation means for this kind of system, due to the difficulties in applying experimental approaches, especially at system level. However,

when modeling systems like BEDCS with SANs or similar formalisms, practical issues concerning the scalability arise. Scalability in the solution process is a well-known problem that has been addressed in several ways, e.g., see [1], [31], and [44]. Instead, in this article, we address the challenges concerning the *specification and maintenance* of such models.

In this perspective, the main challenge is represented by the large number of similar components, which exhibit variability aspects due, for example, to different roles, location, or configuration. Furthermore, these aspects can change over time, leading to what in software engineering is known as *variability in space and time* [45]. In the BEDCS case, variability exists in different aspects. As a simple example, each satellite may use a subset of the available communication protocols, which has impact on which DCPs it may receive data from. More in general, different dependencies may exist between system components. Often, a certain relation or global behavior can be described in a general way. However, its *instances* vary depending on the number and kind of components involved, and possibly other context-dependent parameters.

To some extent, this problem is being addressed by modularization. When modeling a complex system with SANs, the complete model is typically built out of a well-defined set of submodels representing specific aspects of the system, which are then composed by state sharing, following predefined rules. Basic building blocks are first identified (e.g., components or functions), and then, examples of the corresponding SAN models are described. Variability is addressed by defining these building blocks in a general way, as "templates," explaining how variants can be derived from a general abstract structure.

This kind of approach has been used by different authors in the construction of models based on SANs, but always in an informal way; see, for example, [15], [16], [19], and [46]. In fact, while an SAN can be used to accurately describe the model of a specific *instance* of a building block, with its specific characteristics, the general structure of the model and its possible variations can only be described with examples or with descriptions in natural language. This is especially true when variations affect the model structure, e.g., number and names of places or number of cases of activities.

The SAN-T formalism that we define in this article aims to solve this problem, by providing a formal way to specify a general (i.e., template) SAN model *and* its possible variations. The idea is to provide an abstract representation of multiple SAN models that exhibit similar structure and behavior, but having some systematic differences that can be parameterized. Then, from such base skeleton, different variants can be generated, based on the values assigned to its parameters.

C. Running Example: The User Model

We introduce here one of the building blocks of the BEDCS model, which will be used as running example.

Among the other things, the BEDCS model has to take into account for different kinds of services (e.g., raw picture data transfer or telemetry to satellites), having different characteristics but a similar behavior. Also, different kinds of user models are needed, having access to different subsets of these services, and accessing services with different probabilities. Metrics of interest are both related to performance (e.g., throughput) and reliability (e.g., disconnection probability).

The basic idea behind SAN-Ts is visualized in Fig. 2. The SANs models in Fig. 2(a) and (b) are adapted from [47], in which we modeled a vehicular network. We chose this example for its simplicity; a more extensive application of the SAN-T formalism is described later in Section VI.

The behavior represented by the two SAN models in Fig. 2(a) and (b) is the following. Each user is initially in idle state, and may then, request a network service. With a certain probability they can request one of the services that are available to them, by adding a token in the corresponding place. While the service is being delivered, a token stays in the place with the corresponding identifier (e.g., `Req1` or `Req6`). The request can fail or be dropped; in these cases, a token is received in the corresponding place, and the user returns to idle state. The figure shows two instances of the model: an *internal* user may request services 1, 6, and 7, with the respective probabilities, while a *press* user may request services 3 and 7 only, with different probabilities.

It is clear that the two models have a similar structure. In fact, they differ only by the number of services available to the user, the identifiers of those services, and the probabilities of the user requesting each service. The structure of these two models can be generalized by establishing the following informal rule: *"Create one place ReqX for each of the services that are available to the user, and name them according to the identifier of these services. The activity request should have the same amount of cases as the number of ReqX places, and each of them should have an output arc connecting the case to the corresponding ReqX place. The probabilities associated with request cases are set based on a vector parameter having the same length as the number of services available to the user."* This would result in an SAN "template," depicted in Fig. 2(c), which abstracts the common structure among models of different users.

Note that the template in Fig. 2(c) does not represent the instances in Fig. 2(a) and (b) only, but in general, any SAN model that follows the same pattern. Having to maintain similar models that only differ from some details is a common issue in the modeling of complex systems. The formal definition of the *User* SAN-T model is discussed later in Section IV-F.

IV. Stochastic Activity Network Templates (SAN-Ts)

A. Overview and Design Choices

In the definition of the SAN-T formalism, we focus on supporting the specification of reusable submodels that are meant to be composed by state sharing (i.e., place superposition), and that can be used with the approach presented in [11]. The objective is to be able to reuse a submodel in different configurations of a global system model. We focus, therefore, on variability with respect to places, which constitute the "interface" of an SAN model with the other submodels.

Another strategy for the composition of SPNs-based models is action synchronization. However, despite the support of tools like Möbius [14], this approach is not particularly common

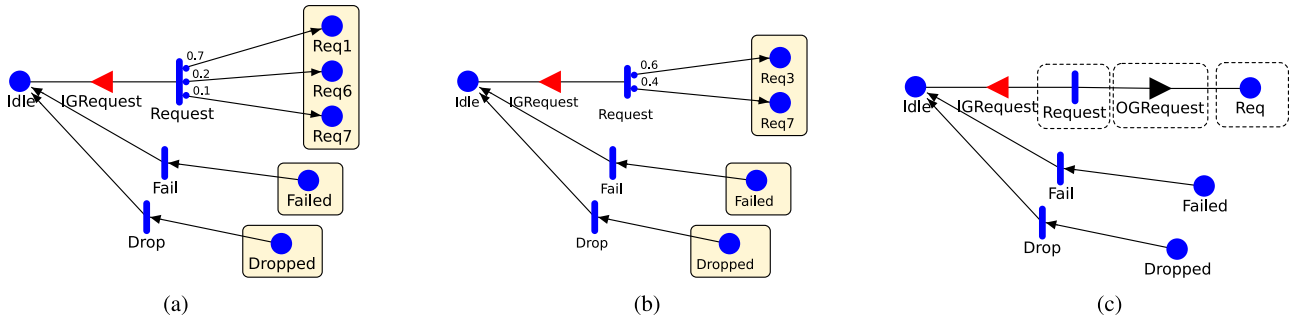

Fig. 2. General idea of SAN-Ts. The two SAN models in (a) and (b) have a similar structure and behavior, and are meant to be connected with other SAN models in the same way. This structure can be abstracted into an SAN-T model as depicted in (c), where dashed elements are template versions of SAN elements, i.e., activity templates and place templates. (a) *UserInternal* SAN model. (b) *UserPress* SAN model. (c) *User* SAN-T model.

in performability evaluation. Possibly, this is because of the problems raised by synchronization of timed transitions (i.e., defining the resulting firing distribution). Also, there is often greater interest in the current state of system components (e.g., healthy, failed, degraded, etc.), rather than in how a component behaves internally. State sharing is, therefore, a more flexible modularization solution for this domain.

Based on these considerations, we allow the *number of places* in an SAN-T model to be parametric, while the *number of activities* is instead fixed. In the proposed approach, replication of activities, or parts of a model involving a sequence of activities, is done by creating multiple instances of a template and composing them as in [11]. The objective of our work is being able to define libraries of small, reusable, templates for performability evaluation based on SANs.

B. Preliminary Definitions

We first introduce some basic notations that will be used in the rest of this article. In particular, the following definitions clarify what is a parameter of a template model, and how it connects to the rest of the formalism.

We adopt the definitions of *sort*, *operator*, *term,* and *assignment* from the ISO/IEC 15909 standard [25], which apply to a wide range of PN-based formalisms, including SANs. According to this formalization, the set of possible values held by a place is defined by its associated *sort* (i.e., type).

A *many-sorted signature* is a pair (S, O), where S is a set of sorts and O is a set of *operators*, together with their arity. The arity is a function $O \to S^* \times S$, where S^* is the set of finite sequences over S, including the empty string ε. The arity function defines, for each operator, the number and sort of its input parameters (S^*), and the sort of the produced result (S). An operator can be denoted as $o_{(\sigma,s)}$, where $\sigma \in S^*$ are the input sorts, and $s \in S$ is the output sort. Constants are operators with empty input sorts, and are denoted as $o_{(\varepsilon,s)}$ or simply o_s.

We denote with Δ a set of parameters; an element of Δ of sort $s \in S$ is denoted with δ_s. $\Delta_s \subseteq \Delta$ is, therefore, the set of parameters of sort s.

Terms of sort $s \in S$ may be built from a signature (S, O) and a set of parameters Δ. Intuitively, these are all the possible expressions of sort s made of any legit combination of operators in O and parameters in Δ [25]. The set of terms of sort s is denoted by $\mathtt{TERM}(O \cup \Delta)_s$. To simplify the notation, in the rest of this article, we will use $\mathtt{TERM}_s$, unless there are ambiguities on the adopted O and Δ sets.

A *many-sorted algebra* $H = (S_H, O_H)$ provides an interpretation of a signature (S, O). For every sort $s \in S$, there is a corresponding set of values $H_s \in S_H$, and for every operator $o_{(s_1 \ldots s_n, s)} \in O$, there is a corresponding function in $o_H \in O_H$, such that $o_H : H_{s_1} \times \ldots \times H_{s_n} \to H_s$.

Given a many-sorted algebra H, and many-sorted parameters in Δ, an *assignment* for Δ under H is a family of functions ξ, comprising a function $\xi_s : \Delta_s \to H_s$ for each sort $s \in S$. The concept of assignment may be extended to terms, thus obtaining the family of functions Val_ξ comprising the function $\mathrm{Val}_{s,\xi} : \mathtt{TERM}_s \to H_s$ for each sort $s \in S$ [25].

To support the subsequent definitions, we require the existence of at least the "integer," "real," "boolean," "ordered set of integers," and "ordered set of reals" sorts, in which we consider sets to be ordered. Formally, we assume a signature (S, O), such that $\{\mathtt{Int}, \mathtt{Real}, \mathtt{Bool}, \mathtt{OrderedSet\{Int\}}, \mathtt{OrderedSet\{Real\}}\} \subseteq S$, and O contains the common operators applicable on such sorts. In particular, besides the standard arithmetic operators, in the rest of this article, we will use the *size* operator, $|x|$, which returns the number of elements in a set x, and the *element at* operator, $x[i]$, which returns the ith element in the ordered set x. The corresponding many-sorted algebra is (S_H, O_H), with $\{\mathbb{N}, \mathbb{R}, \{0,1\}, \mathcal{P}(\mathbb{N}), \mathcal{P}(\mathbb{R})\} \subseteq S_H$, and O_H containing the set of functions corresponding to operators in O, which are not detailed here for simplicity.

C. SAN-T Formal Definition

Based on the previous definitions, we can now introduce the formal definition of SAN-T. Formally, an SAN-T is a tuple

$$\text{SAN-T} = (\Delta, \tilde{P}, \tilde{A}, \tilde{I}, \tilde{O}, \tilde{\gamma}, \tilde{\tau}, \tilde{\iota}, \tilde{o}, \tilde{\mu}_0, \tilde{C}, \tilde{F}, \tilde{G}) \qquad (3)$$

where Δ is a set of parameters, and elements marked with a tilde accent, $\tilde{\cdot}$, are modified versions of elements existing in plain SANs (see Section II-B), reformulated to take parameters into account.

In more details, as follows.

1) Δ is the sorted set of *parameters* of the template.
2) $\tilde{P}$ is a finite set of *place templates*. A place template can be seen as a placeholder for multiple places that, in a regular SAN model, would be strongly related to each other and would vary in different instances of the same template. Based on parameters' values, a place template will be expanded to a precise set of concrete places. Place `Req` in Fig. 2(c) is an example of place template.
   Formally, a place template is defined as a pair (τ, k), where τ is the name of the place, and $k \in \mathtt{TERM}_{\mathtt{OrderedSet\{Int\}}}$ is its multiplicity. Evaluating the term k with respect to an assignment ξ identifies a set of integer indices $K \subset \mathbb{N}$. Such indices determine the set of places to which, with the given assignment of parameters, the place template is expanded. Normal places (i.e., those always expanding to a single place of ordinary SANs) are those for which $\mathrm{Val}_\xi(k) = \{1\}$ for any assignment ξ.
3) $\tilde{A}$ is a finite set of *activity templates*.
4) $\tilde{I}$ is a finite set of *input gate templates*.
5) $\tilde{O}$ is a finite set of *output gate templates*.
6) $\tilde{\gamma} : \tilde{A} \to \mathtt{TERM}_{\mathtt{Int}}$ specifies the *number of cases* for each activity template. For any activity template $\tilde{a} \in \tilde{A}$, evaluating $\tilde{\gamma}(\tilde{a})$ with respect to an assignment ξ yields an integer number, which determines the number of cases of $\tilde{a}$ under that assignment, i.e., $\mathrm{Val}_\xi(\tilde{\gamma}(\tilde{a})) \in \mathbb{N}$.
7) $\tilde{\tau} : \tilde{A} \to \{timed, instantaneous\}$ specifies the *kind of each activity template*, exactly as in ordinary SANs.
8) $\tilde{\iota} : \tilde{I} \to \tilde{A}$ maps *input gate templates to activity templates*.
9) $\tilde{o} : \tilde{O} \to \tilde{A}$ maps *output gate templates to activity templates*.

In order to completely define the elements of an SAN-T, the concept of marking needs to be extended. In particular, we need to take into account for the existence of place templates. In general, the marking of a place template is not a single value (like in ordinary SANs), but a *function*, which associates a value to each index of the place template. We call this function the *marking template* of a place.

Let $F_{\mathbb{N}}$ be the set of all the possible functions $f : \mathbb{N} \to \mathbb{N}$, i.e., $F_{\mathbb{N}} = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$. If $\tilde{S} \subseteq \tilde{P}$ is a set of place templates, a *marking* of $\tilde{S}$ is a mapping $\tilde{\mu} : \tilde{S} \to F_{\mathbb{N}}$. In the particular case in which a place does not have variability, the marking template will be a constant function. The set of possible markings of $\tilde{S}$ is the set of functions $\tilde{M}_{\tilde{S}} = \{\tilde{\mu} \mid \tilde{\mu} : \tilde{S} \to F_{\mathbb{N}}\}$.

Based on these definitions of *marking* and *marking template*, most of the other elements of an SAN-T can be defined by adapting the definitions in [9] to make them depend on the assignment of parameters ξ. In the following, we denote with Ξ the set of all the possible assignments.

An *input gate template* defines an enabling condition for an activity template, and an input function that specifies how the marking is altered by the firing of the activity. An input gate template will always result in a single input gate in the concrete SAN model. Still, the projected output gate may depend on the assignment of parameters. Formally, an *input gate template* is defined as a triple $(\tilde{G}, \tilde{e}, \tilde{f})$, where $\tilde{G} \subseteq \tilde{P}$ is the set of input places associated with the gate, $\tilde{e} : \tilde{M}_{\tilde{G}} \times \Xi \to \{true, false\}$ is the enabling predicate, and $\tilde{f} : \tilde{M}_{\tilde{G}} \times \Xi \to \tilde{M}_{\tilde{G}}$ is the input function.

In ordinary SANs, an output gate defines an output function that is executed upon the firing of an activity. Differently from an input gate, it is associated to an individual case of an activity. In SAN-Ts, an *output gate template* has a similar purpose. However, since the number of cases of an activity template is not known beforehand, the gate is connected directly to the activity. When a regular SAN is generated from the template, an output gate template will be expanded to multiple concrete output gates, depending on the number of cases of the activity to which it is connected.

Formally, an *output gate template* is a pair $(\tilde{G}, \tilde{f})$, where $\tilde{G} \subseteq \tilde{P}$ is the set of output places associated with the gate, and $\tilde{f} : \tilde{M}_{\tilde{G}} \times \mathbb{N} \times \Xi \to \tilde{M}_{\tilde{G}}$ is the output function of the gate. It should be noted that the output function $\tilde{f}$ depends on the index of the case of the associated activity template ($\mathbb{N}$), as well as on the assignment of values to parameters (Ξ).

The probability of cases of an activity template is given by the *case distribution assignment* $\tilde{C}$, which defines a function $\tilde{C}_{\tilde{a}} \in \tilde{C}$ for each activity template $\tilde{a} \in \tilde{A}$. Such functions also depend on parameters, thus, $\tilde{C}_{\tilde{a}} : \tilde{M}_{\tilde{P}(\tilde{a})} \times \mathbb{N}^+ \times \Xi \to [0, 1]$, where $\tilde{P}(\tilde{a})$ is the set of input and output places of the activity. For the model to be well-formed, the following must hold: $(\sum_{1 \le i \le \mathrm{Val}_\xi(\tilde{\gamma}(\tilde{a}))} C_{\tilde{a}}(\mu, i, \xi)) = 1$. This means that the probabilities of the cases of the activity instance should sum to 1, which also implies that the probability of cases beyond those generated with the given assignment ξ should be zero.

Similarly, the firing time of activities is given by the *activity time distribution assignment* $\tilde{F}$, which defines a function $\tilde{F}_a \in \tilde{F}$ for any timed activity template a, with $\tilde{F}_a : \mathbb{R} \times \tilde{M}_{\tilde{P}} \times \Xi \to [0, 1]$. That is, the probability of a certain firing time ($\mathbb{R}$) depends on the marking ($\tilde{M}_{\tilde{P}}$) and on the parameters assignment (Ξ).

The reactivation function of activity templates is given by the reactivation function assignment $\tilde{G}$, such that for any timed activity template a, function $\tilde{G}_a \in \tilde{G}$ defines the reactivation markings, with $\tilde{G}_a : \tilde{M}_{\tilde{P}} \times \Xi \to \wp(\tilde{M}_{\tilde{P}})$ and $\wp(\tilde{M}_{\tilde{P}})$ denoting the power set of $\tilde{M}_{\tilde{P}}$.

Finally, the initial marking of an SAN-T should also depend on the assignment of values to parameters. For this reason, it is defined by the function $\tilde{\mu}_0 : \Xi \to \tilde{M}_{\tilde{P}}$. The original definition of SANs requires the initial marking $\mu_0(\xi)$ to be a stable marking in which the network is stabilizing (see Section II-B). However, because in SAN-T, the actual structure of the model is not completely specified until a value is assigned to all the parameters, we relax this constraint. Well-formedness checks on the structure of the resulting SAN models can be performed at the time of instantiation, based on existing techniques that are applied to ordinary SAN models (e.g., [48]).

D. Arc Templates

One of the distinguishing features of SPNs and their extensions is their convenient graphical notation, which permits describing most aspect of a model using a diagram. In particular, SANs use input arcs and output arcs, represented by arrows in

the diagram, as the graphical representation of particular cases of input gates and output gates, respectively. Following the same idea, we propose a graphical representation of a subset of the new concepts introduced in SAN-Ts.

1) Output Arc Templates: In ordinary SANs, an *output arc* connecting activity a and place p represents an output gate whose function simply adds one token to place p.

Without additional information, drawing a "normal" output arc in an SAN-T can be ambiguous. For example, it is not specified whether the arc should add one token to all the instances of the place template, or only to one of them. Therefore, we extend this concept introducing output arc templates, which can be used in SAN-T models.

Considering an SAN-T model as defined previously, an *output arc template* connects an activity template $a_{\text{src}} \in \tilde{A}$ to a place template $p_{\text{dest}} \in \tilde{P}$. An output arc template has a *label* that defines the function f of the corresponding gate template. The syntax of the label is given by the following grammar.

$$\langle \text{oatlabel} \rangle \models \langle \text{out} \rangle \mid \langle \text{int} \rangle \rightarrow \langle \text{out} \rangle \mid \langle \text{int} \rangle \rightarrow \langle \text{out} \rangle \, / \langle \text{out} \rangle$$

$$\langle \text{out} \rangle \models \langle \text{int} \rangle \mid + \langle \text{int} \rangle$$

$$\langle \text{int} \rangle \models \textit{any integer term } t \in \texttt{TERM}(O \cup \Delta \cup \{\otimes, \odot\})_{\texttt{Int}}.$$

The term $\odot$ is a placeholder for the index of the case of the associated activity template a_{src}, while $\otimes$ is a placeholder for the index of places generated from place template p_{dest}.

A label may specify an unconditional expression, $\langle \text{out} \rangle$, or a conditional expression, $\langle \text{int} \rangle \rightarrow \langle \text{out} \rangle / \langle \text{out} \rangle$.

When an *unconditional expression* is specified, the same expression is used for all the places derived from place template p_{dest}. The expression may specify that the marking of the place(s) should be *set* to a certain value, $\langle \text{int} \rangle$, or that a number of tokens should be *added* to the marking, $+\langle \text{int} \rangle$. The actual value is specified by a term (i.e., expression) of integer type. Note that in this case, the integer term may also include the $\odot$ and $\otimes$ operators, and may, therefore, depend on them. For example, the label "$+3\otimes$" specifies that to each place derived from p_{dest} it should be added a number of tokens equal to three times its index.

A *conditional expression* allows specifying a different expression for a specific instance of the place template. The expression $\langle \text{int} \rangle \rightarrow \langle \text{out} \rangle$ means that for the place template with index $\langle \text{int} \rangle$, the specification $\langle \text{out} \rangle$ will be used, while for all the other ones, the marking is left unchanged. An explicit assignment can be added for the other places, by adding a second $\langle \text{out} \rangle$ element. For example, the label "$1 \rightarrow +2/0$" means that for instance of place p_{dest} having index 1, the marking should be incremented by two tokens, while for all the other instances, it should be set to zero.

When no label is specified, the label "$+1$" is assumed, that is, one token is added to all the instances of the p_{dest} place. When the place template has multiplicity $\{1\}$, and the number of cases of the transition is fixed, an arc with label "$+1$" corresponds to a "normal" output arc.

2) Input Arc Templates: A similar approach can be followed for input arcs. However, in this case, the label must also specify the input predicate, in addition to the input function.

An *input arc template* connects a place template $p_{\text{src}} \in \tilde{P}$ to an activity template $a_{\text{dest}} \in \tilde{A}$. The *label* of an input arc template is defined by the following grammar, where $\otimes$ is a placeholder for the index of places derived from p_{src}.

$$\langle \text{iatlabel} \rangle \models [\langle \text{pred} \rangle \,] \langle \text{func} \rangle \mid - \langle \text{int} \rangle$$

$$\langle \text{pred} \rangle \models \forall \langle \text{cond} \rangle \mid \exists \langle \text{cond} \rangle \mid \langle \text{int} \rangle \, \langle \text{cond} \rangle$$

$$\langle \text{cond} \rangle \models = \langle \text{int} \rangle \mid > \langle \text{int} \rangle \mid \geq \langle \text{int} \rangle$$

$$\langle \text{func} \rangle \models \langle \text{int} \rangle \mid - \langle \text{int} \rangle$$

$$\langle \text{int} \rangle \models \textit{any integer term } t \in \texttt{TERM}(O \cup \Delta \cup \{\otimes\})_{\texttt{Int}}.$$

The label of an input gate template may specify explicitly both the predicate and the input function, $[\langle \text{pred} \rangle \,] \langle \text{func} \rangle$, or the input function only, $-\langle \text{int} \rangle$, leaving the predicate implicit. The input function $\langle \text{func} \rangle$ is specified in a similar way as for output arc templates: it can set the marking of all the places to a specific value, or it can subtract a certain number of tokens.

The predicate is composed of a quantifier and a condition. The condition $\langle \text{cond} \rangle$ specifies a condition on the marking of the connected place template, while the quantifier specifies on which instances of that place the condition should hold.

The predicate $\forall \langle \text{cond} \rangle$ is true when the markings of all the concrete places derived from p_{src} satisfy the condition $\langle \text{cond} \rangle$. The predicate $\exists \langle \text{cond} \rangle$ is true if at least one place instance satisfies that condition. Finally, the predicate $\langle \text{int} \rangle \, \langle \text{cond} \rangle$ is true if the marking of the place instance having index corresponding to $\langle \text{int} \rangle$ satisfies the condition. When the predicate is stated explicitly, the input function is applied only to places satisfying the condition, that is: all the places in the case of $\forall \langle \text{cond} \rangle$; only the places satisfying the condition in the case of $\exists \langle \text{cond} \rangle$; and only the place with index $\langle \text{int} \rangle$ in the case of $\langle \text{int} \rangle \, \langle \text{cond} \rangle$.

For example, the label "$[\exists = 1] \; 0$" means the following:

1) the predicate is true if at least one of the place instances has exactly one token;
2) the function sets the marking of all the place instances satisfying the condition to zero.

When no $\langle \text{pred} \rangle$ term is specified, the input function may only remove a certain number of tokens, and the predicate is considered to hold if all the place instances contain at least that number of tokens. If no label is specified at all, "-1" is assumed, that is, the gate is enabled if all the instances of the place contain at least one token, and the function removes one token from all of them. Similarly as for output arc templates, an arc with label "-1" connected to place template with multiplicity $\{1\}$ corresponds to a "normal" input arc.

E. Graphical Notation

Besides the extended notation for input arc templates and output arc templates, we also adopt some conventions in the graphical representation of SAN-Ts models.

The main difference in notation is that "template elements" in the model are surrounded with a dashed line, as previously shown in Fig. 2(c). With *template elements*, we mean model elements that carry some variability aspect, in particular the following:

1) place templates having nonunary multiplicity (i.e., $k \neq \{1\}$);
2) activities with a variable number of cases (i.e., $\tilde{\gamma}(\tilde{a})$ is not constant);
3) input gate templates connected to a place template with nonunary multiplicity;
4) output gate templates connected to a place template with nonunary multiplicity;
5) output gate templates connected to an activity having a variable number of cases.

Highlighting elements that have variability helps the modeler to better understand which parts of the model will change in the concrete SANs instances.

F. *Example:* User *SAN-T Model*

We now apply the proposed formulation of SAN-Ts to the running example introduced in Section III-C. Here, we show how such SAN-T can be specified in a formal way, according to the definitions given in the previous section.

As a support to the specification in this section, and to those provided later, we define the following functions in $F_{\mathbb{N}}$, which can thus be used to define the marking of the SAN-T (see Section IV-C).

$$f^k(x) = k, \quad \forall x \in \mathbb{N};$$

$$\left[f_j^k(h)\right](x) = \begin{cases} k, & \text{if } x = j \\ h(x), & \text{otherwise} \end{cases}$$

$$\left[f_J(g,h)\right](x) = \begin{cases} g(x), & \text{if } x \in J \\ h(x), & \text{otherwise} \end{cases} \quad J \subseteq \mathbb{N}. \tag{4}$$

Basically, this is a simplified notation for functions defining the marking template of an SAN-Ts. Function f^k is a constant marking template, which assigns the same marking to all the place instances; $f_j^k(h)$ uses the value k for place instance having index j, and the function h for the others; and $f_J(g,h)$ uses function g for places whose index is in J and function h for the others.

Two parameters can be identified for the `User` template model. The first, s, identifies the number and indices of services that the user can access, and it is, therefore, of type "ordered set of integers." The second, pb, determines the probabilities of being selected of the different services, and it is of type "ordered set of reals."

The variable elements of the model are essentially the activity template `Request`, its associated output gate template `OGRequest`, and the place template `Req`. The `Request` activity has a variable number of cases, given by the cardinality of the array of integers assigned to parameter s, and each of these cases is selected with a probability given by parameter pb. Place template `Req` is expanded to a number of concrete places that is again given by the cardinality of s. The selection of case i of the `Request` activity template results in the addition of a token in place $\texttt{Req}_i$.

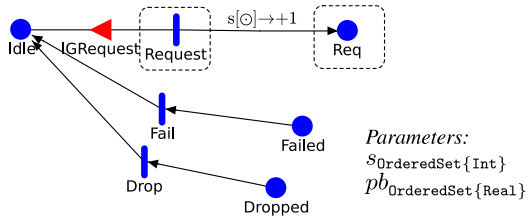

Fig. 3. *User* SAN-T model with the proposed notation for output arc templates.

The graphical representation of the model, using the proposed notation, is shown in Fig. 3. The output arc template connecting activity `Request` and place `Req` has the label "$s[\odot] \rightarrow +1$," which can be interpreted as follows: "*The ith concrete output gate adds one token to the instance of the Req place having index $s[i]$, and leaves the other places unchanged.*" The complete formal specification of the *User* SAN-T model can be found in the Appendix (see Appendix A).

All the place templates have multiplicity 1, except for place `Req`; similarly, all the activities have a fixed number of cases except for activity `Request`. The initial marking is 1 (actually, the function f^1) for place template `Idle`, and 0 (actually, the function f^0) for all the others.

The case distribution function assigns probability 1 to the first and only case of activity templates `Drop` and `Fail`, while for the cases of the `Req` activity template, the probability of the ith case is given by the ith value of parameter pb, which is of type "ordered set of reals."

Most of the variability in this model is contained in the output gate template `OGrequest`, which is defined as follows:

$$\tilde{G}_{\texttt{OGRequest}} = \{\texttt{Req}\}$$

$$\tilde{f}_{\texttt{OGRequest}}(\tilde{\mu}, i, \xi) = f_{s[i]}^{m+1}\left(\tilde{\mu}(\tilde{p})\right) \ \forall \tilde{p} \in \tilde{G}_{\texttt{OGRequest}} \tag{5}$$

where $m = [\tilde{\mu}(\tilde{p})](s[i])$, that is, the current value assigned by the marking template to the $s[i]$th instance of the place $\tilde{p}$. Summarizing, the function of the output gate associated with the ith case should add one token into the instance of the `Req` having index $s[i]$, and leave the other places unchanged.

V. Generation of SAN-T Instances

To actually use SAN-T models for model-based evaluation, concrete instances must be generated, by assigning values to their parameters. Such instances, which are ordinary SAN models, can be evaluated in isolation, or composed into larger models using the TMDL framework [11] or the plain Rep/Join formalism [32].

A. *Overview*

Instances are generated by the *concretize* function, described in the following, which generates an ordinary SAN model from a from a pair (S_Δ, ξ). That is, it generates an SAN model from an

SAN-T model S_Δ and an assignment of values to its parameters ξ.

Given an SAN-T S_Δ

$$S_\Delta = (\Delta, \tilde{P}, \tilde{A}, \tilde{I}, \tilde{O}, \tilde{\gamma}, \tilde{\tau}, \tilde{\iota}, \tilde{o}, \tilde{\mu}_0, \tilde{C}, \tilde{F}, \tilde{G}) \tag{6}$$

and a parameter assignment function ξ, the *concretize* function generates an SAN model S^ξ as

$$S^\xi = (P^\xi, A^\xi, I^\xi, O^\xi, \gamma^\xi, \tau^\xi, \iota^\xi, o^\xi, \mu_0^\xi, C^\xi, F^\xi, G^\xi). \tag{7}$$

The rest of this section describes how its elements are derived from the SAN-T specification. We separate the algorithm in the following two parts:

1) concretization of the individual places, markings, and gates;
2) concretization of the overall model structure.

In all the following definitions, ξ is the assignment of parameters from which the instance should be generated.

B. Places, Marking, and Gates

1) Places: For each *place template* in the SAN-T model, one or more "normal" places are created in the instance model. How many places are created, and with which indices, is given by applying the assignment function on the multiplicity of the place template.

Formally, given a place template $\tilde{p} = (\tau, k) \in \tilde{P}$ of the SAN-T model, and being $\text{Val}_\xi(k) = \{a_1, \ldots, a_m\}$ the indices obtained from applying the assignment function to the multiplicity specification, the places $\{\tau^\xi_{a_1}, \ldots, \tau^\xi_{a_m}\}$ are created in the concrete SAN model.

We denote with $\Pi(\tilde{p}, i) \in P^\xi$ the ith concrete place originating from place template $\tilde{p}$. That is, $\Pi(\tilde{p}, i) = \tau^\xi_{a_i}$.

2) Marking: From any given *marking* of an SAN-T model, a unique mapping to a marking of the generated SAN instance can be identified. Essentially, this is done by applying the marking template function to the index of the generated places.

Formally, given a marking of the SAN-T model, $\tilde{\mu} \in \tilde{M}_{\tilde{P}}$, the marking $\mu^\xi \in M_P$ of the instance model is defined as

$$\mu^\xi(\tau_{a_i}) = f_{\tilde{p}}(a_i) \qquad \forall \tilde{p} \in \tilde{P} \qquad \forall i \in \mathbb{N} \tag{8}$$

where

$$\tau_{a_i} = \Pi(\tilde{p}, i), \qquad f_{\tilde{p}} = \mu(\tilde{p}). \tag{9}$$

That is, the marking of the ith place (τ_{a_i}) generated from place template $\tilde{p}$ is obtained by applying the *marking template* function ($f_{\tilde{p}}$) to the index of the concrete place (a_i).

Given a marking $\tilde{\mu} \in \tilde{M}_{\tilde{P}}$ of the SAN-T model, we denote the corresponding marking $\mu^\xi \in M_P$ of the generated instance as $\Gamma(\tilde{\mu})$. Conversely, given a marking μ^ξ of the concrete (generated) SAN model, we denote as $\Gamma^{-1}(\mu^\xi)$ the corresponding marking $\tilde{\mu}$ of the originating SAN-T.

3) Input Gates: Each *input gate template* of the SAN-T model is translated to exactly one input gate in the SAN instance. Given an input gate template $\tilde{g} = (\tilde{G}, \tilde{e}, \tilde{f}) \in \tilde{I}$, we denote with $\alpha(\tilde{g})$ the corresponding input gate $g^\xi = (G^\xi, e^\xi, f^\xi) \in I^\xi$ in the concrete SAN model, which is obtained as

$$\begin{aligned} G^\xi &= \left\{\Pi(\tilde{p}, j) \mid \tilde{p} = (\tau, k) \in \tilde{G},\ j \in \text{Val}_\xi(k)\right\} \\ e^\xi(\Gamma(\tilde{\mu})) &= \text{Val}_\xi(\tilde{e}\,(\tilde{\mu})) \\ f^\xi(\Gamma(\tilde{\mu})) &= \tilde{f}\,(\tilde{\mu}, \xi)\,. \end{aligned} \tag{10}$$

That is, the input places of the concrete input gate, G^ξ, are all the places generated from place templates in $\tilde{G}$; the input predicate applied to a marking $\Gamma(\tilde{\mu})$ is the result of applying the assignment function to the predicate of the gate template; and the input function applied to marking $\Gamma(\tilde{\mu})$ is the input function of the gate template applied on marking $\tilde{\mu}$ and assignment ξ.

4) Output Gates: Differently from input gate templates, each *output gate template* may be expanded to one or more concrete output gates. The number of concrete output gates that should be generated depends on parameters, and more specifically, from the parameter that controls the number of cases of the connected activity.

Given an output gate template $(\tilde{G}, \tilde{f}) \in \tilde{O}$, we denote with $\beta(\tilde{g}, i)$ the ith output gate $(G^\xi_i, f^\xi_i) \in O$ generated from it in the SAN model, which is obtained as

$$\begin{aligned} G^\xi_i &= \left\{\Pi(\tilde{p}, j) \mid \tilde{p} = (\tau, k) \in \tilde{G},\ j \in \text{Val}_\xi(k)\right\} \\ f^\xi_i(\Gamma(\tilde{\mu})) &= \tilde{f}\,(\tilde{\mu}, i, \xi)\,. \end{aligned} \tag{11}$$

That is, the output places G^ξ are all the places generated from output place templates in $\tilde{G}$, and the output function applied to marking $\Gamma(\tilde{\mu})$ is the output function of the gate template applied on marking $\tilde{\mu}$, index i, and assignment ξ.

C. Overall SAN Definition

We can now provide the complete specification of the SAN derived from an SAN-T S_Δ and an assignment ξ. That is, we can precisely define all the elements in (7), as follows:

$$\begin{aligned} P^\xi &= \bigcup_{\tilde{p}=(\tau,k)\in\tilde{P}} \{\Pi(\tilde{p}, i) \mid i \in \text{Val}_\xi(k)\} \\ A^\xi &= \tilde{A} \\ \gamma^\xi(a) &= \text{Val}_\xi(\tilde{\gamma}\,(\tilde{a})) \\ I^\xi &= \left\{\alpha(\tilde{g}) \mid \tilde{g} \in \tilde{I}\right\} \\ O^\xi &= \bigcup_{\tilde{g}\in\tilde{O}} \{\beta(\tilde{g}, 1), \ldots, \beta(\tilde{g}, \text{Val}_\xi(\tilde{\gamma}\,(\tilde{a}))) \mid \tilde{a} = \tilde{o}(\tilde{g})\} \\ \tau^\xi &= \tilde{\tau} \\ \iota^\xi(\alpha(g)) &= \tilde{\iota}(g) \quad \forall g \in \tilde{I} \\ o^\xi(\beta(g, i)) &= \tilde{o}(g) \quad \forall g \in \tilde{O} \quad \forall i \in \{1, \ldots, \text{Val}_\xi(\tilde{\gamma}\,(\tilde{a}))\} \\ \mu_0^\xi &= \tilde{\mu}_0(\xi). \end{aligned} \tag{12}$$

The rationale behind the aforementioned derivation can be summarized as follows:

1) the set of places P^ξ is given by all the places derived from all the place templates in $\tilde{P}$;
2) the set of activities remains unchanged;
3) the function γ, which specifies the number of cases of an activity, is the result of applying the assignment function to the $\tilde{\gamma}$ function;
4) there is an input gate in I^ξ for each input gate template in $\tilde{I}$;
5) each output gate template in $\tilde{O}$ is expanded to a certain number of output gates, given by the number of cases of the activity to which it is connected;
6) the function τ that determines if an activity is timed or instantaneous remains unchanged;
7) if an input gate template is connected to an activity template, then its concrete projection is connected to the projection of the activity template;
8) if an output gate template is connected to an activity template, then all its concrete projections are connected to the projection of the activity template;
9) the initial marking μ_0^ξ is given by the initial marking of the SAN-T model, applied to the assignment ξ.

Furthermore, the following.

1) For each function $\tilde{C}_{\tilde{a}}$ in the case distribution assignment $\tilde{C}$, a corresponding function C_a^ξ is included in C^ξ, defined as $C_a^\xi(\Gamma(\mu), k) = \tilde{C}_{\tilde{a}}(\mu, k, \xi)\ \forall \mu \in M_{\tilde{P}}\ \forall k \in \mathbb{N}^+$.
2) For each function $\tilde{F}_{\tilde{a}}$ in the activity time distribution assignment $\tilde{F}$, a corresponding function F_a^ξ is included in F^ξ, defined as $F_a^\xi(\Gamma(\mu), r) = \tilde{F}_{\tilde{a}}(\mu, r, \xi)\ \forall \mu \in M_{\tilde{P}}\ \forall r \in \mathbb{R}$.
3) For each function $\tilde{G}_{\tilde{a}}$ in the reactivation function assignment $\tilde{G}$, a corresponding function G_a^ξ is added to G^ξ, defined as $G_a^\xi(\Gamma(\mu)) = \{\Gamma(\tilde{\mu}) \mid \tilde{\mu} \in \tilde{G}_a(\tilde{\mu})\}\ \forall \mu \in \tilde{M}_{\tilde{S}}$.

D. Example: Instances of the User SAN-T

Following the *concretize* algorithm described in the previous section, we show here how it is possible to derive multiple instance of the *User* SAN-T depicted in Fig. 3. Note that it is the same model as the one in Fig. 2(c) drawn with the new notation for arc templates. Its formal definition is given in (19) in the Appendix (see Appendix A).

We show how the two concrete SAN models of Fig. 2(a) and 2(b) can be derived by different assignments of parameters. We define two different assignment functions, $\xi_{\texttt{UserInternal}}$ and $\xi_{\texttt{UserPress}}$, which will result in the generation of the two SAN instances

$$\begin{aligned}\xi_{\texttt{UserInternal}} &= \{(s, \{1, 6, 7\}), (p, \{0.7, 0.2, 0.1\})\}\\ \xi_{\texttt{UserPress}} &= \{(s, \{3, 7\}), (p, \{0.6, 0.4\})\}.\end{aligned} \tag{13}$$

The generation of the two instances follows a similar process; we discuss only one of them in details. The SAN model *UserInternal* is derived by the template–assignment pair $(S_{\texttt{User}}, \xi_{\texttt{UserInternal}})$, resulting in the SAN model detailed in the Appendix (see Appendix B).

Among its elements, it is worth detailing the generation of the output gate template `OGRequest`, which contains variability. In fact, in the SAN-T model, it is connected to an activity template with a variable number of cases.

According to the algorithm, the number of concrete output gates that are generated is given by $\text{Val}_\xi(\tilde{\gamma}(\tilde{a}))$, where $\tilde{a}$ is the associated activity template. In our case, $\text{Val}_{\xi_{\texttt{UserInternal}}}(\tilde{\gamma}(\texttt{Request})) = 3$, and therefore, three output gates are created: $\texttt{OGRequest}_1$, $\texttt{OGRequest}_2$, and $\texttt{OGRequest}_3$. Each of these output gates is connected to the corresponding case of the `Request` activity (see o^ξ in (21) in the Appendix B).

Their definition, in terms of its input places and output function, is obtained by (11). For the *UserInternal* instance, they are defined as follows:

$$\begin{aligned}
&\texttt{OGRequest}_1 = (G_1, f_1), \qquad G_1 = \{\texttt{Req}_1, \texttt{Req}_6, \texttt{Req}_7\}\\
&f_1(\mu) = \mu_1'\ \forall \mu \in M_G \mid \mu_1'(p) = \begin{cases}\mu(p)+1 & \text{if } p = \texttt{Req}_1\\ \mu(p) & \text{otherwise.}\end{cases}\\
&\texttt{OGRequest}_2 = (G_2, f_2), \qquad G_2 = \{\texttt{Req}_1, \texttt{Req}_6, \texttt{Req}_7\}\\
&f_2(\mu) = \mu_2'\ \forall \mu \in M_G \mid \mu_2'(p) = \begin{cases}\mu(p)+1, & \text{if } p = \texttt{Req}_6\\ \mu(p), & \text{otherwise.}\end{cases}\\
&\texttt{OGRequest}_3 = (G_3, f_3), \qquad G_3 = \{\texttt{Req}_1, \texttt{Req}_6, \texttt{Req}_7\}\\
&f_3(\mu) = \mu_3'\ \forall \mu \in M_G \mid \mu_3'(p) = \begin{cases}\mu(p)+1, & \text{if } p = \texttt{Req}_7\\ \mu(p), & \text{otherwise.}\end{cases}
\end{aligned} \tag{14}$$

Each of the three gates adds a token to the corresponding place generated from place template `Req`, corresponding to the output arcs appearing in Fig. 2(a).

VI. Application to the BEDCS Network

In this section, we show how the proposed SAN-T formalization can be used to model a real system. We apply the formalism to the modeling of the BEDCS, and in particular, we discuss here the modeling of its backbone network.

To demonstrate the generality of our formalism, we base the BEDCS network model on the work in [16], in which the objective of the authors was to evaluate a backbone network in Norway, detailing failure correlation between system components. The model in [16] is based on SANs, and it has been defined in a modular way as typically done in the literature. However, as discussed in the motivations for this article, the authors of [16] described model elements only by examples, because they include variability aspects that cannot be represented using SANs alone.

Here, we show how the SAN-T formalism can accurately specify such SAN-based models, including their variability aspects. We emphasize that the authors of [16] defined their models without any collaboration with the authors of this article, and they are not involved in this work.

A. Models of the Backend Network

We focus here on the modeling of the ground sector, and in particular, of the backbone network connecting the ground

stations. In fact, the ground segment is currently composed of four ground stations located in different states of Brazil: two receiving stations, in Cuiabá (MT) and Alcântara (MA), the mission center in Natal (RN), and the remote control center in São José dos Campos (SP) [41]. The two closest stations are more than 1000-km apart.

As mentioned previously, we adopt the approach of [16] to model the BEDCS backbone network. That work fits particularly well to our problem, because the authors provided models for different architectural options, including traditional network infrastructures, as well as those based on software-defined networks (SDNs). Furthermore, the model takes into account for different kinds of correlation between component failures, due to, e.g., physical proximity.

The modeling approach in [16] defines different kinds of building blocks for the system model, grouped in two categories: *Component Blocks* and *Dependence Blocks*.

The *Component Blocks* are simple SAN models that represent physical components of the system architecture. In particular, the authors consider "template" models for a `Link`, an `IPRouter`, a `SDNSwitch`, and a `SDNController`. The variability in these models is given only by the rates associated with the firing of activities, and the probabilities associated with their cases. Therefore, these models can be represented by plain SANs.

The *Dependence Blocks* are used to model the occurrence of dependent failures between components. The authors consider seven kinds of dependencies between components.

1) *Geographical proximity (GEO)*, when a small geographical distance results in common sensitivity to bad weather and natural disasters.
2) *Physical proximity (PHY)*, which causes a strong failure correlation (e.g., blackout).
3) *Common O&M (COM)*, in which the operation and maintenance (O&M) is actually the same for multiple network elements.
4) *Misconfiguration (MIS)*, when elements share the same configuration or have a correlated logic.
5) *Compatibility issue (CIS)*, when a simultaneous failure may occur on multiple components due to incompatibility issues.
6) *Homogeneous equipment (HEQ)*, that is, when a failure happens in a network element, another element with the same equipment may likely fail as well.
7) *Traffic migration (TMI)*, when a network element fails and its replacement is not able to take over.

Specific SAN models that represent these dependencies are "plugged" in the overall system model, according to the scenario to be represented. While these models have been described with examples in [16], they have variability that cannot be expressed with plain SANs, for example, how many components, and which ones, are involved in the dependence.

B. Modeling With SAN-Ts

We discuss here how SAN-Ts can be used to formally specify such dependence blocks, in particular, for the GEO and TMI

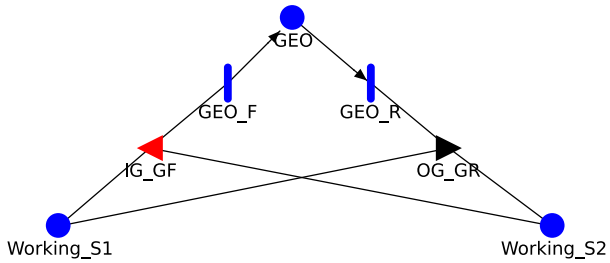

Fig. 4. Example of the GEO building block specified with SANs, for two components. Figure reproduced from [16].

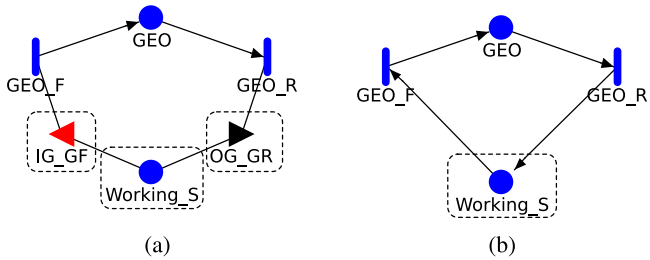

Fig. 5. (a) Generalized GEO building block specified with SAN-T, for any number of components. (b) Simplified notation using input arc templates and output arc templates.

dependencies. Models for the other dependencies can be defined in a similar way.

1) GEO Dependence Block: The original GEO dependence block as defined in [16] is depicted in Fig. 4. The general idea of the block is as follows.

Places `Working_S1` and `Working_S2` represent the working state of the two components involved in the dependence, in this case, two SDN switches (S1 and S2). If place `Working_SX` contains a token it means that the corresponding component is currently working. When both components are working, the activity `GEO_F` is enabled, meaning that the GEO common cause failure may occur. Once the failure has occurred, restoration is possible after some time, represented by the timed activity `GEO_R`. Restoration makes the involved components working again, by adding a token to the `Working_SX` places.

This block has been defined, as an example, for two components only. However, the GEO dependence may involve three or more switches, and in general, any number of components. The block can be generalized, informally as follows: *"For each switch X involved in the dependence create a place* `Working_SX`. *The enabling predicate of the input gate* `IG_GF` *is true when all the places* `Working_SX` *contain a token, and the input function removes all the tokens from those places. The output function of gate* `OG_GR` *adds a token to all the* `Working_SX` *places."*

Using the proposed SAN-T formalism, the generalized "template" version of the block can be defined in a precise way, as follows. The corresponding graphical representation of the model is depicted in Fig. 5(a) using gates, and in Fig. 5(b), using the compact notation with arc templates. Note that the arc between `Working_S` and `GEO_F` is an input arc template

having the default label "-1" (which is thus hidden). Similarly, the arc between GEO_R and Working_S is an output arc template, having the default label "$+1$."

The SAN-T model has three parameters: n, the identifiers of the components involved in the dependence; λ^f, the rate of occurrence of the GEO failure; and λ^r, the restoration rate.

$$
\begin{aligned}
\text{SAN-T}_{\texttt{GEO}} &= (\Delta, \tilde{P}, \tilde{A}, \tilde{I}, \tilde{O}, \tilde{\gamma}, \tilde{\tau}, \tilde{\iota}, \tilde{o}, \tilde{\mu}_0, \tilde{C}, \tilde{F}, \tilde{G}) \\
\Delta &= \{n_{\texttt{OrderedSet\{Int\}}}, {\lambda^f}_{\texttt{Real}}, {\lambda^r}_{\texttt{Real}}\} \\
\tilde{P} &= \{(\texttt{GEO}, 1), (\texttt{Working\_S}, n)\} \\
\tilde{A} &= \{\texttt{GEO\_F}, \texttt{GEO\_R}\} \\
\tilde{I} &= \{\texttt{IG\_GF}, \texttt{GEOtoGEO\_R}\} \\
\tilde{O} &= \{\texttt{OG\_GR}, \texttt{GEO\_FtoGEO}\} \\
\tilde{\gamma} &= \{(\texttt{GEO\_F}, 1), (\texttt{GEO\_R}, 1)\} \\
\tilde{\tau} &= \{(\texttt{GEO\_F}, \textit{timed}), (\texttt{GEO\_R}, \textit{timed})\} \\
\tilde{\iota} &= \{(\texttt{IG\_GF}, \texttt{GEO\_F}), (\texttt{GEOtoGEO\_R}, \texttt{GEO\_R})\} \\
\tilde{o} &= \{(\texttt{OG\_GR}, \texttt{GEO\_R}), (\texttt{GEO\_FtoGEO}, \texttt{GEO\_F})\} \\
\tilde{\mu}_0(\xi) &= \mu'(\tilde{p})\ \forall \xi \mid \mu'(\tilde{p}) = \begin{cases} f^1, & \text{if } \tilde{p} = \texttt{Working\_S} \\ f^0, & \text{otherwise.} \end{cases} \\
\tilde{C} &= \{\tilde{C}_{\texttt{GEO\_F}}, \tilde{C}_{\texttt{GEO\_R}}\} \qquad (16) \\
\tilde{C}_{\texttt{GEO\_F}}(\tilde{\mu}, i, \xi) = \tilde{C}_{\texttt{GEO\_R}}(\tilde{\mu}, i, \xi) &= \begin{cases} 1, & \text{if } i = 1 \\ 0, & \text{otherwise} \end{cases} \\
\tilde{F} &= \{\tilde{F}_{\texttt{GEO\_F}}, \tilde{F}_{\texttt{GEO\_R}}\}, \qquad \tilde{G} = \{\tilde{G}_{\texttt{GEO\_F}}, \tilde{G}_{\texttt{GEO\_R}}\}.
\end{aligned}
\tag{15}
$$

As in the previous examples, the $\tilde{G}_{\texttt{GEO\_F}}$ and $\tilde{G}_{\texttt{GEO\_F}}$ functions are empty, as none of the activities is reactivating. The firing distributions are negative exponential distributions based on the λ^f and λ^r parameters, that is, $\tilde{F}_{\texttt{GEO\_F}}(t) = 1 - e^{-\lambda^f \cdot t}$ and $\tilde{F}_{\texttt{GEO\_R}}(t) = 1 - e^{-\lambda^r \cdot t}$. The gates GEO_FtoGEO and GEOtoGEO_R correspond to the two arcs incoming to and outgoing from the GEO place, respectively, which are "normal" arcs as in ordinary SANs.

Conversely, the gates IG_GF and OG_GR are where the variability in the behavior resides. Intuitively, the input gate template IG_GF specifies that the activity is be enabled if all the instances of Working_S contain at least one token, and that when the activity fires, it removes all the tokens from all of them. The function of the output gate template OG_GF sets the marking of all the instances of Working_S to 1, meaning that all the components have been repaired. Their formal definitions are detailed in the following.

$$
\begin{aligned}
\texttt{IG\_GF} &= (\tilde{G}_{\texttt{IF\_GF}}, \tilde{e}_{\texttt{IF\_GF}}, \tilde{f}_{\texttt{IF\_GF}}) \\
\tilde{G}_{\texttt{IG\_GF}} &= \{\texttt{Working\_S}\} \\
\tilde{e}_{\texttt{IG\_GF}}(\tilde{\mu}, \xi) &= \bigwedge_{i \in \text{Val}_\xi(n)} \Big(\tilde{\mu}_{\tilde{p}}(i) > 0\Big), \quad \text{with } \tilde{\mu}_{\tilde{p}} = \tilde{\mu}(\tilde{p}) \\
\tilde{f}_{\texttt{IG\_GF}}(\tilde{\mu}, i, \xi) &= f^0 \quad \forall \tilde{p} \in \tilde{G}_{\texttt{IG\_GF}}
\end{aligned}
$$

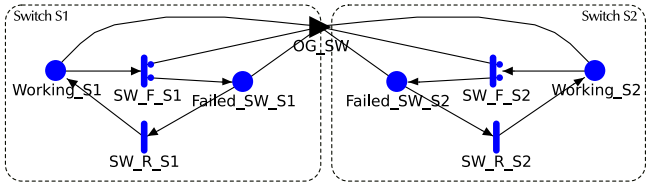

Fig. 6. Example of the TMI dependence between switches S1 and S2, modeled with SANs. Figure adapted from [16].

$$
\begin{aligned}
\texttt{OG\_GR} &= (\tilde{G}_{\texttt{OG\_GR}}, \tilde{f}_{\texttt{OG\_GR}}) \\
\tilde{G}_{\texttt{OG\_GR}} &= \{\texttt{Working\_S}\} \\
\tilde{f}_{\texttt{OG\_GR}}(\tilde{\mu}, i, \xi) &= f^1 \quad \forall \tilde{p} \in \tilde{G}_{\texttt{OG\_GR}}.
\end{aligned}
\tag{16}
$$

2) TMI Dependence Block: The example in Fig. 6 shows the introduction of the TMI dependence among two SDN switches. The idea is that upon software failure of one of them, there is a probability that traffic migration also causes the second switch to fail. This dependence does not actually add a new block to the system model, but instead it modifies the existing SAN models of the involved components.

The model in Fig. 6 shows the SAN models of the two switches (S1 on the left and S2 on the right), and a new output gate OG_SW that represents the dependence. With respect to the normal model of the switch, a new case is added to the failure activity (SW_F_S1 and SW_F_S2), and it is connected to the newly introduced OG_SW output gate. It should be noted that the figure contains a slight abuse of notation (as in the original article), because formally an output gate can be connected to only one activity. We consider, therefore, two identical copies of the gate, OG_SW1 and OG_SW2, each connected to the first case of one of the two activities.

As in the previous case, this dependence can span multiple switches; more in general, each switch can affect a different subset of the switches in the system. This dependence can be generalized and formalized by modeling the SDN switch as an SAN-T. Which switches will be affected by the TMI dependence, and whether the dependence must be represented at all, will be specified by the parameters of the template. Note that the template *will represent only a single switch*; a model equivalent to the one in Fig. 6 is then obtained by instantiation and composition of *two instances*.

A graphical representation of the resulting SAN-T model is provided in Fig. 7, while its formal specification is provided in (17). The template has five parameters: k, the index of the switch represented by the instance; J, an array of identifiers of other switches that can be affected when the switch fails; $p^{\texttt{TMI}}$, the probability that the TMI dependence occurs; and λ^f and λ^r as failure and repair rates of the switch, respectively.

$$
\begin{aligned}
\text{SAN-T}_{\texttt{SwitchTMI}} &= (\Delta, \tilde{P}, \tilde{A}, \tilde{I}, \tilde{O}, \tilde{\gamma}, \tilde{\tau}, \tilde{\iota}, \tilde{o}, \tilde{\mu}_0, \tilde{C}, \tilde{F}, \tilde{G}) \\
\Delta &= \{k_{\texttt{Int}}, J_{\texttt{OrderedSet\{Int\}}}, p^{\texttt{TMI}}_{\texttt{Real}}, {\lambda^f}_{\texttt{Real}}, {\lambda^r}_{\texttt{Real}}\}
\end{aligned}
$$

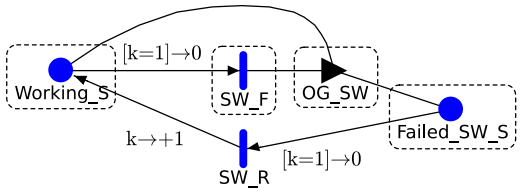

Fig. 7. SAN-T model of the SDN switches considering the TMI dependence in a general way.

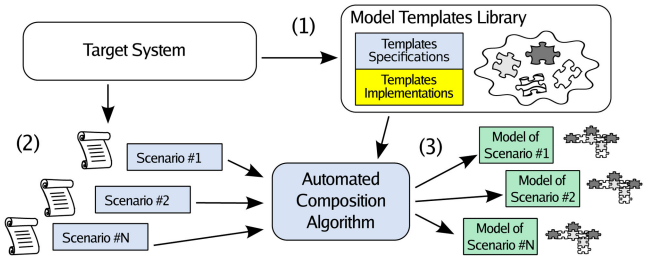

Fig. 8. Workflow of the TMDL framework for the automated generation of performability models [49].

$$
\begin{aligned}
\tilde{P} &= \{(\texttt{Working\_S}, J \cup \{k\}),\\
&\quad\ (\texttt{Failed\_SW\_S}, J \cup \{k\})\}\\
\tilde{A} &= \{\texttt{SW\_F}, \texttt{SW\_R}\}\\
\tilde{I} &= \{\texttt{Working\_StoSW\_F}, \texttt{Failed\_SW\_StoSW\_R}\}\\
\tilde{O} &= \{\texttt{OG\_SW}, \texttt{SW\_RtoWorking\_S}\}\\
\tilde{\gamma} &= \{(\texttt{SW\_F}, 1 + (p^{\texttt{TMI}}_{\texttt{Real}} > 0)), (\texttt{SW\_R}, 1)\}\\
\tilde{\tau} &= \{(\texttt{SW\_F}, \textit{timed}), (\texttt{SW\_R}, \textit{timed})\}\\
\tilde{\iota} &= \{(\texttt{Working\_StoSW\_F}, \texttt{SW\_F}),\\
&\qquad (\texttt{Failed\_SW\_StoSW\_R}, \texttt{SW\_R})\}\\
\tilde{o} &= \{(\texttt{OG\_SW}, \texttt{SW\_F}), (\texttt{SW\_RtoWorking\_S}, \texttt{SW\_R})\}\\
\tilde{\mu}_0(\xi) &= \mu'(\tilde{p})\ \forall \xi \mid \mu'(\tilde{p}) = \begin{cases} f^1 \text{ if } \tilde{p} = \texttt{Working\_S},\\ f^0 \text{ otherwise.} \end{cases}\\
\tilde{C} &= \{\tilde{C}_{\texttt{SW\_F}}, \tilde{C}_{\texttt{SW\_R}}\}\\
\tilde{C}_{\texttt{SW\_F}}(\tilde{\mu}, i, \xi) &= \begin{cases} 1 - p^{\texttt{TMI}}_{\texttt{Real}}, & \text{if } i = 1\\ p^{\texttt{TMI}}_{\texttt{Real}}, & \text{if } i = 2\\ 0, & \text{otherwise.} \end{cases}\\
\tilde{C}_{\texttt{SW\_R}}(\tilde{\mu}, i, \xi) &= \begin{cases} 1, & \text{if } i = 1,\\ 0, & \text{otherwise.} \end{cases}\\
\tilde{F} &= \{\tilde{F}_{\texttt{SW\_F}}, \tilde{F}_{\texttt{SW\_R}}\}, \qquad \tilde{G} = \{\tilde{G}_{\texttt{SW\_F}}, \tilde{G}_{\texttt{SW\_R}}\}.
\end{aligned}
\tag{17}
$$

Places `Working_S` and `Failed_SW_S` are place templates, and their multiplicity is given by the union of index of the switch and those of the switches that should be affected by the TMI failure. That is, the generated SAN instances would contain a place `Working_S`k and a place `Failed_SW_S`k for the switch represented by the instance, and a place `Working_S`j and `Failed_SW_S`j for each other switch $j \in J$ that can be affected by the dependence.

Gates `Working_StoSW_F`, `Failed_SW_StoSW_R`, and `SW_RtoWorking_S` correspond to the arc templates depicted in the figure. The label "$[k = 1] \to 0$" of input arc `Working_StoSW_F` means that the activity is enabled if the instance having index k contains exactly one token, and the marking of that place is set to zero upon firing. The same applies to the label of `Failed_SW_StoSW_R`. The label "$k \to +1$" of output arc `SW_RtoWorking_S` means that one token is added to the place instance having index k, while the others remain unchanged.

The specification of the output gate template `OG_SW` is detailed in the following. Basically, for the first case of the activity, it adds one token only to the place with the same index as parameter k, and for the second case, it also adds one token to the places corresponding to the affected switches, and removes the token from their `Working_S` place.

$$
\begin{aligned}
\texttt{OG\_SW} &= (\tilde{G}_{\texttt{OG\_SW}}, \tilde{f}_{\texttt{OG\_SW}})\\
\tilde{G}_{\texttt{OG\_SW}} &= \{\texttt{Working\_S}, \texttt{Failed\_SW\_S}\}\\
\tilde{f}_{\texttt{OG\_SW}}(\tilde{\mu}, i, \xi) &= \mu'_i(\tilde{p}) \quad \forall \tilde{p} \in \tilde{G}_{\texttt{OG\_SW}} \mid\\
\mu'_1(\tilde{p}) &= \begin{cases} f^1_k(\tilde{\mu}(\tilde{p})), & \text{if } \tilde{p} = \texttt{Failed\_SW\_S}\\ \tilde{\mu}(\tilde{p}), & \text{otherwise.} \end{cases}\\
\mu'_2(\tilde{p}) &= \begin{cases} f_J(0, \tilde{\mu}(\tilde{p})), & \text{if } \tilde{p} = \texttt{Working\_S}\\ f_{\{k\}\cup J}(1, \tilde{\mu}(\tilde{p})), & \text{if } \tilde{p} = \texttt{Failed\_SW\_S}. \end{cases}
\end{aligned}
\tag{18}
$$

VII. TDML Framework

The work in this article complements the TMDL framework that we defined in [11]. In this section, we briefly recall it, and discuss the relation with the work in this article.

The idea behind the TMDL framework is organized in the following three steps:

1) there exists a *library* of parametric reusable submodels, defined with a template-level formalism, and called *model templates*;
2) based on the scenario to be modeled, a set of templates is selected and proper parameters are assigned;
3) models in the instance-level formalism are automatically generated and assembled to obtain the overall system model.

The corresponding workflow is detailed in Fig. 8.

In Step #1, a *library* of reusable *model templates* is created by an expert. In Step #2, the different system configurations that should be analyzed are defined in terms of "scenarios." Scenarios are composed of *model variants*, that is, a selection of model templates with their parameter assignment. In Step #3, all the needed *model instances* are automatically created and assembled, thus generating the complete system model for each scenario. Note that the steps in the workflow are not strictly

sequential. In particular, the creation of the model library is performed once, and the library is stored for future access.

What makes the model templates reusable is that they have well-defined *interfaces* and *parameters*. Briefly summarizing, interfaces specify how they can be connected to other templates, while parameters make it possible to derive different concrete models from the same template. A model template has a *specification* (of its parameters and interfaces), and an *implementation*.

The specification of a template is provided with the TMDL, a domain-specific language specifically defined for that purpose. The implementation of a template can be *atomic* or *composite*. A composite implementation simply specifies which other templates can be composed and how, and it is also specified with the TMDL. The implementation of an atomic template should be given using a *template-level formalism*, that is, a modeling formalism that defines partially specified models. Conversely, we call *instance-level formalism* the modeling formalism concretely used for the analysis, generated in Step #3 (e.g., "normal" SANs).

In [11], we introduced some assumptions, both for simplicity but also to keep the approach independent of a specific modeling formalism. In particular, we assumed that for a certain instance-level formalism (e.g., SANs), it was possible to define the following:

1) a corresponding *template-level formalism*, to specify model templates;
2) a *concretize* function that, given a model in the template-level formalism and an assignment of values to its parameters, generates a model in the instance-level formalism;
3) a notion of *compatibility* between the TMDL specification of a template (i.e., interfaces and parameters) and its implementation with the template-level formalism.

In this article, we have provided a formal definition of SAN-Ts, and of the corresponding concretize function. These definitions enable the application of the TMDL framework considering SANs as the instance-level formalism. In particular, an SAN-T model $\mathcal{S}$ can be used as the *atomic implementation* of any TMDL template $\mathcal{T}$ that is compatible with it. For $\mathcal{T}$ to be compatible with $\mathcal{S}$, the following two conditions must hold:

1) for each metavariable in an interface of $\mathcal{T}$, there should be a corresponding place template in $\mathcal{S}$;
2) for each parameter specified in $\mathcal{T}$, there should be a corresponding parameter in $\mathcal{S}$.

VIII. Conclusion

In this article, we proposed a formal definition of SAN-Ts, a formalism that generalizes SANs with the addition of variability aspects. SAN-T models defined abstract models depending on parameters, from which concrete SAN models can be generated by assignment of values. We demonstrated the applicability of the formalism by using it for the generalization of SAN-based models present in the literature. The proposed formalization can accurately describe the variation points present in the models, and at the same time, provide a compact notation, thanks to the proposed extensions to the graphical notation of SANs.

This work complements our work in [11], in which we defined an approach to simplify the composition of models based on SPNs. As future work, we are working on different directions. The first one is to apply the methodology for the actual evaluation of a real system. In this article, we showed parts of the model of the BEDCS systems, a large-scale infrastructure for environmental monitoring in Brazil. Further work is ongoing, in collaboration with the INPE, within the ADVANCE project [43].

Second, we have not addressed how metrics were defined at the template level. A straightforward adaptation of reward variables to SAN-Ts consists in defining operators that allow reasoning on the marking of the different instances of the place, in a similar way as done for arc templates in Section IV-D. Further work is needed in this direction. However, we note that often metrics need to be defined at the system level; it is thus not possible to avoid defining metrics on the final composed model.

Finally, we are working on providing tool support for the proposed formalism, to facilitate its application by other researchers. This task involves an expanded and more precise formalization for the arc templates notation, and in general, a simplified user-friendly notation for specifying SAN-T models. A prototype editor for SAN-Ts models, based on the eclipse modeling framework and the Sirius modeling tools is being developed as open source software. The architecture of the editor and tool under development has been presented in [49].

Appendix

This section contains some definitions that were omitted in the main body of the document to improve legibility. In particular, it contains the formal specification of the SAN-T model in Fig. 3 (see Appendix A), and the formal specification of its concretization to yield the SAN models in Fig. 2(a) and (b) (see Appendix B).

A. *Example:* User *SAN-T Model*

The formal specification of the *User* SAN-T model is as follows.

$$\begin{aligned}
\text{SAN-T}_{\texttt{User}} &= (\Delta, \tilde{P}, \tilde{A}, \tilde{I}, \tilde{O}, \tilde{\gamma}, \tilde{\tau}, \tilde{\iota}, \tilde{o}, \tilde{\mu}_0, \tilde{C}, \tilde{F}, \tilde{G}) \\
\Delta &= \{s_{\texttt{OrderedSet\{Int\}}}, pb_{\texttt{OrderedSet\{Real\}}}\} \\
\tilde{P} &= \{(\texttt{Idle}, 1), (\texttt{Req}, s), (\texttt{Dropped}, 1), \\
&\qquad (\texttt{Failed}, 1)\} \\
\tilde{A} &= \{\texttt{Request}, \texttt{Fail}, \texttt{Drop}\} \\
\tilde{I} &= \{\texttt{IGRequest}, \texttt{ArcInFail}, \texttt{ArcInDrop}\} \\
\tilde{O} &= \{\texttt{OGRequest}, \texttt{ArcOutFail}, \texttt{ArcOutDrop}\} \\
\tilde{\gamma} &= \{(\texttt{Request}, |s|), (\texttt{Fail}, 1), (\texttt{Drop}, 1)\} \\
\tilde{\tau} &= \{(\texttt{Request}, \textit{timed}), (\texttt{Fail}, \textit{instantaneous}), \\
&\qquad (\texttt{Drop}, \textit{instantaneous})\}
\end{aligned}$$

$$\tilde{\iota} = \{(\texttt{IGRequest}, \texttt{Request}), (\texttt{ArcInFail}, \texttt{Fail}), (\texttt{ArcInDrop}, \texttt{Drop})\}$$
$$\tilde{o} = \{(\texttt{OGRequest}, \texttt{Request}), (\texttt{ArcOutFail}, \texttt{Fail}), (\texttt{ArcOutDrop}, \texttt{Drop})\}$$
$$\tilde{\mu}_0(\xi) = \mu'(\tilde{p}) \ \forall \xi \mid \mu'(\tilde{p}) = \begin{cases} f^1, & \text{if } \tilde{p} = \texttt{Idle} \\ f^0, & \text{otherwise.} \end{cases}$$
$$\tilde{C} = \{\tilde{C}_{\texttt{Request}}, \tilde{C}_{\texttt{Drop}}, \tilde{C}_{\texttt{Fail}}\} \quad (20)$$
$$\tilde{C}_{\texttt{Request}}(\tilde{\mu}, i, \xi) = \begin{cases} Val_\xi(pb_i) & \text{if } 1 \le i \le |s| \\ 0 & \text{otherwise.} \end{cases}$$
$$\tilde{C}_{\texttt{Drop}}(\tilde{\mu}, i, \xi) = \tilde{C}_{\texttt{Fail}}(\tilde{\mu}, i, \xi) = \begin{cases} 1, & \text{if } i = 1 \\ 0, & \text{otherwise.} \end{cases}$$
$$\tilde{F} = \{\tilde{F}_{\texttt{Request}}\}$$
$$\tilde{G} = \{\tilde{G}_{\texttt{Request}}, \tilde{G}_{\texttt{Drop}}, \tilde{G}_{\texttt{Fail}}\}. \quad (19)$$

In the reference model presented in [47], the firing time of the `Request` activity is regulated by a *uniform* distribution, thus $\tilde{F}_{\texttt{Request}}$ is set accordingly. None of the activities are reactivating, that is, $\tilde{G}_{\texttt{Request}} = \tilde{G}_{\texttt{Fail}} = \tilde{G}_{\texttt{Drop}} = \emptyset$, or, in other words, the set of reactivating markings is empty.

B. Example: Instances of the User SAN-T

We define two different assignment functions, $\xi_{\texttt{UserInternal}}$ and $\xi_{\texttt{UserPress}}$, which will result in the generation of the following two SAN instances:

$$\xi_{\texttt{UserInternal}} = \{(s, \{1, 6, 7\}), (p, \{0.7, 0.2, 0.1\})\}$$
$$\xi_{\texttt{UserPress}} = \{(s, \{3, 7\}), (p, \{0.6, 0.4\})\}. \quad (20)$$

The generation of the two instances follows a similar process, and for this reason, we show only one of them in details. The SAN model *UserInternal* is derived by the template–assignment pair $(S_{\texttt{User}}, \xi_{\texttt{UserInternal}})$, resulting in the following SAN model, where $\texttt{ActivityName}(k)$ denotes the kth case of the (concrete) activity `ActivityName`.

$$\text{SAN}_{\texttt{UserInternal}} = (P^\xi, A^\xi, I^\xi, O^\xi, \gamma^\xi, \tau^\xi, \iota^\xi, o^\xi, \mu_0^\xi, C^\xi, F^\xi, G^\xi)$$
$$P^\xi = \{\texttt{Idle}_1, \texttt{Req}_1, \texttt{Req}_6, \texttt{Req}_7, \texttt{Dropped}_1, \texttt{Failed}_1\}$$
$$A^\xi = \{\texttt{Request}, \texttt{Fail}, \texttt{Drop}\}$$
$$I^\xi = \{\texttt{IGRequest}, \texttt{ArcInFail}, \texttt{ArcInDrop}\}$$
$$O^\xi = \{\texttt{OGRequest}_1, \texttt{OGRequest}_2, \texttt{OGRequest}_3, \texttt{ArcOutFail}, \texttt{ArcOutDrop}\}$$
$$\gamma^\xi = \{(\texttt{Request}, 3), (\texttt{Fail}, 1), (\texttt{Drop}, 1)\}$$
$$\tau^\xi = \{(\texttt{Request}, \textit{timed}), (\texttt{Fail}, \textit{instantaneous}), (\texttt{Drop}, \textit{instantaneous})\}$$
$$\iota^\xi = \{(\texttt{IGRequest}, \texttt{Request}), (\texttt{ArcInFail}, \texttt{Fail}), (\texttt{ArcInDrop}, \texttt{Drop})\}$$
$$o^\xi = \{(\texttt{OGRequest}_1, \texttt{Request}(1)), (\texttt{OGRequest}_2, \texttt{Request}(2)), (\texttt{OGRequest}_3, \texttt{Request}(3)), (\texttt{ArcOutFail}, \texttt{Fail}(1)), (\texttt{ArcOutDrop}, \texttt{Drop}(1))\}$$
$$\mu_0^\xi(p) = \begin{cases} 1, & \text{if } p = \texttt{Idle}_1, \\ 0, & \text{otherwise.} \end{cases}$$
$$C^\xi = \{C_{\texttt{Request}}, C_{\texttt{Drop}}, C_{\texttt{Fail}}\}$$
$$C_{\texttt{Request}}(\tilde{\mu}, i) = \begin{cases} 0.7, & \text{if } i = 1 \\ 0.2, & \text{if } i = 2 \\ 0.1, & \text{if } i = 3 \\ 0, & \text{otherwise.} \end{cases}$$
$$C_{\texttt{Drop}}(\tilde{\mu}, i) = C_{\texttt{Fail}}(\tilde{\mu}, i) = \begin{cases} 1, & \text{if } i = 1 \\ 0, & \text{otherwise.} \end{cases}$$
$$F^\xi = \{F_{\texttt{Request}}\},$$
$$G^\xi = \{G_{\texttt{Request}}, G_{\texttt{Drop}}, G_{\texttt{Fail}}\}. \quad (21)$$

Elements F^ξ and G^ξ are not discussed in details, since their derivation is straightforward for this model. Also, as discussed before, there are no reactivation markings, and therefore, the functions $G_{\texttt{Request}}$, $G_{\texttt{Drop}}$, and $G_{\texttt{Fail}}$ are in this case the empty function.

According to (12), the number of concrete output gates that are generated from each output gate template is given by $\text{Val}_\xi(\tilde{\gamma}(\tilde{a}))$, where $\tilde{a}$ is the associated activity template. In our case, $\text{Val}_{\xi_{\texttt{UserInternal}}}(\tilde{\gamma}(\texttt{Request})) = 3$, and therefore, three output gates are created: $\texttt{OGRequest}_1$, $\texttt{OGRequest}_2$, and $\texttt{OGRequest}_3$. Each of these output gates is connected to the corresponding case of the `Request` activity [see o^ξ in (21)].

The definition of each output gate, in terms of its input places and output function, is obtained by (11). For the *UserInternal* instance, they are defined in (14).

REFERENCES

[1] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: From dependability to security," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 48–65, Jan.–Mar. 2004.

[2] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and variability in software engineering," *IEEE Softw.*, vol. 15, no. 6, pp. 37–45, Nov./Dec. 1998.

[3] J. van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Proc. Work. IEEE/IFIP Conf. Softw. Architecture*, 2001, pp. 45–54.

[4] A. Bondavalli, S. Bouchenak, and H. Kopetz, Eds., *Cyber-Physical Systems of Systems - Foundations - A. Conceptual Model and Some Derivations: The AMADEOS Legacy, (Programming and Software Engineering)*, vol. 10099. New York, NY, USA: Springer, 2016.

[5] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[6] J. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Trans. Comput.*, vol. C- 29, no. 8, pp. 720–731, Aug. 1980.

[7] L. Montecchi, P. Lollini, and A. Bondavalli, "Towards a MDE transformation workflow for dependability analysis," in *Proc. 16th IEEE Int. Conf. Eng. Complex Comput. Syst.*, Las Vegas, NV, USA, 2011, pp. 157–166.

[8] S. Bernardi, J. Merseguer, and D. C. Petriu, "Dependability modeling and analysis of software systems specified with UML," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–48, 2012.

[9] W. Sanders and J. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on Formal Methods and Performance Analysis (Lecture Notes in Computer Science)*. Berlin, Germany: Springer, 2002, vol. 2090, pp. 315–343.

[10] G. Clark *et al.*, "The mobius modeling tool," in *Proc. 9th Int. Workshop Petri Nets Perform. Models*, 2001, pp. 241–250.

[11] L. Montecchi, P. Lollini, and A. Bondavalli, "A template-based methodology for the specification and automated composition of performability models," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 293–309, Mar. 2020.

[12] M. Stamatelatos *et al.*, "Fault tree handbook with aerospace applications," *NASA Office of Safety and Mission Assurance*, Washington, DC, USA, Aug. 2002.

[13] G. Ciardo, R. German, and C. Lindemann, "A characterization of the stochastic process underlying a stochastic petri net,"*IEEE Trans. Softw. Eng.*, vol. 20, no. 7, pp. 506–515, Jul. 1994.

[14] T. Courtney *et al.*, "Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models," in *Proc. 39th IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Estoril, Portugal, 2009, pp. 353–358.

[15] S. Chiaradonna, F. Di Giandomenico, and G. Masetti, "A stochastic modelling framework to analyze smart grids control strategies," in *Proc. IEEE Smart Energy Grid Eng.*, Oshawa, ON, Canada, Aug. 21–24, 2016, pp. 123–130.

[16] G. Nencioni, B. E. Helvik, and P. E. Heegaard, "Including failure correlation in availability modeling of a software-defined backbone network," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 1032–1045, Dec. 2017.

[17] R. Entezari-Maleki *et al.*, "Performance aware scheduling considering resource availability in grid computing," *Eng. Comput.*, vol. 33, no. 2, pp. 191–206, Jul. 2016.

[18] L. D. da Silva, D. Mongelli, P. Lollini, A. Bondavalli, and G. Mando, "Performability analysis of a tramway system with virtual tags and local positioning," in *Proc. IEEE 9th Latin- Amer. Symp. Dependable Comput.*, Nov. 2019, pp. 1–10.

[19] N. Veeraragavan *et al.*, "Modeling QoE in dependable tele-immersive applications: A. case study of world opera," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2667–2681, Sep. 2016.

[20] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Trans. Comput. Syst.*, vol. 2, no. 2, pp. 93–122, 1984.

[21] J. K. Muppala, G. Ciardo, and K. S. Trivedi, "Stochastic reward nets for reliability prediction," *Commun. Rel., Maintainability Serviceability*, vol. 1, no. 2, pp. 9–20, 1994.

[22] C. A. Petri, "Communication with automata," Ph.D. dissertation, Universität Hamburg, Hamburg, Germany, 1966.

[23] K. Jensen, *Coloured Petri Nets*. New York, NY, USA: Springer-Verlag, 1996.

[24] K. Jensen, and G. Rozenberg, Eds., *High-Level Petri Nets: Theory and Application*. New York, NY, USA: Springer-Verlag, 1991.

[25] *Systems and Software Engineering—High-Level Petri nets - Part 1: Concepts, Definitions and Graphical Notation*, ISO/IEC 15909-1:2004, Dec. 2004.

[26] T. Murata, "Some recent applications of high-level petri nets," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1991, pp. 818–821.

[27] F. Liu, M. Heiner, and M. Yang, "An efficient method for unfolding colored Petri nets," in *Proc. IEEE Winter Simul. Conf.*, Dec. 2012, pp. 1–12.

[28] M. Schwarick, C. Rohr, F. Liu, G. Assaf, J. Chodak, and M. Heiner, "Efficient unfolding of coloured petri nets using interval decision diagrams," in *Proc. Appl. Theory Petri Nets Concurrency*, 2020, pp. 324–344.

[29] P. Ballarini, S. Donatelli, and G. Franceschinis, "Parametric stochastic well-formed nets and compositional modelling," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer, 2000, pp. 43–62.

[30] S. Bernardi, S. Donatelli, and A. Horváth, "Implementing compositionality for stochastic Petri nets," *Int. J. Softw. Tools Technol. Transfer*, vol. 3, no. 4, pp. 417–430, Sep. 2001.

[31] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis, "30 years of GreatSPN," in *Principles of Performance and Reliability Modeling and Evaluation (Springer Series in Reliability Engineering)*. Cham, Switzerland: Springer, 2016, pp. 227–254.

[32] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 1, pp. 25–36, Jan. 1991.

[33] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[34] M. Cinque, D. Cotroneo, and C. Di Martino, "Automated generation of performance and dependability models for the assessment of wireless sensor networks," *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 870–884, Jun. 2012.

[35] G. Masetti, F. Di Giandomenico, and S. Chiaradonna, "A stochastic modeling approach for an efficient dependability evaluation of large systems with non-anonymous interconnected components," in *Proc. IEEE 28th Int. Symp. Softw. Rel. Eng.*, Oct. 2017, pp. 46–55.

[36] G. Masetti, S. Chiaradonna, F. Di Giandomenico, B. Feddersen, and W. H. Sanders, "An efficient strategy for model composition in the möbius modeling environment," in *Proc. IEEE 14th Eur. Dependable Comput. Conf.*, Sep. 2018, pp. 116–119.

[37] M. A. Chamon, "Scientific and technological satellites at INPE/BRAZIL," in *Proc. 57th Int. Astronautical Congr.*, to be published, Oct. 2006, doi: 10.2514/6.IAC-06-B5.2.01.

[38] R. Pereira., M. dos Santos, M. Lima-Marques, and M. Mattiello-Francisco, "Improving satellite data archiving facility for environmental R&D purposes based on architecture of information approach," in *Proc. SpaceOps Conf.*, Jun. 2012, vol. 11, p. 15, doi: 10.2514/6.2012-1295180.

[39] *BNDES Approves R 23 Million to Monitor the Amazon Forest in Other South American Countries*, Amazon Fund Newslett., no. 37, Apr. 2013.

[40] E. Cabrera, G. Galindo, and D. Vargas, "Protocolo de Procesamiento Digital de Imágenes para la Cuantificación de la Deforestacin en Colombia, Nivel Nacional Escala Gruesa y Fina," *Instituto de Hidrolog-a, Meteorologa, y Estudios Ambientales (IDEAM).* Bogot D.C., Colombia, 2011.

[41] M. J. M. de Carvalho, J. S. dos Santos Lima, L. dos Santos Jotha, and P. S. de Aquino, "CONASAT: Constelao de Nano Satlites para Coleta de Dados Ambientais," in *Proc. Anais XVI Simpósio Brasileiro d Sensoriamento Remoto*, Foz do Iguaçu, Brazil, Apr. 13–18, 2013, pp. 9108–9115.

[42] L. M. G. Fonseca, J. C. N. Epiphanio, D. M. Valeriano, J. V. Soares, J. C. L. Dalge, and M. A. Alvarenga, "Earth observation applications in Brazil with focus on the CBERS program," *IEEE Geosci. Remote Sens. Mag.*, vol. 2, no. 2, pp. 53–55, Jun. 2014.

[43] "ADVANCE: Addressing verification and validation challenges in future cyber-physical systems," H2020 MSCA-RISE Grant 823788, Nov. 17, 2021. [Online]. Available: https://cordis.europa.eu/project/id/823788

[44] G. Ciardo, Y. Zhao, and X. Jin, "Ten years of saturation: A petri net perspective," in *Transactions on Petri Nets and Other Models of Concurrency V (Lecture Notes in Computer Science)*. Berlin, Germany: Springer, 2012, pp. 51–95.

[45] C. Seidl, I. Schaefer, and U. A**mann, "Integrated management of variability in space and time in software families," in *Proc. 18th Int. Softw. Product Line Conf.*, 2014, pp. 22–31.

[46] N. Ge, M. Pantel, and S. D. Zilio, "Formal verification of user-level real-time property patterns," in *Proc. 11th Int. Symp. Theor. Aspects Softw. Eng.*, Sophia Antipolis, France, Sep. 13–15, 2017, pp. 1–8.

[47] A. Bondavalli, P. Lollini, and L. Montecchi, "QoS perceived by users of ubiquitous UMTS: Compositional models and thorough analysis," *J. Softw.*, vol. 4, no. 7, pp. 675–685, 2009.

[48] D. D. Deavours and W. H. Sanders, "An efficient well-specified check," in *Proc. 8th Int. Workshop Petri Nets Perform. Models*, 1999, pp. 124–133.

[49] L. Montecchi, F. Moncini, P. Lollini, and K. Keefe, "An eclipse-based editor for SAN templates," in *Proc. 12th Int. Workshop Softw. Eng. Resilient Syst.*, Munich, Germany, 2020, pp. 159–167.