HPnGs go non-linear: statistical dependability evaluation of battery-powered systems

Carina Pilch
Westfälische Wilhelms-Universität
Münster, Germany
carina.pilch@wwu.de

Mathis Niehage
Westfälische Wilhelms-Universität
Münster, Germany
m_nieh09@wwu.de

Anne Remke
Westfälische Wilhelms-Universität
Münster, Germany
anne.remke@wwu.de

***Abstract*—Hybrid Petri nets with general transitions (HPnGs) provide a formalism for modeling safety-critical systems and evaluating their dependability with means of model checking. HPnGs form a restricted subclass of Stochastic Hybrid Automata and allow discrete, continuous and stochastic variables. Previously, discrete-event simulation and Statistical Model Checking (SMC) have been used to overcome the restrictions of existing analytical approaches, e.g., to a limited number of random variables. Also when simulating, the evolution of continuous variables has been restricted to piecewise-linear trajectories, where derivatives do not change between two events. Here, we extend the modeling formalism, the simulation and SMC approach to variables with a non-linear continuous evolution. The core idea of this extension lies in transforming the input system into a so-called second-order quantized state system. A case study on the Kinetic Battery Model validates our approach by comparing results to those obtained by Matlab.**

I. Introduction

When dealing with safety-critical systems, for example power distribution networks, one often has to consider both discrete and continuous features. Those systems are then called hybrid systems. In addition, many real systems also exhibit random behavior, for example within their failure and repair processes. The modeling formalism of *hybrid Petri nets with general transitions (HPnGs)* [1] is well-suited for modeling hybrid systems with stochastic variables and evaluating their dependability. Previous analytical approaches [1]–[4] for HPnGs have been restricted to models with piecewise-linear continuous evolution and their analysis was restricted to a maximum of one or two random variables. However, in reality the continuous system evolution is often non-linear and expressed as differential equations. As an example, the state-of-charge of a battery evolves non-linearly, even when charged or discharged with a constant load [5].

This work aims at expanding a previously presented approach towards non-linear continuous variables. [6] presented *discrete-event simulation (DES)* and time-bounded *Statistical Model Checking (SMC)* of HPnGs for continuous variables with a piecewise-linear evolution. It provides three major contributions: Firstly, the syntax and semantic of HPnGs are extended to include dynamic continuous transitions, which allows modeling non-linear continuous evolution. Secondly, the existing DES approach for HPnGs is extended to cope with non-linear trajectories, which is achieved by transforming the input system into a so-called *second-order quantized state system (QSS2)* [7]. Third, the SMC approach is extended according to the error introduced by the transformation to a QSS2. Within QSS2, the continuous state variables are approximated as piecewise-linear trajectories, whose derivatives are updated whenever the function value deviates too far from the original value. On the one hand, this approach requires the estimation of the trajectory of the original state variable and on the other hand the prediction of the point in time when the next recalculation should take place. Furthermore, the deviation of the approximated values from the original values has to be considered, when model checking properties that depend on the values of the continuous variables.

Based on the challenges identified above, we present an approach to verify several (safety) properties on systems with discrete, non-linear continuous and a large number of stochastic variables, which follow arbitrary distributions. This work is restricted to systems that can be represented by systems of ordinary differential equations (ODEs), which only contain time as independent variable. An extension for other classes of differential equations might be feasible in the future. Our approach has been implemented in the Java tool *HYPEG* [8], which is available on Github[1].

A case study featuring the non-linear battery model KiBaM [9] is used to validate our results by comparison to a hard-coded analysis implemented in *Matlab*. In [10], the KiBaM has been compared to the so-called diffusion model when computing lifetimes of a pocket computer battery. The KiBaM model has previously been used for modeling the battery of a low-earth-orbit satellite, named GOMX-3, to solve the task scheduling problem for this satellite [11]. While the first approach hard-coded the equations of the KiBaM model, the second one uses a discretization approach to analyze the stochastic KiBaM model, which is induced by a Markov task process.

Being able to model such battery behavior in terms of a hybrid Petri net will allow more realistic models of smart homes, e.g., adapting the smart house models in [10], [12], [13] and the electric vehicle models in [14], [15].

A. Related Work

Hybrid Petri nets with general transitions (HPnGs) [1] form a restricted subclass of *Stochastic Hybrid Automata (SHAs)*

[1]https://github.com/jannikhuels/libhpng

[16], which additionally allow non-deterministic behavior. In HPnGs, non-determinism is resolved by assigning priorities and weights to the transitions.

Other common subclasses of SHA are amongst others *Stochastic Timed Automata (STAs)* and *Hybrid Automata (HAs)*. The former (e.g., c.f. [17]) form an extension of Timed Automata, which are equipped with probability measures, such that they are able to model stochastic variables but exclude continuous variables. The latter (c.f. [18]) combine discrete and continuous (i.e., hybrid) features but exclude random behavior. An extension of HAs is given by *Linear Stochastic Hybrid Automata (LSHA)* [19], which are able to model systems that can be represented by Poisson processes and stochastic differential equations (SDEs).

This work extends HPnGs to models whose continuous behavior is described by systems of ordinary differential equations (ODEs). A simulation of such behavior always requires either an approximation of time or state. Next to various algorithms for ODE solving, which are based on time discretization, Kofman and Junco introduced so-called *Quantized State Systems (QSS)* [20], which are based on the work of Zeigler and Lee on quantized systems [21]. Kofman and Junco add a quantization function equipped with hysteresis to the piecewise-linear behavior of state variables and thus convert it into piecewise-constant behavior. The resulting QSS approximates the continuous behavior of the original system and can be simulated by discrete-event simulation. In [7], Kofman introduced *second-order quantized state systems (QSS2)*, in which state variables with piecewise-parabolic trajectories (i.e., piecewise-linear derivatives) are converted to variables with piecewise-linear trajectories, again by an added quantization function. He shows that in case of a linear time-invariant system, the error resulting from the QSS2 transformation can be bounded by a linear function of the quantization.

Kofman presents a *Discrete Event System Specification (DEVS)* for the QSS2, which uses a quadratic Taylor polynomial approximation of the input trajectories. He proves that QSS2, which result from a transformation of linear time-invariant systems, can be modeled exactly by this DEVS. However, he points out that QSS2 resulting from general non-linear systems do not have an exact DEVS representation, but can be represented by the given approximated DEVS model. Hence, by adding a quantizer, any input system with non-linear continuous variables can be transformed into a QSS2. For linear time-invariant systems, the resulting QSS2 can be exactly simulated according to the DEVS model and for non-linear general systems, the resulting QSS2 can be approximated (using Taylor polynomial approximation) and thus be simulated as well by the DEVS. In [22], the QSS2 method is extended for differential algebraic equation (DAE) systems. Since his original method was restricted to continuous systems, Kofman has extended his approach to hybrid systems in [23] by taking both the continuous and discrete parts of a system into account when determining the next event within discrete-event simulation.

In this work, we present a discrete-event simulation approach, which adds quantizers to the non-linear continuous variables of an HPnG, resulting in a QSS2. Hence, we are able to overcome the restriction to piecewise-linearity and hence, to approximately simulate non-linear continuous trajectories.

Statistical Model Checking (SMC) [24] has previously been applied to different models with stochastic variables, for example to Markov chains in [25] and STAs in [26]. Another Petri net formalism is given by *Fluid Stochastic Petri Nets (FSPNs)* [27], for which DES has been investigated in [28]. The continuous behavior of FSPNs is also described by differential equation. However, firing times in these models are restricted to exponential distributions. Uppaal SMC [29], [30] is able to do Statistical Model Checking for stochastic hybrid models, where the continuous behaviors follows ODEs. It is also able to model random timing following arbitrary distributions. However, modeling and implementation of such delays is cumbersome. A different approach for bounded model checking is proposed in [31]. Furthermore, [32] introduces a Statistical Model Checking approach that combines rare-event simulation with scheduler sampling for nondeterministic models.

Within this work, we use an HPnG model of the *Kinetic Battery Model (KiBaM)* by Manwell and McGowan [9] as running example, which later also serves as case study. When modeling the charging and discharging behavior of batteries, a linear approximation of the state-of-charge is often not sufficient. The KiBaM provides a popular model for batteries. Given a constant load, the stored energy of a battery is divided into two wells, whose derivatives are described by two coupled differential equations. In [33], the definition of the model is extended to cover also bounded capacity and random charging and discharging.

B. Outline

Section II gives an overview of the model formalism of hybrid Petri nets with general transitions and presents the running example. Next, the extended discrete-event simulation approach is discussed in Section III. Section IV then explains how Statistical Model Checking is applied to systems with non-linear continuous behavior. The case study is presented in Section V and finally the paper is concluded in Section VI.

II. Hybrid Petri nets with multiple general transitions

Hybrid Petri nets with general transitions (HPnGs) [1] provide a modeling formalism which describes systems with discrete, continuous and stochastic components and form a subclass of stochastic hybrid systems. Previous model definitions have been restricted to piecewise-linear continuous behavior. To allow non-linear behavior, we add a new dynamic continuous transition to the model definition in Section II-A, followed by an extended semantic in Section II-B, which explains the evolution of continuous variables. Section II-C gives a overview about the definition of states and events within an HPnG. Section II-D presents an HPnG model that will serve as running example throughout this work.

A. Model definition

According to [1] and [6], a hybrid Petri net with general transitions is a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M}_0, \Phi)$ with Φ a tuple $(\Phi_b^{\mathcal{P}}, \Phi_w^{\mathcal{T}}, \Phi_p^{\mathcal{T}}, \Phi_d^{\mathcal{T}}, \Phi_{st}^{\mathcal{T}}, \Phi_{dyn\_1}^{\mathcal{T}}, \Phi_{dyn\_2}^{\mathcal{T}}, \Phi_g^{\mathcal{T}}, \Phi_w^{\mathcal{A}}, \Phi_s^{\mathcal{A}}, \Phi_p^{\mathcal{A}})$ of parameter functions. $\mathcal{P} = \mathcal{P}^d \cup \mathcal{P}^c$ is the finite set of discrete and continuous places. A discrete place $P_i \in \mathcal{P}^d$ holds a discrete marking of $m_i \in \mathbb{N}_0$ tokens and a continuous place $P_j \in \mathcal{P}^c$ holds a continuous marking described by a fluid level $x_j \in \mathbb{R}_0^+$.The initial marking $\mathbf{M}_0 = (\mathbf{m}_0, \mathbf{x}_0)$ is determined by the initial number of tokens $\mathbf{m}_0$ and fluid levels $\mathbf{x}_0$ of all places. For every continuous place, $\Phi_b^{\mathcal{P}} : \mathcal{P}^c \to \mathbb{R}^+ \cup \infty$ defines an upper boundary, whereas the lower boundary is always zero.

The finite set of transitions $\mathcal{T} = \mathcal{T}^I \cup \mathcal{T}^D \cup \mathcal{T}^G \cup \mathcal{T}^C$ consists of immediate, deterministic, general and continuous transitions. An immediate transition $T_i \in \mathcal{T}^I$ fires as soon as it is *enabled*, whereas a deterministic transition $T_d \in \mathcal{T}^D$ fires after being enabled for a deterministic period of time, which is assigned by the function $\Phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mathbb{R}^+$. For a general transition $T_g \in \mathcal{T}^G$, the firing time is a random variable, which is distributed according to a cumulative continuous probability distribution function $f_{T_g}(t)$, assigned by $\Phi_g^{\mathcal{T}} : \mathcal{T}^G \to (f : \mathbb{R}^+ \to [0, 1])$.

Given $\mathcal{T}^S = \mathcal{T} \setminus \mathcal{T}^C$, the function $\Phi_w^{\mathcal{T}} : \mathcal{T}^S \to \mathbb{R}^+$ adds a weight to any discrete (i.e., non-continuous) transition and $\Phi_p^{\mathcal{T}} : \mathcal{T}^S \to \mathbb{N}_0$ assigns a priority. A continuous transition $T_c \in \mathcal{T}^C$ has a continuous inflow and outflow of fluid if it is enabled. We divide the set $\mathcal{T}^C$ into two disjoint sets: the set of static transitions $\mathcal{T}^{St}$ and the set of dynamic transitions $\mathcal{T}^{Dyn}$. The function $\Phi_{st}^{\mathcal{T}} : \mathcal{T}^{St} \to \mathbb{R}^+$ specifies a constant nominal flow rate for every static transition. Dynamic continuous transitions have a flow rate that depends on the continuous marking, resulting in non-linear behavior. Since Ghasemieh e.a. also introduced dynamic transitions in [34], whose rates depend on the actual rates of other static continuous transitions, we additionally include this approach as a second part of the flow rate. The nominal flow rate is defined as follows:

Definition 1: For dynamic continuous transitions, the nominal flow rate is divided into two parts:

1) $\Phi_{dyn\_1}^{\mathcal{T}} : \mathcal{T}^{Dyn} \to (d : X \to \mathbb{R}_0^+)$ with $X = \{\mathbf{x} = (x_1, ..., x_{|\mathcal{P}^c|}) \in (\mathbb{R}_0^+)^{|\mathcal{P}^c|} \mid \forall i \in \mathbb{N}.1 \leq i \leq |\mathcal{P}^c| : P_i \in \mathcal{P}^c \wedge 0 \leq x_i \leq \Phi_b^{\mathcal{P}}(P_i)\}$ assigns a function $d_{T_{dyn}}(\mathbf{x})$ for to each dynamic transition $T_{dyn} \in \mathcal{T}^{Dyn}$, which specifies the dependence of its nominal flow rate on the continuous marking $\mathbf{x}$.
2) $\Phi_{dyn\_2}^{\mathcal{T}} : \mathcal{T}^{Dyn} \to (f : \mathbb{R}^{|\mathcal{T}^{St}|} \to \mathbb{R}_0^+)$ assigns a function $f_{T_{dyn}}(r)$ for $r \in \mathbb{R}^{|\mathcal{T}^{St}|}$ to each dynamic transition $T_{dyn} \in \mathcal{T}^{Dyn}$, which determines the dependence of its nominal flow rate on the actual flow rates of the static continuous transitions.

The nominal flow rate of $T_{dyn} \in \mathcal{T}^{Dyn}$ is given by the sum $d_{T_{dyn}}(\mathbf{x}) + f_{T_{dyn}}(r)$.

The finite set $\mathcal{A} = \mathcal{A}^d \cup \mathcal{A}^f \cup \mathcal{A}^t \cup \mathcal{A}^h$ contains discrete, continuous, test and inhibitor arcs. Arcs connect places with transitions and at most one arc of each type is allowed between a pair of a place and a transition. The function $\Phi_w^{\mathcal{A}} : \mathcal{A} \to \mathbb{R}_0^+$ assigns a weight to each arc. Any discrete arc $A_k \in \mathcal{A}^d \subseteq ((\mathcal{P}^d \times \mathcal{T}^S) \cup (\mathcal{T}^S \times \mathcal{P}^d))$ connects a discrete place to a non-continuous transition and any continuous arc $A_l \in \mathcal{A}^f \subseteq ((\mathcal{P}^c \times \mathcal{T}^C) \cup (\mathcal{T}^C \times \mathcal{P}^c))$ connects a continuous place to a continuous transition. Hence, any discrete or continuous arc is whether an input or an output arc for its connected place and their firing rates are determined by their weights. For fluid sharing purposes, continuous arcs have shares assigned by $\Phi_s^{\mathcal{A}} : \mathcal{A}^f \to \mathbb{R}^+$ and priorities assigned by $\Phi_p^{\mathcal{A}} : \mathcal{A}^f \to \mathbb{N}_0$.

A test arc $A_q \in \mathcal{A}^t \subseteq (\mathcal{P} \times \mathcal{T})$ respectively an inhibitor arc $A_r \in \mathcal{A}^h \subseteq (\mathcal{P} \times \mathcal{T})$ connects any place to any transition. For test arcs, the marking of the place has to fulfill a condition given by the arc weight to enable the connected transition. For inhibitor arcs, the transition is enabled if the condition is not fulfilled. Test and inhibitor arcs are also called guard arcs.

The places of an HPnG are represented by circles and the transitions are represented by bars. A graphical representation of the places, transitions and arcs is given by Figure 1 and the notations used in this work are summarized by Table I.

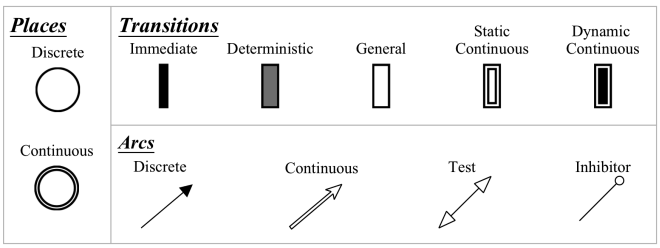

Fig. 1. Graphical representation of HPnGs components.

B. Behavior specification and continuous evolution

We adopt concession and enabling rules and the definition of the evolution of clocks and enabling times from [1]. Furthermore, we adopt the approach for conflict resolution from [1] for immediate and deterministic transitions and from [6] for general transitions. The behavior of general transitions and the definition of events in an HPnG are both also adopted from [6]. Since we added dynamic continuous transitions in our definition in Section II-A, we slightly change the definition of the evolution of continuous variables in the following.

Continuous transitions with concession continuously transport fluid from one continuous place to another via fluid arcs. Recall that every static transition $T_i \in \mathcal{T}^{St}$ has an assigned nominal flow rate, given by $\Phi_{st}^{\mathcal{T}}(T_i)$. Considering a certain point in time t, the flow rate of a dynamic transition $T_j \in \mathcal{T}^{Dyn}$ is determined by $d_{T_j}(\mathbf{x}(t)) + f_{T_j}(r)$. Recall that $\mathbf{x}(t)$ is the continuous marking at time point t and d_{T_j} the function assigned by $\Phi_{dyn}^{\mathcal{T}}(T_j)$.

Let $\theta(T_c, t)$ denote the actual flow rate of a transition $T_j \in \mathcal{T}^C = \mathcal{T}^{St} \cup \mathcal{T}^{Dyn}$ at some point in time t. If no continuous place is at one of its boundaries, i.e., $\forall P_i \in \mathcal{P}^c : 0 < x_i(t) < \Phi_b^{\mathcal{P}}(P_i)$ for $x_i(t) \in \mathbf{x}(t)$, then the continuous transitions continuously fire at their nominal rates. If one of

TABLE I
NOTATIONS

$\mathcal{P} = \mathcal{P}^d \cup \mathcal{P}^c$	Set of discrete and continuous places
$\mathcal{T} = \mathcal{T}^I \cup \mathcal{T}^D \cup \mathcal{T}^G \cup \mathcal{T}^C$	Set of immediate, deterministic, general and continuous transitions
$\mathcal{T}^C = \mathcal{T}^{St} \cup \mathcal{T}^{Dyn}$	Set of static continuous and dynamic continuous transitions
$\mathcal{T}^S = \mathcal{T} \setminus \mathcal{T}^C$	Set of discrete transitions
$\mathcal{A} = \mathcal{A}^d \cup \mathcal{A}^f \cup \mathcal{A}^t \cup \mathcal{A}^h$	Set of discrete, continuous, test and inhibitor arcs
$\mathbf{M}_0 = (\mathbf{m}_0, \mathbf{x}_0)$	Initial marking
$\Phi_b^{\mathcal{P}} : \mathcal{P}^c \to \mathbb{R}^+ \cup \infty$	Upper boundary of continuous places
$\Phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mathbb{R}^+$	Firing time for deterministic transitions
$\Phi_g^{\mathcal{T}} : \mathcal{T}^G \to (f : \mathbb{R}^+ \to [0,1])$	Function assigning firing probability distribution for general transitions
$\Phi_w^{\mathcal{T}} : \mathcal{T}^S \to \mathbb{R}^+$	Weight of discrete transitions
$\Phi_p^{\mathcal{T}} : \mathcal{T}^S \to \mathbb{N}_0$	Priority of discrete transitions
$\Phi_{st}^{\mathcal{T}} : \mathcal{T}^{St} \to \mathbb{R}^+$	Nominal flow rate for static continuous transitions
$\Phi_{dyn\_1}^{\mathcal{T}} : \mathcal{T}^{Dyn} \to (d : X \to \mathbb{R}_0^+)$	Function assigning 1st part of nominal flow rate for dynamic transitions
$\Phi_{dyn\_2}^{\mathcal{T}} : \mathcal{T}^{Dyn} \to (f : \mathbb{R}^{\lvert\mathcal{T}^{St}\rvert} \to \mathbb{R}_0^+)$	Function assigning 2nd part of nominal flow rate for dynamic transitions
$\Phi_w^{\mathcal{A}} : \mathcal{A} \to \mathbb{R}_0^+$	Weight function for arcs
$\Phi_s^{\mathcal{A}} : \mathcal{A}^f \to \mathbb{R}^+$	Share function for continuous arcs
$\Phi_p^{\mathcal{A}} : \mathcal{A}^f \to \mathbb{N}_0$	Priority function for continuous arcs
$\theta : \mathcal{T}^C \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$	Actual flow rate of cont. transitions
$\alpha : \mathcal{P}^C \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$	Available flow rate at cont. places
$\omega_l : \mathcal{P}^C \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$	Required fluid function at continuous places with priority l
$\mathcal{I}^c : \mathcal{P}^C \to P(\mathcal{T}^C)$	Input bag of continuous places
$\mathcal{O}^c : \mathcal{P}^C \to P(\mathcal{T}^C)$	Output bag of continuous places
$d_i : \mathbb{R}_0^+ \to \mathbb{R}$	Drift at continuous place $P_i \in \mathcal{P}^C$

the continuous places is empty or full, the flow rate is reduced to prevent overflow or underflow. Hence, the actual flow rate $\theta(T_j, t)$ is given by:

$$\theta(T_j,t) \leq \begin{cases} \Phi_{st}^{\mathcal{T}}(T_j) & \text{if } T_j \in \mathcal{T}^{St}, \\ d_{T_j}(\mathbf{x}(t)) + f_{T_j}(r) & \text{if } T_j \in \mathcal{T}^{Dyn}. \end{cases} \tag{1}$$

Based on this definition of the actual flow rate, the rate adaption procedure from [1] can be analogously defined for non-linear continuous behavior. The only differences lie in the facts that (i) the available flow rate $\alpha(P_i, t)$ at a place $P_i \in \mathcal{P}^c$ is a time-dependent function, since it is related to the re-defined actual flow rate $\theta(T_j, t)$ and (ii) the amount of required fluid $\omega_l(P_i, t)$ for a priority l in P_i is also time-dependent as it is related to the nominal flow rate, which is time-dependent for dynamic continuous transitions. If a continuous transition has no concession, the actual flow rate is zero. As in [1], we define for any continuous place $P_i \in \mathcal{P}^c$ an input bag $\mathcal{I}^c(P_i) = \{T_j \in \mathcal{T}^C | \langle T_j, P_i \rangle \in \mathcal{A}^f\}$ and output bag $\mathcal{O}^c(P_i) = \{T_j \in \mathcal{T}^C | \langle P_i, T_j \rangle \in \mathcal{A}^f\}$. The drift $d_i(t)$ of P_i is then determined by:

$$d_i(t) = \sum_{T_j \in \mathcal{I}^c(P_i)} \theta(T_j,t) \cdot \Phi_w^{\mathcal{A}}(\langle T_j, P_i \rangle) - \sum_{T_j \in \mathcal{O}^c(P_i)} \theta(T_j,t) \cdot \Phi_w^{\mathcal{A}}(\langle T_j, P_i \rangle). \tag{2}$$

C. Definition of states and events

We define the state of a hybrid Petri net with general transitions according to [6]. A state is given by a tuple $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g}, \mathbf{e})$. The vector $\mathbf{m}$ describes the discrete marking and $\mathbf{x}$ the continuous marking. The clock vector $\mathbf{c}$ holds for any deterministic transition the time that it has been enabled since the previous firing and the vector $\mathbf{d}$, describes the *drift* of each continuous place, i.e., the current change of fluid level per time unit, which corresponds to the difference of incoming and outgoing flow. The enabling time for the general transitions is collected in $\mathbf{g}$ and for all kinds of transitions, the enabling status is given by $\mathbf{e}$. Note that the drift and enabling status are both determined uniquely by $\mathbf{m}$ and $\mathbf{x}$ and $\Phi_w^{\mathcal{A}}$, but included for the sake of simplification. The initial state is denoted by $\Gamma_0 \in S$ with $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}, \mathbf{d}_0, \mathbf{0}, \mathbf{e}_0)$.

We adopt the definition of events also from [6]. Recapped, an event denotes the change from one state $\Gamma_i = (\mathbf{m}_i, \mathbf{x}_i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{g}_i, \mathbf{e}_i)$ to another state $\Gamma_{i+1} = (\mathbf{m}_{i+1}, \mathbf{x}_{i+1}, \mathbf{c}_{i+1}, \mathbf{d}_{i+1}, \mathbf{g}_{i+1}, \mathbf{e}_{i+1})$, such that either (i) a non-continuous transition fires, such that $\mathbf{m}_i \neq \mathbf{m}_{i+1}$ or (ii) a continuous place reaches one of its boundaries, such that $\mathbf{d}_i \neq \mathbf{d}_{i+1}$ or (iii) a guard arc condition is fulfilled or stops being fulfilled, sucht that $\mathbf{e}_i \neq \mathbf{e}_{i+1}$.

D. An example HPnG modeling the Kinetic Battery Model

We will present an HPnG, which models the Kinetic Battery Model (KiBaM) [9] that serves as a running example throughout the paper. As illustrated in Figure 2, the KiBaM models the state of charge of a battery as two wells: the available charge $a(t)$ and the bound charge $b(t)$. A pipe connects both wells with a diffusion rate p in both directions. The proportional width of the well that contains the available charge, with respect to the bound charge, is denoted by c. Hence, the well that contains the bound charge has proportional width $1 - c$.

Only the available charge is exposed to an external load and can be consumed immediately. The bound charge is converted into available change over time, which is called the *recovery effect*. The following system of differential equations defines the derivatives of the continuous variables a and b, where the battery is charged if $I(t) < 0$ and discharged when $I(t) > 0$:

$$\begin{aligned} \frac{\delta a(t)}{\delta t} &= p \cdot \left(\frac{b(t)}{1-c} - \frac{a(t)}{c} \right) - I(t) && \text{for } a(0) = a_0, \\ \frac{\delta b(t)}{\delta t} &= p \cdot \left(\frac{a(t)}{c} - \frac{b(t)}{1-c} \right) && \text{for } b(0) = b_0. \end{aligned} \tag{3}$$

How to deal with bounded capacities and (piecewise constant) random charging and discharging rates within the KiBaM, is explained in [33].

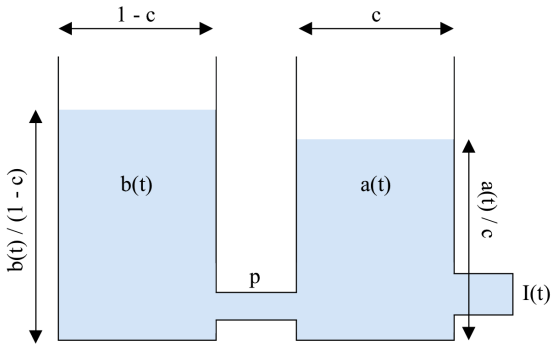

Fig. 2. The two-well model of the KiBaM with the available charge (right) and the bound charge (left) (c.f. Figure 2 [33]).

The HPnG in Figure 3 models $a(t)$ and $b(t)$ as continuous places **a** and **b**, respectively. The flow between a and b is modeled by the two dynamic continuous transitions **fill_a** and **fill_b**. The diffusion rate p is set to 0.01 and we set $c = 0.5$, i.e., the wells are equally large. Both wells have a respective maximum capacity $a_{\max}$ and $b_{\max}$. Our example considers a capacity of 5000 mAh for each well and initial charges of 1000 mAh. When charging and the available charge is almost at its maximum capacity, the rate $I(t)$ needs to be adapted as follows: if $a > a_{max} - |I(t)|$, then $I(t) := (-1) \cdot (a_{max} - a)$. Similarly, when discharging, both values a and b cannot decrease below zero. Hence, $I(t)$ needs to be adapted as follows: if $I(t) > a$, then $I(t) := a$.

The static continuous transitions **charge_1**, **charge_2** and **discharge** model the charging and discharging behavior, i.e., charging with rates of 1000 mA and 400 mA and discharging with a rate of 500 mA. At each point in time, only one of these three transitions is enabled, depending (via guard arcs) on the marking of the discrete places **enable_charge_1**, **enable_charge_2**, **enable_discharge**. These discrete places are connected to each other via three deterministic transitions, namely **reduce_rate**, **plug_out** and **plug_in**, which forward a single token after $2, 8$ or 6 hours. The model also includes a random outage of the charging / discharging process, modeled by the general transition **power_off**. We will consider different distributions for this random variable. When the general transition fires, a token from the discrete place **power** is removed, which leads to enabling and firing one of the immediate transitions **out_1**, **out_2** and **out_3** (connected to **power** via inhibitor arcs). The token is then transported to the place **enable_outage**, so that the previous load is stopped and a new discharging process with rate 200 mA is enabled, modeled by the static continuous transition **outage**.

III. Discrete-event simulation of HPnGs with non-linear continuous evolution

Discrete-event simulation (DES) and Statistical Model Checking (SMC) for HPnGs with linear continuous behavior has been proposed in [6], considering only piecewise-linear behavior of continuous variables, whose derivatives only change at events. Any DES approach requires the determination of all potential next events and the point in time of their occurrence.

For example in the KiBaM model from Section II-D, we may have to determine when $a(t)$ reaches its maximum capacity. At any point in time within the simulation, we would need to determine the respective next point in time when the fluid level of the place **a** meets its upper boundary. For a constant drift (i.e., derivative), the occurrence time of the next event can simply be precomputed. However, non-linear evolutions are described by differential equations, which in general do not have closed solutions. The KiBaM expresses a system of linear ODEs, solvable for time by using the *Lambert W* function [35]. However, this function does not have a closed form, but can be approximated numerically. One possible solution to cope with non-linear trajectories is to transform the non-linear variables into piecewise-linear ones, by using so-called *quantization*.

In the piecewise-linear approach, the derivative of the continuous variables only changes at events. However, when allowing non-linear evolution, the drifts of the continuous places change also in between events. By quantizing the continuous variables, they are treated as piecewise-linear variables, while constantly checking that the error between the original and the quantized variable does not exceed a predefined threshold. This results in a second-order quantized state systems (QSS2) [7] with piecewise-linear evolution, which can be dealt with like in our previous DES approach. However, we have to ensure that the quantized variable does not differ too much from the original continuous variable and its value needs to be reset, whenever the difference between the two values meets some threshold, denoted as Δq. Such a reset is called an *internal transition* of a continuous place. The details on this approach are discussed in the following.

Section III-A presents the core idea of the extended DES approach and presents a formal definition of the resulting QSS2. Details on how to determine the time to the respective next internal transitions are discussed in Section III-B and Section III-C explains how the rate adaption technique has been extended.

A. DES of HPnGs using transformation into QSS2

We presented an approach for the discrete-event simulation of HPnGs with a piecewise-linear evolution of continuous variables in [6]. Within a simulation experiment, several simulation runs are executed, which simulate single paths of a given model. For each simulation run, the model is set to its initial state and the current simulation time *currentTime* is initialized to zero. The firing time for each general transition is sampled from the corresponding distribution at the beginning of every simulation run and re-sampled after each firing. Depending on the sampling policy [6], variables are also re-sampled in case a general transition is disabled and then enabled before a firing takes place.

After the initialization, the following loop is executed:

```
while (currentTime <= maxTime)
  currentTime = getAndCompleteNextEvent(currentTime);
```

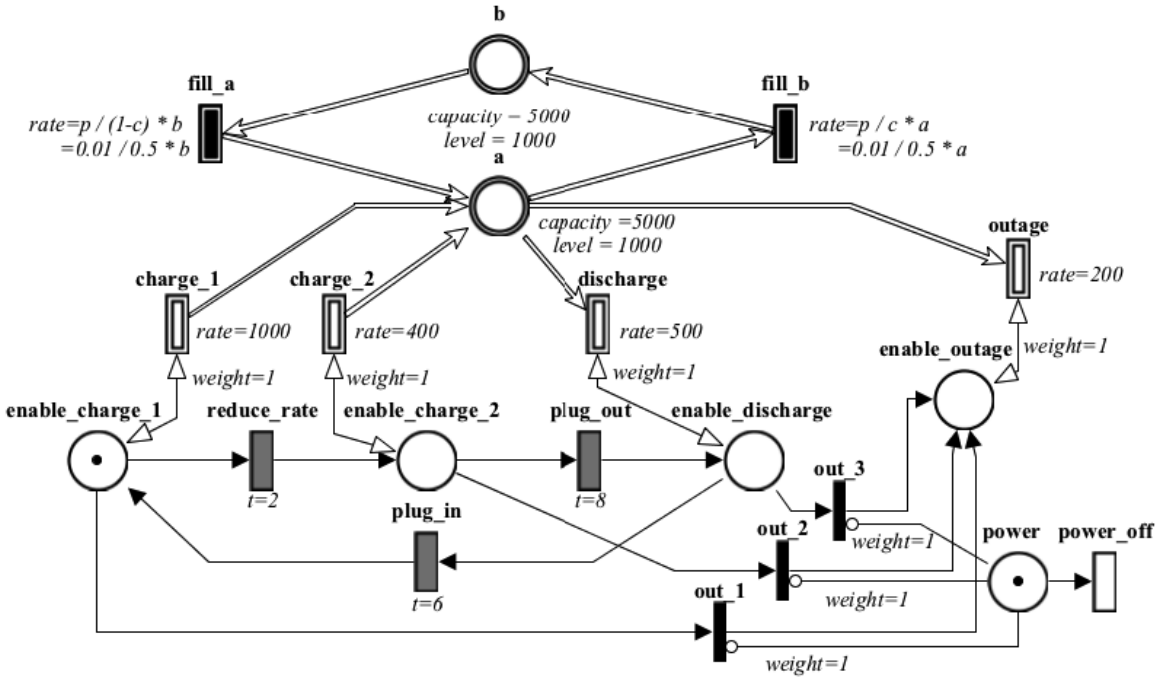

Fig. 3. HPnG model of the Kinetic Battery Model with specific loads and random outage.

The function *getAndCompleteNextEvent()* is repeatedly called for any event until the maximum time *maxTime* is reached. Since this function is called within the given **while**-loop, the events are always handled one-by-one, such that the occurrence of an event can be dependent on previous events.

Algorithm 1 presents pseudo code for this function and is based on Algorithm 1 in [6]. At first, the new event is initialized with its occurence time set to *maxTime* (Line 1). If any immediate transition is enabled, the next event is already found and updated (Lines 2–4). Otherwise, the algorithm iterates through deterministic and general transitions, continuous places and guard arcs which are connected to continuous places (Lines 5–18), since these components can cause further events (recall from Section II-C). We extended the original algorithm by Lines 15–17 to include the *internal transitions* of continuous places as a new type of event. When the respective next event has been determined, the continuous marking is updated to the point in time of the new event (Line 19). Then, the event is executed, which results in an update of the state of the model (Line 20).

Following the work of [7] and [23], we consider the state equation system given by:

$$\dot{x}(t) = f[x(t), u(t)], \tag{4}$$

where $x(t) \in \mathbb{R}^n$ is the state vector and $u(t) \in \mathbb{R}^m$ is an input vector. In our case $x(t)$ represents the fluid levels of the continuous places and $u(t)$ represents their drift.

Definition 2: A QSS2 system, associated to a state equation system $\dot{x}(t) = f[x(t), u(t)]$, for a pre-chosen vector $\Delta q \in \mathbb{R}^n$ is defined as:

$$\dot{x}(t) = f[q(t), u(t)]. \tag{5}$$

Algorithm 1 getAndCompleteNextEvent(currentTime)

```
1: event = new SimulationEvent(maxTime)
2: if (any immediate transition enabled) then
3:     event.update(currentTime)
4: end if
5: if (no immediate transition event found yet) then
6:     if (any deterministic and general transition enabled and firing before
       event.time) then
7:         event.update(newTime)
8:     end if
9:     if (any guard arc condition changed before event.time) then
10:        event.update(newTime)
11:    end if
12:    if (any continuous place boundary reached before event.time) then
13:        event.update(newTime)
14:    end if
15:    if (any internal transition due before event.time) then
16:        event.update(newTime)
17:    end if
18: end if
19: advanceMarking(event.time)
20: completeEvent()
21: return event.time
```

The *first-order quantizer* $q(t)$ is defined as:

$$q(t) = \begin{cases} x(t) & \text{if } t = t_0 \vee t = t_{j+1}, \\ q(t_j) + m_j(t - t_j) & \text{otherwise,} \end{cases} \tag{6}$$

with internal transitions taking place at time points t_j, with $j \geq 0$, where the next time point is given by:

$$t_{j+1} = \min(t | t > t_j \wedge |x(t_j) + m_j(t - t_j) - x(t)| = \Delta q), \tag{7}$$

with $m_0 = 0$ and $m_j = \dot{x}(t_j)$.

In the resulting QSS2, the derivatives of the quantized variables only change at internal transitions, which is required

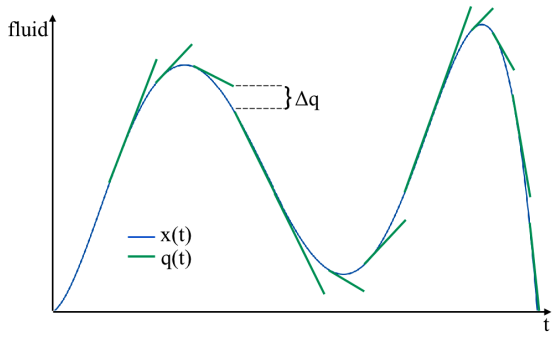

Fig. 4. Example trajectories of the state variable $x(t)$ and quantized variable $q(t)$ in a QSS2 system [7].

when $|q(t) - x(t)|$ equals Δq. Figure 4 illustrates example trajectories for the original variable $x(t)$ and the quantizer $q(t)$. It can be seen that the quantizer is reset to the value of $x(t)$, if the difference between both values becomes too large, i.e., equal to Δq. This would require solving the ODE $x(t)$ at time point t, which is however approximated using a Taylor polynomial approximation.

When determining the next event within the simulation, the internal transitions of the continuous places are treated like every other event. If the time to any next internal transition is less than the time to the next event, the simulation steps to the point in time of the internal transition. The internal transition is executed by updating both the value of $x(t)$ and setting $q(t)$ to $x(t)$. Afterwards, the time to the next internal transition is re-computed.

But how can we update the value $x(t)$ without solving differential equations? In [7], a Discrete Event System Specification (DEVS) model of a first-order quantized integrator is given, which provides a basis for our DES approach. In this approach $x(t)$ is updated using its previous derivative u and the previous derivative of u, named m_u. This results in a quadratic Taylor polynomial approximation given by Equation 8:

$$x(t) \approx x(t_j) + u(t_j) \cdot (t - t_j) + \frac{m_u(t_j)}{2} \cdot (t - t_j)^2, \quad (8)$$

with t_j the point in time of the previous internal transition and $u(t_j) = \dot{x}(t_j)$ and $m_u(t_j) = \dot{u}(t_j)$.

After discussing how the continuous variables are updated in the simulation algorithm, we now consider how the current fluid level of a place is taken into account e.g. for checking the enabling conditions of inhibitor and test arcs. As $q(t)$ can both under- and over-approximate the actual value $x(t)$, both need to be taken into account, to find the exact point in time t, when the validity of a condition changes. For $q(t)$ this time point is determined by equating its linear function with the boundary value of the condition and then solving for time. For $x(t)$, its quadratic Taylor polynomial approximation is used in the equation and the least positive solution time is taken.

B. Determining the time until the next internal transition

At the start of the simulation, the time until the next internal transition takes place is determined for every continuous place. In the DEVS model, for some time point t_j at which an internal transition takes place, the time to the respective next internal transition σ_1 is computed according to [23] (c.f. Equation 6):

$$\sigma_1 = \begin{cases} \sqrt{\frac{2\Delta q}{m_u(t_j)}} & \text{if } m_u(t_j) \neq 0, \\ \infty & \text{otherwise.} \end{cases} \quad (9)$$

Whenever an event (which is not an internal transition) is completed, the time to the next internal transition of all continuous places is recalculated, since the event might have caused a change of u and m_u. However, note that the value for $x(t)$ is not updated and that $q(t)$ is linearly updated instead of being set to the value of $x(t)$. This is sufficient because $|q(t) - x(t)| \leq \Delta q$ (otherwise, an internal transition would have been executed before the event). The time to the next internal transition σ_2 for some t_k is then given by the least positive solution of Equation 10 (c.f. Equation 7 [23]):

$$|x(t_j) + u(t_j) \cdot e + \frac{m_u(t_j)}{2} e^2 + u(t_k) \cdot \sigma_2 + \frac{m_u(t_k)}{2} \cdot \sigma_2^2 - (q(t_j) + m_q(t_j)(e + \sigma_2))| = \Delta q, \quad (10)$$

where t_j the time point of the previous internal transition and $e = t_k - t_j$ the amount of time, which has passed since the previous internal transition.

If a boundary of a place is reached, two special cases apply: Whenever the upper boundary of the place has been reached, the next internal transition is executed when the outflow equals the inflow. This can only happen if the derivative of the outflow is greater than the derivative of the inflow. In case the place is empty, the same applies only if the derivative of the inflow is greater than the derivative of the outflow. Otherwise, the place will stay at its boundary. The details on how the rates are adapted are discussed in the next section.

C. Time-dependent rate adaption

As mentioned in Section II-B, the rate adaption procedure for non-linear continuous behavior is similar to the one in [1], only extended by some time-dependent factors. Rate adaption is required, whenever a continuous place reaches its upper or lower boundary and the rates of continuous transitions have to be reduced to match the inflow or respectively the outflow. In addition to the previous algorithm, we now consider (constant) derivatives of the available and required fluids, resulting in a time-dependent fluid distribution. Note that the relationship between the amount of required fluid for each priority class of transitions and the amount of available fluid changes over time.

For the lowest priority of those transitions, for which the required fluid can still be served completely, the available fluid might stop being sufficient to serve these transitions. The point in time t_a at which this will happen can be computed by

the intersection of both linear functions of the required and available fluid. Note that t_a might be infinite if the available fluid does not decrease or decreases more slowly than the required fluid. Analogously, for the highest priority of those transitions which could not be completely served, we need to determine the point in time t_b at which the available fluid starts being sufficient for serving these transitions. If one of these two points t_a and t_b takes place before the next internal transition of the considered continuous place, an adaption of the rates is required. In our implementation, this is realized by setting the time of the next internal transition to the minimum of both points t_a and t_b and hence, force a rate adaption.

We want to point out that, due to rate adaption, it is possible that a non-linear dynamic transitions causes also non-linear behavior in static continuous transitions. For example, when quickly discharging a battery, a state can be reached where the available charge is empty and the bound charge is not yet empty. Hence, the static transition **discharge** can only output power which has been previously transformed from bound charge to available charge by the non-linear transition **fill_a**.

IV. Statistical Model Checking of HPnGs with non-linear continuous evolution

Stochastic Time Logic has been previously introduced for specifying, e.g., dependability properties, and is repeated in Section IV-A. Section IV-B discusses how the quantization function is used within the extended Statistical Model Checking approach. Section IV-C discusses how to cope with the approximation error that is introduced by the quantization. The resulting overall error is discussed in Section IV-D.

A. Stochastic Time Logic

The Stochastic Time Logic (STL) was introduced in [36] for the specification of properties of HPnGs. In [6], the definition was extended and adjusted for the Statistical Model Checking (SMC) approach. In the following, we use the latter definition of the syntax and semantics of STL. For a given point in time t, it is checked whether the probability that a formula Ψ holds meets a certain threshold Θ. Alternatively, a confidence interval for such a probability can be computed. According to [6], an STL formula is defined as:

$$\varphi ::= P_{\bowtie\Theta}(\Psi), \tag{11}$$

$$\Psi ::= tt \mid AP \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi \mathcal{U}^{[t_1,t_2]} \Psi, \tag{12}$$

for comparison operator $\bowtie \in \{<, >\leq, \geq\}$, probability bound $\Theta \in [0,1]$ and time points $t_1, t_2 \in \mathbb{R}_0^+$, with $t_1 \leq t_2$.

AP is a placeholder for any atomic property. Types of atomic properties, with $\sim \in \{=, <, >, \leq, \geq\}$, are summarized by Table II. The semantics of STL is defined in [6].

Considering the KiBaM example from Section II-D, we want to ensure that the available charge of the battery is not empty. The atomic property $x_\mathrm{a} \leq 0$ holds if the fluid level of the place **a** is less than or equal to zero, i.e., if the place is empty. (Recall that zero is the lower boundary for all fluid levels). Properties can be combined using negation, conjunction or the *Until* property. An example for an *Until* property is investigated in our case study (see Section V), namely $tt\ \mathcal{U}^{[0,48]}(x_\mathrm{a} \leq 0)$. This property holds if there exists a point in time t' within the time interval $[0, 48]$ (e.g., 48 hours), at which the place **a** is empty. (Note that in general the sub-property on the left side of $\mathcal{U}$ must hold up to t', which is always valid for $tt = \mathit{true}$.) We can then define a safety property, which requires that the probability that such an empty state is reached is reasonably low: $P_{\leq 0.1}(tt\ \mathcal{U}^{[0,48]}(x_\mathrm{a} \leq 0))$.

TABLE II
Atomic Properties

AP	Comparing...	To...
$x_P \sim a$	fluid level x_P of continuous place P	$a \in \mathbb{R}_0^+$
$m_P \sim b$	marking m_P of discrete place P	$b \in \mathbb{N}$
$c_T \sim c$	clock value c_T of deterministic transition T	$c \in \mathbb{R}_0^+$
$d_P \sim d$	drift d_P of continuous place P	$d \in \mathbb{R}$
$g_T \sim e$	enabling time g_T of general transition T	$e \in \mathbb{R}_0^+$
e_T	enabling status e_T of a transition T	*true*

B. How to use quantizers for Statistical Model Checking

Time-bounded model checking of HPnGs always aims at verifying if an STL property holds at a certain point in time t with a certain probability. Statistical Model Checking allows to either estimate the probability that an STL formula holds at time t, expressed as $P_{=?}[\Psi, t]$, or to check whether said probability matches the threshold $\bowtie\ \Theta$, formulated as $P_{\bowtie\Theta}[\Psi, t]$. Within our SMC approach, each simulation run is checked against the inner part of the STL formula Ψ, which is constructed according to Equation 12 and statistical methods, like the calculation of confidence intervals and hypothesis tests, are used to evaluate the whole set of simulation runs with regards to the probability operator. For a detailed description on the SMC approach, we refer to [6].

Supposed we want to check whether the available charge in the battery is lower than a given threshold b ($x_\mathrm{a} \leq b$) at a given point in time t_P. Since the diffusion between the two wells evolves non-linearly over time, a linear interpolation of the value of the continuous variable x_a between two events would result in a considerable error. Also in this case, quantizers can be used for non-linear continuous variables: since a quantized variable $q(t)$ also has a piecewise-linear derivative, we can verify a property at time point t_P, by linearly interpolating the value of $q(t)$ between the previous and the next event to compute $q(t_\mathrm{P})$. The execution of internal transitions as introduced in Section III-A, ensures that the distance between $q(t)$ to $x(t)$ never exceeds the pre-defined value Δq.

Hence, if the quantized variable that corresponds to the fluid level of a place is more than Δq away far from bound b, we can accurately decide whether $|b - x(t_\mathrm{P})| > 0$ holds:

$$x_\mathrm{a}(t_\mathrm{P}) \leq b = \begin{cases} \mathit{false} & \text{if } q(t_\mathrm{P}) - \Delta q > b, \\ \mathit{true} & \text{if } q(t_\mathrm{P}) + \Delta q < b, \\ \mathit{undetermined} & \text{otherwise.} \end{cases} \tag{13}$$

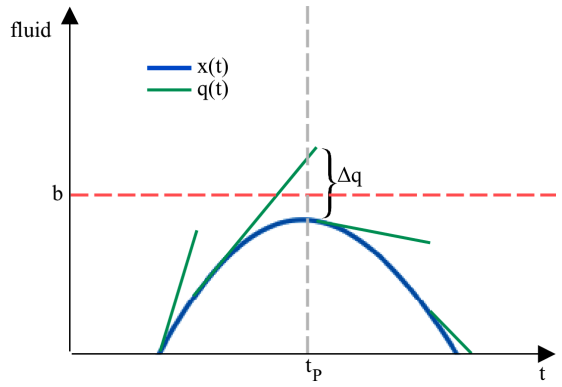

Fig. 5. Possible evolution of the state variable $x(t)$ and quantized variable $q(t)$ when comparing to a threshold b at time point t_P.

The third case of the above formula is illustrated in Figure 5. It might be that $x(t_P) < b$ while $q(t_P) > b$ or the other way round. At time point t_P, $q(t_P)$ exceeds the threshold b, while the true value $x(t_P)$ does not, which would result in a wrong outcome of the model checking algorithm. Hence, the simulation approach needs to be adjusted to properly deal with those cases. This is done by making the approach *property-dependent*, as discussed in the next section.

C. Property-dependent simulation

When model checking a property which takes into account the continuous state of the model at a specific time point t_P, the core idea is to make the simulation *dynamic*, by adding internal transitions for the considered continuous variables whenever they approach a boundary. The details on this procedure are described in the following. Note that nested STL properties can contain multiple atomic properties and hence, multiple boundaries may needed to be checked for one or more continuous variables. In the following, we describe the property-dependent simulation for a single boundary to be considered, but in case there are multiple boundaries required to validate a property, the algorithm loops over all boundaries and applies the following method for each of them.

Consider the state of a simulation run, when the next event has been determined, but not yet executed. This corresponds to completing Line 18 in Algorithm 1 from Section III-A. Let t_{Current} denote the execution time of the previous event and t_{Next} the occurrence of the previously determined next event. If the time point t_P, at which a property is checked, lies between t_{Current} and t_{Next}, we loop through all considered continuous places and call for each considered boundary a function named *checkAtomicFluidProperty()*. This function checks Equation 13 and inserts internal transitions whenever the result is undetermined. The Pseudo code for this function is presented in Algorithm 2, where $propertyTime$ represents t_P, $currentTime$ represents t_{Current} and $untilTime$ the maximum time bound t_{Until} of any (possibly nested) *Until* property. $event$ is the possible next event that has been determined.

In case the property does not contain any *Until* operator (Lines 2–7), the quantized variable of the considered place is linearly updated to the time of the property t_P (Line 3). The fluid level lies close to the bound of the property if the following condition holds (Line 4):

$$q(t_P) - \Delta q \leq b \leq q(t_P) + \Delta q. \tag{14}$$

In this case, an internal transition at t_P is enforced (Line 5–6), which approximates the value of $x(t_P)$ as defined in Equation 8 and updates $q(t_P)$, accordingly. This allows to verify the validity of the property by using the value of $q(t_P)$. If Condition 14 does not hold, we know that $x(t_P)$ differs from b with at least $|\Delta q|$, as already mentioned in the previous section, such that no additional internal transition is required.

Verifying the validity of a property that contains at least one *Until* operator works similarly, but requires considering a time interval instead of a single point in time. In this case (Lines 8–22), the algorithm determines the intersection of the interval $[t_P,\ t_{\text{Until}}]$ and the interval between the occurrence times of the previous and next event $[t_{\text{Current}},\ t_{\text{Next}}]$. This intersection is given by the interval $[t_{\text{Start}}, t_{\text{End}}]$ where $t_{\text{Start}} = max\{t_P, t_{\text{Current}}\}$ and $t_{\text{End}} = min\{t_{\text{Until}}, t_{\text{Next}}\}$ (Lines 9–10). For both of these interval boundaries, the quantized variable is linearly updated (Lines 11–12). Next, it is checked for both time points if the fluid level lies close to the bound of the atomic properties or if the quantizer and the fluid level lie on different sides of the bound, i.e., if Equation 15, Equation 16 or Equation 17 holds (Line 13).

$$q(t_{\text{Start}}) - \Delta q \leq b \leq q(t_{\text{Start}}) + \Delta q, \tag{15}$$

$$q(t_{\text{End}}) - \Delta q \leq b \leq q(t_{\text{End}}) + \Delta q, \tag{16}$$

$$(q(t_{\text{Start}}) - b)\,(q(t_{\text{End}}) - b) \leq 0. \tag{17}$$

If one of these three conditions is fulfilled, the algorithm determines the minimum time period to a point when either $q(t)$ or $x(t)$ will meet the value of b, starting from the previous event (Line 14). This is realized by calling the functions *getTimeToBoundHitByFluidLevel()* and *getTimeToBoundHitByQuantizer()*, respectively. The former uses a quadratic Taylor polynomial approximation for $x(t)$ as in Equation 8, sets it equal to the bound b and solves it for the time.

If the resulting least positive solution plus t_{Current} is before t_{End} (Line 15), an internal transition is enforced at this time. Hence, the value of $q(t)$ is updated to approximate $x(t)$, which ensures the required accuracy when approaching the bound b (Line 17).

One disadvantage that comes with this solution is that time-convergent behavior may occur if $q(t)$ repeatedly approaches the bound b before $x(t)$ does, such that the time between two consecutive internal transitions decreases and is always less than the time until $x(t)$ meets the bound. Theoretically, this issue would lead to an infinite number of internal transitions, but since floating point representation is limited, implementations finish within finite runtime. The tool HYPEG limits the representation for continuous variables and quantizers to eight decimal places to avoid floating point errors and therefore the time-convergent behavior has only low impact on the runtime.

Algorithm 2 checkAtomicFluidProperty($property$, $currentTime$, $propertyTime$, $untilTime$, $event$)

```
1: place = property.relatedPlace
2: if (property is not until property) then
3:     fluid = max(0, place.quantizedFluidLevel + place.quantizedDrift*(propertyTime - currentTime)
4:     if (fluid - place.quantum ≤ property.bound and property.bound ≤ fluid + place.quantum) then
5:         place.timeToNextTransition = propertyTime - currentTime
6:         event.update(place.timeToNextTransition)
7:     end if
8: else
9:     startPoint = max(currentTime, propertyTime)
10:    endPoint = max(untilTime, event.time)
11:    startFluid = max(0, place.quantizedFluidLevel + place.quantizedDrift*(startPoint - currentTime)
12:    endFluid = max(0, startFluid + place.quantizedDrift*(endPoint - startPoint)
13:    if ((startFluid - place.quantum <= property.bound and property.bound ≤ startFluid + place.quantum) or (endFluid -
       place.quantum <= property.bound and property.bound ≤ endFluid + place.quantum) or ((startFluid − bound) * (endFluid −
       property.bound) ≤ 0)) then
14:        timeToBound = min(place.getTimeToBoundHitByFluidLevel(), place.getTimeToBoundHitByQuantizer(), currentTime)
15:        if (currentTime + timeToBound < endPoint and timeToBound > 0) then
16:            place.timeToNextTransition = timeToBound
17:            event.update(timeToBound)
18:        end if
19:    end if
20: end if
21: return event
```

D. The resulting error

According to [7], the error in the quantized system is bounded by Δq, which can be precomputed given a maximum desired error bound Δe_i for each q_i. Given a linear time-invariant system $\dot{\widetilde{x}}(t) = A \cdot \widetilde{x}(t) + B \cdot u(t)$ and the associated QSS2 $\dot{x}(t) = A \cdot q(t) + B \cdot u(t)$, the i-th element of Δq can be computed as follows (c.f. Equation 30 [7]):

$$\Delta q_i = \frac{1}{n} \min_j \left(\frac{e_i}{T_{ji}} \right), \tag{18}$$

with vector Δq of size n and matrix T defined as:

$$T = |V| \cdot \text{diag} \left(\frac{|\lambda_i|}{|\mathbb{R}\text{e}(\lambda_i)|} \right) \cdot |V^{-1}| \cdot \Delta q. \tag{19}$$

V is the eigenvector matrix and λ_i are eigenvalues[2] of matrix A, where $|V^{-1}| \cdot A \cdot V = \text{diag}\,(\lambda_i|)$ holds.

However, Kofman does not take into account that, when simulating the system according to a DEVS model, the value for $x(t)$ needs to be approximated. For example, Taylor approximation results in an additional error being introduced with every internal transition. The implementation of this work uses 2nd-degree Taylor polynomial approximation. For each state variable $x(t)$, the error $R_2(t)$, which arises due to this Taylor approximation since the previous internal transition, is bounded by the *Lagrange remainder*, given by Equation 20 [37]:

$$R_2(t) = \frac{M}{3!}(t - t_j)^3, \tag{20}$$

where t_j is the occurrence time of the previous internal transition and M an upper bound on the third derivative for $x(t)$ for the interval (t_j, t), such that $\forall t^* \in (t_j, t) : |x^{(3)}(t^*)| \leq M$. The error of the Taylor approximation affects the computation of $x(t)$ and $q(t)$ across the whole simulation run, since every internal transition updates $x(t)$ based on its approximated previous value and $q(t)$ is set to the newly approximated value of $x(t)$. Note that our approach only considers the error caused by the quantization, but not the one due to the Taylor approximation.

[2]Note that, $\mathbb{R}\text{e}(\lambda_i)$ denotes the real component of λ_i.

V. Case study on the Kinetic Battery Model

We have carried out a case study based on the HPnG model from Section II-D, which models the Kinetic Battery Model (KiBaM) with predefined loads and one random outage. Both, the extended DES and SMC approaches presented in this work have been implemented in our tool HYPEG. Within this case study, we evaluate an STL property of the HPnG model and compare the output of HYPEG to an analysis technique, which has been implemented in Matlab and tailored to this specific model. Section V-A explains the main features of the Matlab implementation and Section V-B discusses the results of the case study. It further illustrates the capabilities of the approach when handling a large number of random variables by adding a repair mechanism to the existing model.

A. Analyzing the KiBaM using Matlab

We implemented a hard-coded analysis of the KiBaM example from Section II-D in Matlab[3] to compute the probability that the available charge is empty within the first 24 or 48 hours, i.e., checking the validity of the STL property $tt\ \mathcal{U}^{[0,t_U]}(x_\text{a} \leq 0)$ with $t_U \in \{24, 48\}$, as explained in Section IV-A.

The analysis is based on a *Stochastic Time Diagram (STD)* [2], [34], that can be created for an HPnG with an given initial state, in which the time is plotted by the vertical axis and the firing times of general transitions are entered on the remaining

[3]https://de.mathworks.com/products/matlab.html

axes. Since we have designed our example KiBAm model in such a way that only a single general transition (**power_off**) can be enabled and fire only once, a two-dimensional STD can be created, for which analysis has been investigated in [2], [34]. In such a two-dimensional STD, each point (s, t) represents a unique system state at time t if the general transition fires at time s. The diagram can be divided in two areas: the deterministic area, where $t < s$ holds and the general transition has not yet fired, and the stochastic area with $t > s$, where the general transition has already fired.

In our Matlab implementation, the STD for the KiBaM example is created with the given loads. The possible points in time, at which the general transition might fire, are discretized with step size 0.1. For each (discretized) firing time and for each specific load, the values of the two wells representing the available and the bound charge are computed, using the solved ODE system of the KiBaM as given in [33]. In case the available charge reaches its maximum capacity or gets empty, the point in occurrence time of the corresponding event is computed as follows.

In the solved ODE system, $a(t)$ is replaced with its maximum capacity and zero, respectively and the resulting equations are solved for t. Based on the gathered information, the STD is created and the state space in the diagram is partitioned into several regions, in which states have the same discrete marking and enabling status.

The probability, that the available charge is empty, is then computed as follows: All regions that correspond to states, in which the available charge is empty, are intersected with the line $t = 24$ respectively $t = 48$. From the intersection points, intervals for s can be derived, for which the considered property holds. Since the firing times have been discretized before, linear interpolation is used between two successive firing times. Next, the probability distribution of the general transition is integrated over each interval and all the values are added up, resulting in the total probability that the available charge is empty within the first 24 respectively 48 hours.

Summarizing, the simulation approach presented in this paper is validated using a dedicated Matlab implementation of the KiBaM. The latter is based on an analytical approach but not exact since general transitions are discretized (without estimating the introduced error). Our simulation approach has been efficiently implemented in a Java tool, which can be used for arbitrary HPnG models. However, the run time of the total simulation depends on the number of required simulation runs. So, on the one hand, Matlab is a relatively slow language and on the other hand, simulation requires a large number of simulation runs to achieve a certain accuracy. This is why we include computation times.

B. Case study results

We used the Matlab implementation to validate the results of our tool HYPEG, which implements the approach presented in this paper. Therefore, we have defined different scenarios with three probability distributions (uniform, normal and exponential) for the general transitions. Starting from the initial state at $t = 0$, we have computed confidence intervals for the probability that the available charge gets empty within the first 24 or 48 hours, i.e., using Statistical Model Checking to solve the problem $P_{=?}[tt\ \mathcal{U}^{[0,t_U]}(x_{\mathrm{a}} \leq 0), 0.0]$ with $t_U \in \{24, 48\}$. The confidence intervals have a 99% confidence level and an interval width ≤ 0.02. Each scenario is investigated for $\Delta q = 0.1$ and $\Delta q = 1$ for both continuous places, **a** and **b**.

Table III presents the results of Matlab and HYPEG and Figure 6 illustrates how the results of HYPEG (green and red) deviate from the probability obtained from Matlab (blue diamond). In most scenarios, the latter lies within the confidence intervals (triangles) of the former. Especially for $\Delta q = 0.1$, the mean (green diamond) lies close to the Matlab result, which shows that a smaller value for Δq improves the accuracy of the result.

In Scenario 1b, HYPEG overestimates the probability for $\Delta q = 1$ (red), such that the value obtained from Matlab lies outside the confidence interval. However, results match for $\Delta q = 0.1$. Scenario 3b presents a special case, in which the probability lies very close to 1, such that HYPEG finishes

TABLE III
CASE STUDY RESULTS COMPARING THE RESULTS OF HYPEG TO THE RESULTS OF MATLAB

Setting			**HYPEG**					**Matlab**	
Scenario	Distribution	t_U	Δq	Mean	Confidence Interval	Time	Runs	Probability	Time
1a)	*uniform(0, 48)*	24	0.1	0.101339	$[0.091341, 0.111338]$	21.5 s	6049	0.102645	56.93 s
			1	0.100117	$[0.090117, 0.110117]$	8.65 s	5983		
1b)	*uniform(0, 48)*	48	0.1	0.577498	$[0.567498, 0.587498]$	88.63 s	16194	0.574231	56.93 s
			1	0.599699	$[0.589699, 0.609699]$	28.53 s	15933		
2a)	*normal(12, 6)*	24	0.1	0.118343	$[0.108344, 0.128342]$	24.15 s	6929	0.119231	53.05 s
			1	0.12262	$[0.112621, 0.13262]$	9.36 s	7144		
2b)	*normal(12, 6)*	48	0.1	0.969254	$[0.959259, 0.979249]$	11.22 s	1984	0.970734	53.05 s
			1	0.978184	$[0.968186, 0.988183]$	4.37 s	1421		
3a)	*exponential(0.5)*	24	0.1	0.917162	$[0.907162, 0.927162]$	12.07 s	5046	0.914862	52.78 s
			1	0.917479	$[0.907479, 0.927478]$	5.51 s	5029		
3b)	*exponential(0.5)*	48	0.1	1	$[1, 1]$	4.71 s	1000 (min)	0.999991	52.78 s
			1	1	$[1, 1]$	2.88 s	1000 (min)		

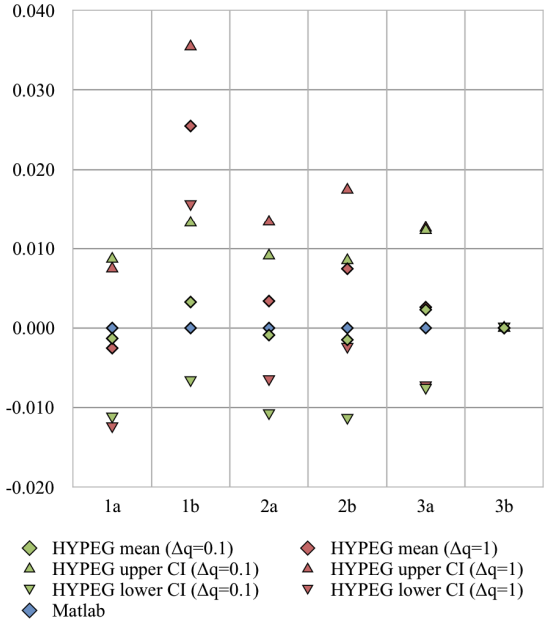

Fig. 6. Relative deviations from HYPEG results to Matlab results.

after having the property fulfilled in the pre-defined minimum number of simulation runs (here 1000 runs). Hence, HYPEG returns 1 as result since no runs occurred, in which the property does not hold. A larger number of minimum runs would probably lead to more accuracy in this case.

The case study has been executed on a *macOS Sierra* system (2.7 GHz Intel Core i5 processor and 8 GM RAM). On this system, the Matlab analysis took between 52.78 to 56.93 seconds, whereas the sub scenarios a) and b) of each scenario could be evaluated together in one execution, i.e., the STD was created only once per scenario. The run time of HYPEG varied between 4.71 and 88.63 seconds for $\Delta q = 0.1$ and between 2.88 and 28.53 seconds for $\Delta q = 1$, which underlines the trade-off between performance and accuracy.

The advantage of the presented modeling and simulation approach lies in the easy integration of general transitions with possibly different probability distributions. To illustrate the capabilities of the approach and tool, we have extended the previous model, such that a failure is followed by a repair after a randomly distributed amount of time, as shown in Figure 7.

TABLE IV
CASE STUDY RESULTS FOR EXTENDED MODEL WITH REPAIR PROCESS

Sc.	Mean	Confidence Interval	Time	Mean #RV	Max #RV
1a)	0	[0, 0]	5.17 s	1.16	6
1b)	0.001	[0, 0.003581]	7.37 s	3.05	10
2a)	0.005	[0, 0.010759]	3.34 s	2.52	7
2b)	0.003	[0, 0.007466]	9.33 s	5.75	12
3a)	0.0375	[0.027501, 0.047499]	11.46 s	9.39	19
3b)	0.049196	[0.039199, 0.059193]	22.58 s	19.04	33

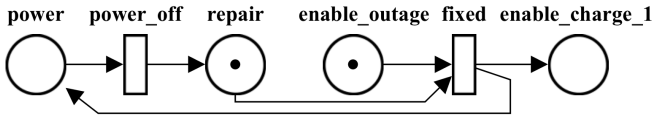

Fig. 7. Failure and repair part of HPnG model in repair mode.

Note that firing the repair transition again enables a random failure. This cycle of general transition firings introduces a finite number of random variables in the system before a finite maximum simulation time.

Table IV shows the results for the extended model, where the failure transition is distributed as indicated in Table III and the repair transition fires according to a uniform distribution (i.e., *uniform(0, 6)*) for all scenarios. Table IV shows the mean and maximum number of random variables (per simulation run) in the last two columns. The mean number of random variables lies between 1 and 19, depending on the failure distribution. In Scenario 3*b*, around 19 general transition firings occurred in average per run, simulated within a run time of less than 23 seconds.

Overall, these results show that in most cases HYPEG is able to produce sufficiently precise results, when applying Statistical Model Checking to an HPnG with non-linear continuous evolution, and thereby needs less time than a hard-coded analytical approach.

VI. Conclusion

This work extends the syntax and semantics of hybrid Petri nets with general transitions to include so-called dynamic continuous transitions. This new kind of transition allows modeling non-linear continuous evolution in HPnGs. We presented an approach which applies discrete-event simulation to any model of the extended formalism by transforming the system into a QSS2. Furthermore, we present algorithms for Statistical Model Checking in the extended approach. These extensions have been implemented in our tool HYPEG and compared to a hard-coded Matlab analysis within a case study on the Kinetic Battery Model. As a result, our tool has output sufficiently precise results in a comparably short run time. Future work will compare our approach to classical numerical ODE solvers, like Runge-Kutta-Fehlberg 4(5) [38], which uses a fourth order approximation scheme.

The presented approach has been restricted to systems whose continuous trajectories can be expressed by systems of ODEs. Since QSS2 have been also extended for differential algebraic equation (DAE) systems [22], future work might extend the approach to HPnGs with DAE representations and investigate the error propagation within the presented approach. Next to this, we will investigate how non-determinism can be included in HPnGs and how to simulate these models in combination with schedulers for non-deterministic decisions.

Acknowledgements

We thank the anonymous reviewers for their valuable comments.

References

[1] M. Gribaudo and A. Remke, "Hybrid Petri nets with general one-shot transitions," *Performance Evaluation*, vol. 105, pp. 22 – 50, 2016.

[2] H. Ghasemieh, A. Remke, B. Haverkort, and M. Gribaudo, "Region-based analysis of hybrid Petri nets with a single general one-shot transition," in *10th International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2012, pp. 139–154.

[3] A. Godde and A. Remke, "Model checking the STL time-bounded Until on hybrid Petri nets using Nef polyhedra," in *14th European Workshop on Performance Engineering*. Springer, 2017, pp. 101–116.

[4] J. Hüls, S. Schupp, A. Remke, and E. Ábrahám, "Analyzing hybrid Petri nets with multiple stochastic firings using HyPro," in *11th EAI International Conference on Performance Evaluation Methodologies and Tools*. ICST, 2018.

[5] M. Jongerden and B. Haverkort, "Which battery model to use?" in *24th UK Performance Engineering Workshop*, ser. Department of Computing Technical Reports. Imperial College London, 7 2008, no. 9, pp. 76–88.

[6] C. Pilch and A. Remke, "Statistical Model Checking for hybrid Petri nets with multiple general transitions," in *47th IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2017, pp. 475–486.

[7] E. Kofman, "A second-order approximation for DEVS simulation of continuous systems," *SIMULATION*, vol. 78, no. 2, pp. 76–89, 2002.

[8] C. Pilch and A. Remke, "HYPEG: Statistical Model Checking for hybrid Petri nets," in *11th EAI International Conference on Performance Evaluation Methodologies and Tools*. ICST, 2018.

[9] J. Manwell and J. McGowan, "Lead acid battery storage model for hybrid energy systems," *Solar Energy*, vol. 50, no. 5, pp. 399–405, 1993.

[10] M. Jongerden, J. Hüls, B. Haverkort, and A. Remke, "Assessing the cost of energy independence," in *IEEE International Energy Conference*. IEEE, 2016, pp. 1–6.

[11] G. Nies, M. Stenger, J. Krl, H. Hermanns, M. Bisgaard, D. Gerhardt, B. Haverkort, M. Jongerden, K. Larsen, and E. Wognsen, "Mastering operational limitations of LEO satellites - the GOMX-3 approach," in *Proceedings of the 23rd IAA Symposium on Small Missions at the 67th International Astronotical Congress*. International Astronautical Federation, 2016, pp. 1–15.

[12] H. Ghasemieh, B. Haverkort, M. Jongerden, and A. Remke, "Energy resilience modelling for smart houses," in *45th IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 275–286.

[13] J. Hüls and A. Remke, "Energy storage in smart homes: Grid-convenience versus self-use and survivability," in *24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2016, pp. 385–390.

[14] K. Clement-Nyns, E. Haesen, and J. Driesen, "The impact of charging plug-in hybrid electric vehicles on a residential distribution grid," *IEEE Transactions on Power Systems*, vol. 25, no. 1, pp. 371–380, 2010.

[15] J. Hüls and A. Remke, "Coordinated charging strategies for plug-in electric vehicles to ensure a robust charging process," in *10th EAI International Conference on Performance Evaluation Methodologies and Tools*. ICST, 2016.

[16] J. Hu, J. Lygeros, and S. Sastry, "Towards a theory of stochastic hybrid systems," in *3rd International Workshop on Hybrid Systems: Computation and Control*, N. Lynch and B. H. Krogh, Eds. Springer, 2000, pp. 160–173.

[17] N. Bertrand, P. Bouyer, T. Brihaye, Q. Menet, C. Baier, M. Größer, and M. Jurdzinski, "Stochastic timed automata," *Logical Methods in Computer Science*, vol. 10, no. 4, 2014.

[18] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," Cornell University, USA, Tech. Rep., 05 1993.

[19] A. Julius, "Approximate abstraction of stochastic hybrid automata," in *9th International Workshop on Hybrid Systems: Computation and Control*. Springer, 2006, pp. 318–332.

[20] E. Kofman and S. Junco, "Quantized-state systems: a DEVS approach for continuous system simulation," *Simulation: Transactions of the Society for Computer Simulation International*, vol. 18, no. 3, pp. 123–132, 2001.

[21] B. Zeigler and J. Lee, "Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment," in *12th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls: Enabling Technology for Simulation Science*, vol. 3369. SPIE, 1998, pp. 49–59.

[22] E. Kofman, "Quantization-based simulation of differential algebraic equation systems," *SIMULATION*, vol. 79, no. 7, pp. 363–376, 2003.

[23] ——, "Discrete event simulation of hybrid systems," *SIAM Journal on Scientific Computing*, vol. 25, no. 5, pp. 1771–1797, 2004.

[24] A. Legay and B. Delahaye, "Statistical Model Checking: An Overview," *CoRR*, vol. abs/1005.1327, 2010.

[25] D. El Rabih and N. Pekergin, "Statistical model checking using perfect simulation," in *7th International Symposium on Automated Technology for Verification and Analysis*. Springer, 2009, pp. 120–134.

[26] J. Bogdoll, A. Hartmanns, and H. Hermanns, "Simulation and statistical model checking for modestly nondeterministic models," in *16th International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, 2012, pp. 249–252.

[27] G. Horton, V. Kulkarni, D. Nicol, and K. Trivedi, "Fluid stochastic Petri nets: Theory, applications, and solution techniques," *European Journal of Operational Research*, vol. 105, no. 1, pp. 184–201, 1998.

[28] G. Ciardo, D. Nicol, and K. Trivedi, "Discrete-event simulation of fluid stochastic Petri nets," *IEEE Transactions on Software Engineering*, vol. 25, no. 2, pp. 207–217, 1999.

[29] A. David, K. Larsen, M. Mikucionis, D. Poulsen, A. Legay, S. Sedwards, and D. Du, "Statistical model checking for stochastic hybrid systems," in *1st International Workshop on Hybrid Systems and Biology*. ARXIV, 2012, pp. 122–136.

[30] A. David, K. Larsen, A. Legay, M. Mikucionis, and D. Poulsen. (2018, January) Uppaal SMC tutorial. [Online]. Available: http://www.it.uu.se/research/group/darts/papers/texts/uppaal-smc-tutorial.pdf

[31] C. Ellen, S. Gerwinn, and M. Frnzle, "Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 485–504, 2015.

[32] C. Budde, P. D'Argenio, A. Hartmanns, and S. Sedwards, "A statistical model checker for nondeterminism and rare events," in *Tools and Algorithms for the Construction and Analysis of Systems*, D. Beyer and M. Huisman, Eds. Springer, 2018, pp. 340–358.

[33] H. Hermanns, J. Krčál, and G. Nies, "Recharging probably keeps batteries alive," in *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer, 2015, pp. 83–98.

[34] H. Ghasemieh, A. Remke, and B. Haverkort, "Analysis of a sewage treatment facility using hybrid Petri nets," in *7th EAI International Conference on Performance Evaluation Methodologies and Tools*. ICST, 2013, pp. 165–174.

[35] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth, "On the Lambert W function," *Advances in Computational Mathematics*, vol. 5, no. 1, pp. 329–359, 1996.

[36] H. Ghasemieh, A. Remke, and B. Haverkort, "Survivability evaluation of fluid critical infrastructures using hybrid Petri nets," in *2013 IEEE 19th International Symposium on Dependable Computing*. IEEE, 2013, pp. 152–161.

[37] (2018, April) Taylor series - error bounds. Brilliant.org. [Online]. Available: https://brilliant.org/wiki/taylor-series-error-bounds/

[38] E. Fehlberg, "Low-order classical runge-kutta formulas with stepsize control and their application to some heat transfer problems," NASA, Tech. Rep. NASA TR R-315, 07 1969.