On SMC-Based Dependability Analysis in LoLiPoP-IoT Project

Josef Strnadel[0000−0001−6327−5990], Jakub Lojda[0000−0002−5745−587X], Pavel Smrž[0000−0002−5638−1362], and Václav Šimek[0000−0002−9837−4128]*

Brno University of Technology, Bozetechova 2, 61200 Brno, Czech Republic
{strnadel|ilojda|simekv|smrz}@fit.vut.cz
https://www.fit.vut.cz/

Abstract. Many systems require certain level of dependability to fulfill their purpose in predefined conditions. To check whether such a requirement can be met, the designer of a system must use proper means to assess dependability qualitatively or quantitatively, whereas this paper focuses on the latter assessment manner. The first problem with the assessment is that we cannot judge it except by evaluating its sub-attributes such as reliability, availability or maintainability. The second problem relates to the assessment itself – ideally, assessment builds on an analytical solution; however, if it does not exist, its presumptions are violated etc., an alternative approach must take place. This paper presents our alternative, simulation based approach with a special attention paid to reliability and maintainability; it builds on stochastic timed automata, an instrument able to model a wide class of systems/conditions of one's interest. In our approach, the assessment process takes the advantage of the statistical model checking technique, powerful enough to quantify dependability attributes in realistic situations and with a predefined degree of uncertainty. Finally, the paper evaluates our approach, outlines our research perspectives and gives a conclusion.

Keywords: Fault · Failure · Random variable · Dependability · Reliability · Maintainability · Analysis · Quantitative assessment · Model · Timed automaton · Simulation · Statistical model checking · Uncertainty

1 Introduction

Motivation&Scope of this Paper. Traditionally, our research focuses on modeling & analyzing various issues that relate to dependability of systems.

In the modeling field, it focuses especially on issues related to uncertainty & dynamics of phenomena such as the occurrence time of a fault, time a fault spends in a system, time it takes to initiate, start or finish a repair mechanism and time to failure of a service provided by a system.

* This work was supported by the Chips JU Project LoLiPoP-IoT (Long Life Power Platforms for Internet of Things), www.lolipop-iot.eu, grant agreement No. 101112286, which is jointly funded by the Chips Joint Undertaking and national public authorities.

In the analysis field, it focuses on the assessment of dependability attributes such as reliability, maintainability or availability and their control using various approaches to fault tolerance, maintenance etc.

The fact that the kind of modeling/analysis could fail when classical approaches are applied in uncommon – but still realistic – conditions, motivated us to show that instruments such as stochastic timed automata and statistical model checking are able to avoid such a failure. To demonstrate the applicability of our approach, we used a set of case-studies, each of which can be used as a design-pattern for solving a particular problem of one's interest.

LoLiPoP-IoT Project. The project [32] aims to design advanced *energy harvesting* (EH) and micro-power management solutions for long-lasting *wireless sensor network* (WSN) devices. This project focuses on solutions that will be integrated into (or near) monitored equipment and infrastructure, serving as pivotal technology platforms for data collection. By enabling effective and uninterrupted data collection, detection of potential anomalies, and performance monitoring, the project enhances the efficiency and potential of the trillion-sensor economy projected for 2025. The collected data promises unprecedented opportunities, potentially saving billions of euros, reducing carbon emissions, and increasing the usage of renewable energy across various industries.

The LoLiPoP-IoT project covers these important Application Domains, that comprehend Industry 4.0, Smart Mobility, and efficiency in energy management. These include Asset Tracking, Energy Efficiency and Comfort Optimization, but also, same to the focus of this paper, Condition Monitoring and Predictive Maintenance.

Condition monitoring and predictive maintenance means a continuous monitoring of equipment and machinery parameters. This allows for the detection of anomalies indicating developing faults. Predictive maintenance, utilizing Industry 4.0 principles, minimizes maintenance overhead and downtime by enhancing operational efficiency and significantly reducing costs. While the LoLiPoP-IoT project covers all the Application Domains, the ones targeted at condition monitoring and predictive maintenance include:

(i) tracking of air cargo pallets and monitoring their condition,
(ii) supervision of a paint-dying facility,
(iii) surveillance of bearings,
(iv) examination of fermentation processes in agriculture,
(v) optimization of district heating, and
(vi) monitoring a fill level of lightweight vehicles.

The overview of targets of the project in the scope of Condition Monitoring and Predictive Maintenance is shown in Fig. 1.

The condition monitoring and predictive maintenance in these use cases, aim to continuously monitor equipment and machinery parameters to detect anomalies indicating potential faults. In the context of dependability and predictive maintenance, ensuring continuous operation and timely maintenance is essential for optimizing productivity and preventing costly failures, which in turn ensures the reliability and availability of critical systems.

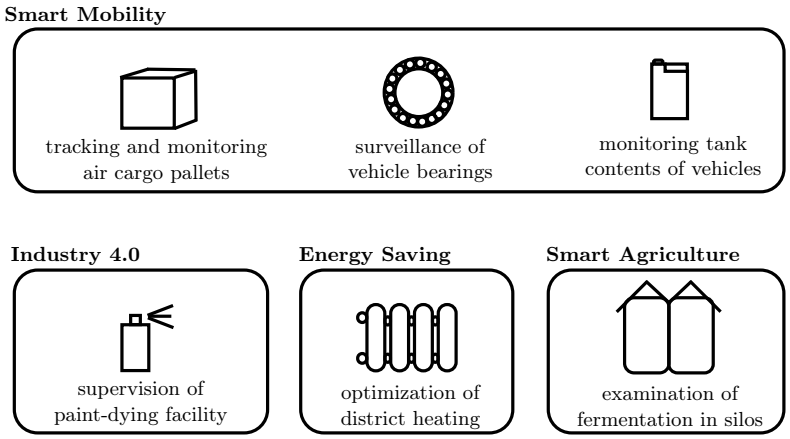


Fig. 1. Overview of the project targets from the point of view of Condition Monitoring and Predictive Maintenance

Structure of this Paper. The rest of this paper is organized as follows. Sect. 2 summarizes key preliminaries, including facts such as dependability and its assessment, related work in areas of our interest and means/methods we used in our approach. Sect. 3 presents our approach to constructing models for the areas of our interest such as faults, fault-tolerant architectures, repair and maintenance mechanisms. Sect. 4 summarizes representative results – we have achieved so far based on the models – to show the applicability of our approach. Sect. 5 concludes the paper and outlines our future research perspectives.

2 Preliminary

2.1 Dependability and Its Assessment

Dependability as a service failure. Qualitativelly, *dependability* of a system – often being understood as a provider of a service – can be seen as its ability to provide a specified *service* in specified conditions and for a specified period of time, whereas by "specified" we, ideally, mean "intended". A deviation between the specified service and the provided one is called a *failure* – see Fig. 2.

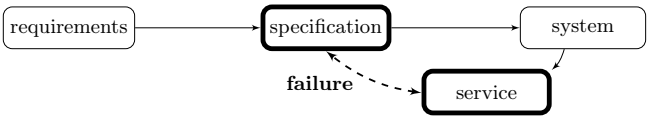


Fig. 2. Illustration to the failure of a service

A deviation between the specified service and the provided one is called a *failure* – see Fig. 2. Typically, a failure results from (series of) cause-and-effect elements (such as states, events, actions) and their products – see Fig. 3.

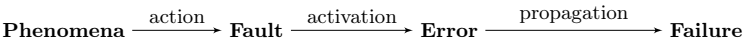


Fig. 3. Typical causal chain resulting into a failure. Initially, a fault is *dormant* (*passive*) – i.e., it is present in a system, but has no impact to its state/behavior; later, it may get activated, become *active* and manifest itself by an *error* (i.e., a deviation from the intended state). Alike a fault, an error is dormant until it gets activated, i.e., by using it during a computation; if an error propagates beyond the boundary/interface of a (sub-)system affected by an error, it may propagate through further (sub-)systems up to the outputs observable by a user, i.e., result into a failure

A failure may have different forms, *failure modes*, ranked according to the *severity* of the failure. For example, DO-178B standard defines the so-called *seriousness classes* to express a particular level of severity – see Fig. 4.

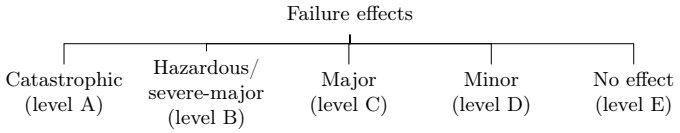


Fig. 4. Seriousness classes of failures effects based on DO-178B

Attributes of dependability. Dependability, seen qualitatively or quantitatively, is a complex feature composed of multiple *attributes* [4,16] – see Fig. 5. Let us emphasize that we can't judge it except by evaluating its sub-attributes, each giving a partial image about dependability; from the quantitative viewpoint, we can't evaluate dependability by a single, overall number.

This paper focuses only on the quantitative assessment of dependability. Ideally, the quantitative assessment provides data that in/validates an existing qualitative assessment; vice versa, one may derive the qualitative assessment from an existing quantitative assessment.

The assessment of dependability must be based on precisely defined concepts. Since supposedly identical systems – operating under similar conditions – fail at different times, associated phenomena can only be expressed stochastically. In some specific cases, such as an exponentially-distributed probability density function of a time-to-fault random variable, the assessment can be done on analytical basis [16]. However, in general, an analytical solution need not exist, so one must use an alternative solution instead, see Sect. 2.2.

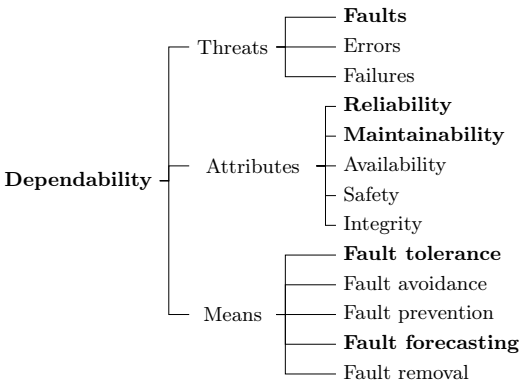


Fig. 5. Dependability characteristics [4]: threats, attributes and means. Facts of our interest are highlighted

Quantitative dependability assessment (QDA) problem. To better understand the QDA problem, let us express it mathematically in the following text – consult Tab. 1, please. Let X_{TTF} be a continuous random variable used to represent the *time-to-fault* (TTF) and $f_{X_{TTF}}(t)$, $F_{X_{TTF}}(t)$ and $R_{X_{TTF}}(t)$ functions used to represent *probability density function* (PDF), *cumulative distribution function* (CDF) and *reliability function* ("reliability" in brief) of X_{TTF}, resp. Similarly (for repairable systems), we may introduce a continuous random variable X_{TTR} used to represent the *time to repair* (TTR), called the *repair rate* too. If it is unambiguous, we can omit X_{TTF} and X_{TTR} to make the notation more readable. Tab. 1 clearly shows an important fact – i.e., that all attributes of dependability can be derived, e.g., from PDFs. Having them, we are able to evaluate an attribute using a common calculus.

Fault/repair rates and mean times. A relation between fault and repair rates is illustrated in Fig. 6. Initially, a system is fault-free. If a fault occurs in a system, the fault may manifest itself by an error, a consequence of which may propagate further into the system and result into a failure – see Fig. 3 too.

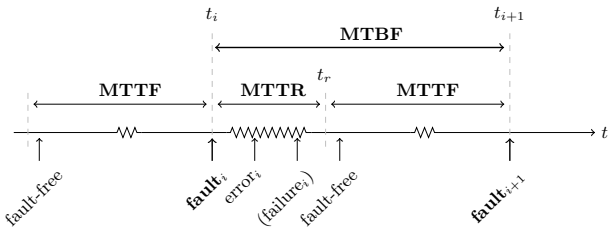


Fig. 6. Illustration to the relation between fault and repair rates and to MTBF

Table 1. Basic symbols and formulas to assess dependability

Attribute		
Symbol	**Evaluation**	**Meaning**
$f_{X_{TTF}}(t)$	identified empirically	probability density function, PDF
$f_{X_{TTR}}(t)$		
$F_{X_{TTF}}(t)$	$\int_{-\infty}^{t} f_{X_{TTF}}(x)\ dx$	cumulative distribution function, CDF
$F_{X_{TTR}}(t)$	$\int_{-\infty}^{t} f_{X_{TTR}}(x)\ dx$	
$R_{X_{TTF}}(t)$	$1 - F_{X_{TTF}}(t)$	reliability (survival) function, R
$h_{X_{TTF}}(t)$	$f_{X_{TTF}}(t)/R_{X_{TTF}}(t)$	hazard/failure (rate) function, h
$MTTF_{X_{TTF}}$	$\int_0^\infty t \times f_{X_{TTF}}(t)\ dt =$ $= \int_0^\infty 1 - F_{X_{TTF}}(t)\ dt$	mean time to failure, $MTTF$
$M_{X_{TTR}}(t)$	$\int_0^t f_{X_{TTR}}(s)\ ds$	maintainability, M
$MTTR_{X_{TTR}}$	$\int_0^\infty t \times f_{X_{TTR}}(t)\ dt =$ $= \int_0^\infty 1 - F_{X_{TTR}}(t)\ dt$	mean time to repair, $MTTR$
$MTBF_{X_{TTF}}^{X_{TTR}}$	$MTTF_{X_{TTF}} +$ $+\ MTTR_{X_{TTR}}$	mean time between failures, $MTBF$
$A_{X_{TTF}}^{X_{TTR}}(\infty)$	$\lim_{t\to\infty} A_{X_{TTF}}^{X_{TTR}}(t) =$ $= \frac{MTTF_{X_{TTF}}}{MTBF_{X_{TTF}}^{X_{TTR}}}$	availability (steady-state), A

For a *non-repairable* system, this is the first and only fault that finishes the mission/operation of the system; so, MTTF must be, ideally, much bigger than the expected mission/operation time, guarantee period etc.

A repairable system can recover from a fault (practically, from a given class only), e.g., by a masking, detection followed by a reconfiguration/repair etc.; this process, however, takes some time – MTTR in average. Then, the system (again) becomes fault-free and can operate until a new fault occurs. However, if a new fault occurs before the repair finishes, the behavior of the system may get undefined. As $A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTR+MTTF}$, it must hold (ideally) $MTTR \ll MTTF$ to maximize availability of a system. For fixed $MTTF$, the value of A increases with the decreasing value of $MTTR$; ideally, $\lim_{MTTR\to 0} A = 1$, i.e., max. if the repair takes no time.

Maintenance analysis and control. A key dependability attribute of our interest is *maintainability*, M(t), formally introduced in Tab. 1 (as the probability that a repair finishes by $t > 0$, initiated by the occurence of a fault at t=0). Practically, it affects, e.g., the amount of time a system spends in a downtime state (before it returns back to an uptime state) as well as attributes such as MTTR, MTTB and thus, availability. It can be controlled by means of various *maintenance* policies that can be classified into four classes [10], see Fig. 7.

The simplest policy is *corrective* (called *reactive* too) – as it gets apply after a fault only, a system must enter a downtime state to initiate it. The second one is *condition-based* – continuously, it monitors a proper set of states/quantities to identify the healthy level of a system; the maintenance gets activated only if

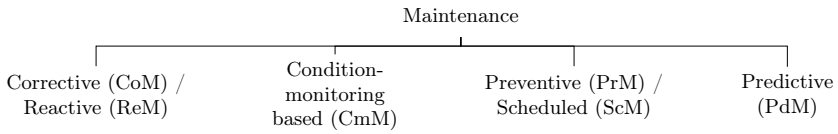


Fig. 7. Basic classification of maintenance policies/paradigms

the system is found in a non-healthy condition, so the policy can prevent from entering a downtime state. The third policy is *preventive* (called *scheduled* too) – usually, it starts periodically, based on historical failure data, in the hope of preventing a failure; however, its periodic activity may have negative effects. The last policy is *predictive* – it tries to predict how much it is likely that a failure occurs in a predefined interval of time; such a prediction allows a system to make it ready for recovering from the failure and, ideally, to minimize downtime.

With regard to Fig. 6, let us recall that, ideally, the value of parameters such as $MTTR$, $MTBF$ should be minimal or, at least, "As Low As it is Reasonably Practical" (ALARP). As mentioned in relation to Fig. 7, such an optimization is very difficult if a system applies a simple maintenance such as CoM/ReM; however, one can approach to the optimum if the system uses a properly set PrM/ScM process or a more complex CmM/PdM process. Each of the maintenance approaches has its benefits and costs [27]: CoM/ReM can be easily to implemented (in an event-driven manner), but it typically leads to an expensive downtime; PrM/ScM can be easily to implemented too (in a time-triggered manner), but may can – wastefully – replace a fault-free equipment or miss a failure that have occurred before the start of maintenance; CmM can be relatively low-cost in terms of needed equipment, but may consume a lot of energy as it intensively uses a computing/communication infrastructure; finally, PdM can predict a failure before it occurs, and maintenance can be planned and conducted in advance to avoid the failure. The authors of [28,38] identify the three approaches for making the predictions: model-based (e.g., using rules or physical models), data-driven (e.g., using machine/deep learning) and hybrid. Although our approach is hybrid too, we present only its model-based part in this paper.

2.2 Related Work

Tab. 1 shows that to assess dependability, data such as PDF for random variables such as X_{TTF} and X_{TTR} are needed. Typically, data like that are gathered from practice, estimated based on previous experience or based on a qualified model of reality. Then, a particular dependability attribute, such as R, M or A, can be evaluated. Ideally, the attribute can be identified using an analytical solution, if it exists and its assumptions are met [16]. For example, such an assumption may expect that X_{TTF} (alike, X_{TTR} etc.) follow a predefined distribution of probability. Particularly, for exponentially distributed X_{TTF} and X_{TTR} parameterized by λ and μ, resp., the following analytical solution exists – written in

the simplified notation: $f(t) = \lambda e^{-\lambda t}$, $F(t) = 1 - e^{-\lambda t}$, $R(t) = e^{-\lambda t}$, $h(t) = \lambda$, $MTTF = \frac{1}{\lambda}$. , $M(t) = 1 - e^{-\mu t}$, $MTTR = \frac{1}{\mu}$, $A(t) = \frac{\mu}{\mu+\lambda} + \frac{\lambda}{\mu+\lambda} e^{-(\mu+\lambda)t}$.

Despite such assumptions comply with a wide range of practical needs, they may be very limiting and not strong enough to cope easily (or, at all) with realistic issues such as assessing the reliability of a system across its life time [7, 24, 26, 35] or "dynamic reliability" [14, 37] trying to cope with facts such as the transience of faults in a system, stochasticity of a recovery process and faults introduced into a system before the process is complete. To cope with factors like that, instruments such as *Dynamic Fault Trees*, (DFTs), *Driven Boolean Markov Processes* (DBMP), or *Dynamic Reliability Block Diagrams* exist. (DRBDs). Further instruments extend earlier concepts such as *Stochastic Petri Nets*, (SPNs), *Stochastic Reward Nets*, (SRNs), *Stochastic Activity Networks*, (SANs) or *Stochastic Process Algebra*. (SPA). But, also instruments like that suffer from above-mentioned disadvantages, i.e., they require complex procedures to cope with the "problematic" factors and/or long time for the accurate dependability assessment [37].

Many alternatives to the analytical solution of the reliability assessment problem rely on the *Monte Carlo* simulation [15]. However, they suffer from the long running time and a need to solve further problems, such as speeding up the simulation [23], to make the assessment applicable in practice. Authors of [36] substituted the Monte Carlo approach by a more efficient stochastic analysis over a restricted model. Further works, such as [8, 9, 25, 29], build on Markov models and reward models with stochastic behaviors. They use the model checking technique [5] from the PRISM tool [19] to quantify a property of a system.

Further approaches, especially in the field of *reliability-centered maintenance* (RCM), build on instruments such as *Fault Maintenance Trees* [1, 31] and *Fault trees* [33, 34] optimize maintenance planning by maintaining critical assets more intensively than less critical ones in order to well-balance costs and value of maintenance. *Fault Tree Analysis* (FTA) is a popular methodology [33], commonly used in industry. Traditional FTA is very useful to analyse the reliability of systems when failure rates are given, typically exponentially-distributed TTFs. In practice, however, these failure rates are strongly affected by maintenance – as FTs are unable to reflect that, FTA is not suitable to compare the various maintenance policies. To overcome these limitations and to determine the effect of different maintenance strategies on system reliability and costs, fault maintenance trees (FMTs) have been proposed in [31]; e.g., they introduced the *rate dependency* (RDEP) gate to allow the failure of a component to accelerate the degradation of other components, they allowed one to combine FTs with arbitrary probability distributions (of TTF, TTR) and with maintenance models and they summarized facts needed to model&analyze maintenance: degradation of components, inspections, and repairs. But even this approach does not include many realistic facts such as imperfections in the performance of inspection and maintenance, fluctuation of inspection times and maintenance, non-zero repair times, transient/intermittent faults and dynamic/temporal redundancy.

2.3 Timed Automata and Statistical Model Checking

Timed Automata. Various modeling means have been developed to describe the behavior of systems [11]. A particular behavior of a system, described by a computational model ($\mathcal{M}$) can be seen as a (potentially infinite) *path* through a digraph $G_{\mathcal{M}}$ that corresponds to $\mathcal{M}$. Our paper builds on the *timed automata* ($\mathcal{TA}$) formalism [3] that extends non-timed formalisms by temporal aspects. 1, use *moves* instead of *actions* etc. [5, 11] The "classical" definition of $\mathcal{TA}$ can be further extended [17] by further important concepts such as variables, communication channels, stochastic decisions and by the ability to reflect prices (costs, rewards), to solve differential equations [6, 13]. Also, one can use a certainly known (deterministic, fixed) delay, unknown (non-deterministic) delay or to combine the approaches. This paper builds on the so-called *Stochastic Timed Automata* (STA) as used in the UPPAAL SMC toolset [13].

One of the biggest advantages of $\mathcal{TA}$ formalism is that it allows one to model uncertainty due to the variation of parameters in time; such a variation can be described stochastically by means of $\mathcal{TA}$, e.g., using random variables following an appropriate distribution of probability. Among the others, $\mathcal{TA}$ allows one to model the variability using the so-called *Stop-watch* concept (see Fig. 8). In the figure, a 3-state (*sStart*, *sWait*, *sEnd*) $\mathcal{TA}$ is illustrated, whereas t and *vDly* are variables of the clock type and *fDly()* is a function that returns the value of a predefined random variable, i.e., a (pseudo) random number. The automaton starts in *sStart*. During *sStart* $\rightarrow$ *sWait*, x is reset and *vDly* set to the return value of *fDly()*; at this moment, *vDly* represents a random value. Staying in *sWait* cannot take longer than *vDly* units of time, whereas the time is measured using x while the progression of *vDly* is suspended (to capture the target/final time) by $vDly' == 0$; in other words, time of t passes until it reaches the value of *vDly*. Thus, *sWait* $\rightarrow$ *sEnd* is possible just if x matches *vDly*. Particular $\mathcal{TA}$ based instruments, allows one to generate pseudo random numbers following particular distributions of probability; typically, some of them are built-in, further ones can be added by a user. Also, it is possible to choose a fixed delay, choose the delay non-deterministically or to combine the approaches.

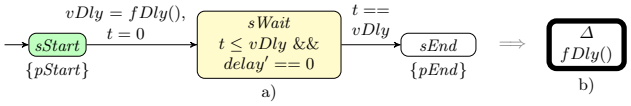


Fig. 8. The way we used to model a block (Δ) with a stochastically defined delay: a) full schema, b) simplified schema for increasing readability

Model Checking. On top of means for $\mathcal{M}$, further means can be used to formalize desired properties (φ) of a system and then to check whether such properties can imply ($\models$) from such a behavior, i.e., whether $\mathcal{M} \models \varphi$. We use a *model checker* (MC) to produce such a decision: if $\mathcal{M} \models \varphi$ holds, the checking finishes;

otherwise, the checker can produce a *counter-example* (CE), i.e., a sequence of timed actions leading to the violation of a property – see Fig. 9.

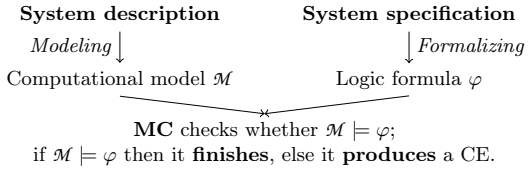


Fig. 9. Classical ("binary") model checking.

As a "classical" MC technique [5,11] is prone to the state space explosion and its decision capability is limited and binary (i.e., either $\mathcal{M} \models \varphi$ or $\mathcal{M} \not\models \varphi$), one may use a technique such as *statistical model checking* (SMC) to overcome disadvantages of MC [2,22]. Simply said, SMC conducts simulations over a stochastic model and monitors/processes them statistically to infer, with a predefined *degree of uncertainty* (ε), whether they provide a statistical evidence for the satisfaction of a property. SMC replaces the binarity regarding the satisfaction by the ability to quantify the impact of a change in a system for a predefined *degree of confidence* ($1 - \varepsilon$). SMC easily scale and has already been applied to solve various problems [20].

Query Language. For reasoning purposes, viewpoint, the *"classical" logic* [12] helps one to check whether a formula φ (ψ) is true or false for all/some realization(s) of logic and evaluation(s) of logic variables, while the so-called *temporal logic*, a special kind of the *modal logic* [5,11], distinguishes between various *modes of truth* e.g., necessarily true, known to be true, believed to be true or always true with respect to the given context (e.g., time). Using a (S)MC, one may check properties such as reachability, safety, liveness, possibility, invariance, potentiality, eventuality, probability estimation/comparison or hypothesis testing. To save space, we omit a general overview herein; instead, please consult Sect. 4 to find some queries we used to check key properties of our interest; to express such properties, quantifiers like $\exists$, $\forall$ and operators like $\Box$, $\Diamond$ or $\rightsquigarrow$ are typically used:

- $\exists\Diamond\varphi$ ("Possibly" property) holds if there is a reachable path in $G_{\mathcal{M}}$ such that φ possibly holds on the path,
- $\forall\Box\varphi$ ("Invariantly" property) holds if each reachable path in $G_{\mathcal{M}}$ satisfies φ; its equivalent is $\neg\exists\Diamond\neg\varphi$,
- $\exists\Box\varphi$ ("Potentially always" prop.) holds if exists a reachable path in $G_{\mathcal{M}}$ such that φ always holds on the path,
- $\forall\Diamond\varphi$ ("Eventually" property) holds if φ possibly holds on all paths in $G_{\mathcal{M}}$; its equivalent is $\neg\exists\Box\neg\varphi$,

- $\varphi \rightsquigarrow \psi$ ("Leads to" property) holds if whenever φ holds, eventually ψ holds too; its equivalent is $\forall\Box(\varphi \Rightarrow \exists\Diamond\psi)$,
- $\varphi \rightsquigarrow_{\leq t} \psi$ (time-bounded "Leads to" property) holds if $\varphi \rightsquigarrow \psi$ holds in at most t units of time.
- `Pr[limit](`φ`)` (Probability Estimation) estimates the probability of a property φ being true given that the predicate `limit` is true,
- `E[limit;nos](expr)` (Value Estimation) estimates the value of an expression `expr` by running a given number of simulations `nos`.

In this paper, we focus mainly on quantitative MC based on the probability estimation query.

3 Proposed Approach

This section introduces key concepts of our approach to QDA. Firstly, Sect. 3.1, outlines key instruments we used to parameterize our model(s) as well as components of our model(s), each able to express a particular stochastic behavior such as the arrival of a fault, presence of a fault in a system and the duration of a reconfiguration process. Next, Sect. 3.2 presents our representative models.

3.1 Means of Parameterization and Expressing Stochastic Behavior

To make our approach highly flexible, we specified a set of parameters used to parameterize it. Their representatives are listed in the following text (blue information in brackets by each of them represents the implicit value we used in our approach):

- `NUNITS`(3) – # of functional units (modules),
- `NSPARES`(6) – # spare units,
- `SPARE_BASE`(`NUNITS`+0) – index at which a block of spares starts,
- `MAX_FAULTS`(6000) – max. # of faults,
- `MAX_FTYPES`(3) – max. # of fault types,
- `MAX_FDEFS`(12) – max. # of fault definitions,
- `MAX_FGDEFS`(`MAX_UNITS`+`MAX_SPARES`) – # of fault generators,
- `MAX_RDEFS`(`MAX_SPARES`) – # of recovery definitions,
- `MAX_PDIST`(32) – max. # of supported distributions of probability,
- `MAX_PATTR`(4) – max. # of attributes per a probability distribution etc.

To express a desired stochastic behavior, one must have a set of properly-selected random variables, each defined by its distribution of probability. Using a probability distribution (`Prob Dist`), one can express multiple random variables such as fault occurrence/presence time (`Fault`) or reconfiguration time (`Reconf`). Each such a variable can be produced by multiple sources (`Fault Gen`, `Rec Ctrl`), each used to produce a predefined event, such as the occurrence of a fault or the end of a reconfiguration, for a predefined component of a system. In our approach, we implemented that using instruments outlined in List. 1.1–1.3. Particularly, List. 1.1 outlines instruments we used to characterize selected distributions of probability, i.e., their types defined by `pdist[i]` and attributes defined by `pattr[i]`.

Listing 1.1. An illustration to the characterization of probability distributions

```
1  const int PUNI=0, PEXP=1,PNOR=2,PWEI=3,PPOI=4,PBET=5,PGAM=6,PTRI=7,PARC=8;
2  typedef int[0, MAX_PDIST-1] t_pdist;
3  typedef int[0, MAX_PATTR-1] t_pattr;
4  // list of prob. dist. (set their attributes via pattr[])
5  const t_pdist pdist[t_pdist] = {  PNOR, PUNI, PNOR, PEXP, ... };
6  // attributes of prob. dist. from pdist[]
7  const double pattr[t_pdist][t_pattr] = {
8  /* scale, attr0, attr1, attr2, ... */
9      {1.0, 3000.0, 50.0, 0.0},    // attrs of pdist[0]
10     {1.0, 5000.0, 0.0, 0.0},     // attrs of pdist[1]
11     {1.0, 1000.0, 10.0, 0.0},    // attrs of pdist[2]
12     {1.0, 0.5, 0.0, 0.0},        // attrs of pdist[3]
13     ... };
```

List. 1.2 shows a way we use to characterize a fault. Each fault (see `t_sfault`) is characterized by its type (`ft`), time of its occurrence (`pttf`) and time of its leaving (`pttd`). To facilitate the access to the characteristics, we use functions such as `ftype()` and `pttf()` – see lines 16, 17.

Listing 1.2. An illustration to the characterization of faults

```
1  const int FPERM=0, FTRAN=1, FINTER=2;
2  typedef struct {
3     int ft;       // fault type: 0-perm, 1,tran., 2-interm.
4     int pttf;     // time-to-fault ID
5     int pttd;     // time-to-disappear ID
6  } t_sfault;
7  // a fault is defined by its t_sFault:
8  const t_sFault fdef[] =  {
9      {FPERM, 0, 1},// perm.f., pdist[0], pdist[1]
10     {FTRAN, 1, 5},// trans.f., pdist[1], pdist[1]
11     {FTRAN, 2, 6},// trans.f., pdist[2], pdist[0]
12     ... };
13 // a fault generator is defined by an index to fdef
14 const t_nfault fid[] = { 0, 1, 2, 0, 0, ... };
15 // auxiliary functions
16 t_ftype ftype(int gid){ return fdef[fid[gid]].ft; }
17 t_pdist pttf(int gid){return pdist[fdef[fid[gid]].pttf];}
18 ...
```

List. 1.3 shows a way we use to characterize a reconfiguration process. Analogically to `fdef[]`, we use `rdef[]` to characterize the reconfiguration time. Lines 3–8 illustrate instruments (`t_spares`) we used to characterize a particular reconfiguration mechanism – a spare. Each module (functional unit) may use a set of `cnt` spares, indexed from `first`. The setup of spares s_0, s_1, ... reserved for a particular module (functional unit) m indexed by $i = 0, 1, \ldots$ (i.e., m_i) of a system is stored in `sdef[i]`; line 9 shows the setup for a *Triple Modular Redundancy* (TMR) with no spares.

Listing 1.3. An illustration to the characterization of reconfiguration by spares

```
1  // assigns a pdist to a reconfiguration unit
2  const t_pdist rdef[t_nrecon] = { 14, 0, 6, ... };
3  const int SHOT=0, SWARM=1, SCOLD=2;
4  int stype[MAX_SPARES] = { 0, ... };
5  typedef struct {
6     int cnt;      // # spares per a unit
7     int first;    // offset to SPARE_BASE
8  } t_spares;
9  s_spares sdef[NUNITS] = {  {0,0}, {0,0}, {0,0} };
```

To express stochastic behavior of systems, we extend the instruments from Sect. 3.1 by means such as `ttf()`, `ttd()`, `ttr()`, i.e., functions that returns a random number following the probability distribution associated with their parameter, such as the id of a generator (`gid`), and id of a reconfiguration (`rid`), passed to the function. By means of such extensions and, using the mechanism

introduced in Fig. 8, we created models (Fig. 10) able to express stochastic behavior of systems we are interested in and representative models (Fig. 11) we used for the dependability assessment purposes.

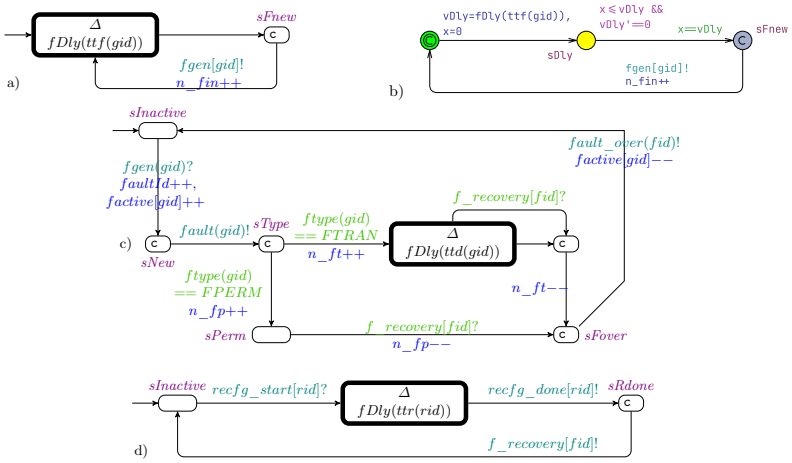


Fig. 10. Some of our applications of the mechanism from Fig. 8: a) tool-agnostic and b) UPPAAL-based fault generator/arrival models, c) fault-presence model, c) reconfiguration model. To make our consequent models more readable, we may omit the labeling function (l) in our further illustrations. Further, to clarify labels, we use UPPAAL's text-color schema: *violet* – a place, *teal* – a synchronization, *olive* – a select, *blue* – an action, *green olive* – a guard.

Fig. 10a shows our model of a fault generator identified by `gid`. After a random delay, given by `ttf(gid)` in Δ, it sends a synchronization via the `fgen[gid]` channel; then, it increments the counter of incoming faults and waits until new fault should occur. Fig. 10b depicts our model of a fault. After a fault gets activated via `fgen[gid]`, it moves to *sFnew* (`c` in its box denotes that the state is "committed", i.e., that time does no pass there). Then, it sends a synchronization via the `fault[gid]` channel to interact with models presented in Sect. 3.2 and enters *sType*. If the fault is permanent, it enters *sPerm* and waits there until its recovery is over; else, it enters Δ and waits there until one of the following events occur: time given by $fDly()$ expires, fault recovery completes. Then, it enters *sFover*, send a synchronization via `fault_over[fid]` and enters *sInactive*. Fig. 10c shows our model of a reconfiguration timing. Initially, the model waits in *sInactive* until the reconfiguration gets started. Then, it waits until the reconfiguration completes. Finally, it sends a synchronization via `recfg_done[rid]` and `f_recovery[fid]`.

3.2 Means for Constructing Dependability Models

Fig. 11 presents some of models we created for the dependability assessment purposes. To better understand Fig. 11, let us explain symbols we've not intro-

duced yet: `bad` – id of a faulty unit, `faulty[i]` – indicates whether the component indexed by i is faulty, `s_On(i)` – indicates whether the spare indexed by i is active, `isRdy(i)` – returns `true` if i represents an active non-faulty component; else, it returns `false`, `nOK()` – returns # of components i for which `isRdy(i)` is `false`, `chkSpare()` – returns id of some non-faulty spare if it exists; else, it returns -1. Typically, attributes of dependability are evaluated for architectures, based on static/dynamic redundancy, such as *Simplex* (SPX), *Duplex* (DPX), *Triple Modular Redundancy* (TMR), *Triplex with Successive Degradation* (TSD), *Triplex To Simplex* (TTS), *Triplex with 1 Spare* (T1S) – for details, refer, e.g., to [30].

Models for QDA over representative architectures. Fig. 11a illustrates a SPX – it waits in *sOK* until the only module (indexed by 0) in a system fails; then, it stores the id of the failed module into *bad* (we skip to comment this action w.r.t. further models) and enters *sFail*. Fig. 11b illustrates a DPX – it waits in *sOK* until one of the two modules (indexed by $0, 1$) in a system fails; then, it enters *sFail*. Fig. 11c illustrates a TMR – it waits in *sOK* until one of the three modules (indexed by $0, 1, 2$) in a system fails; then, it enters

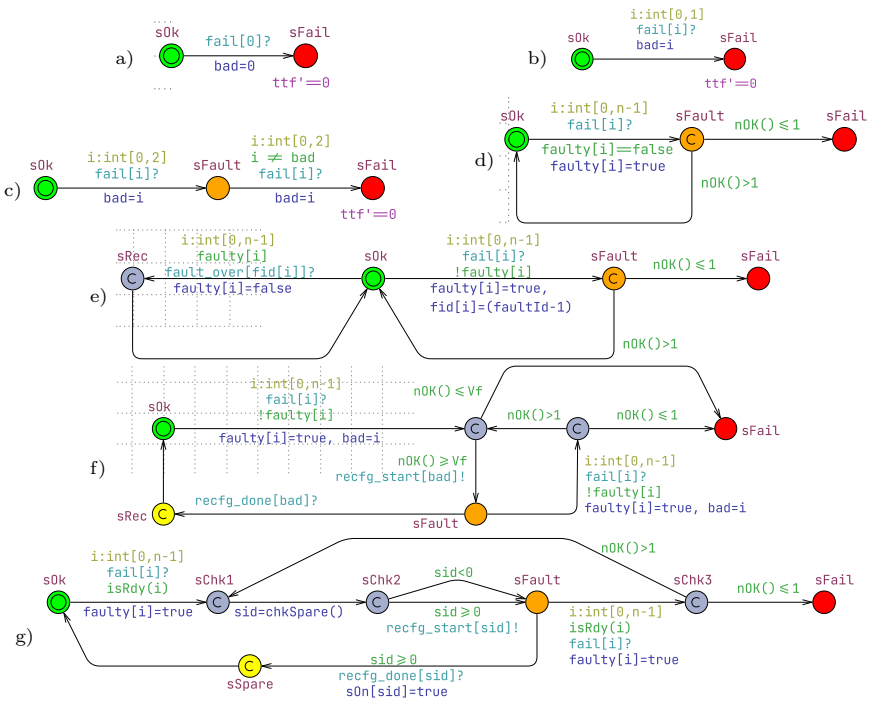


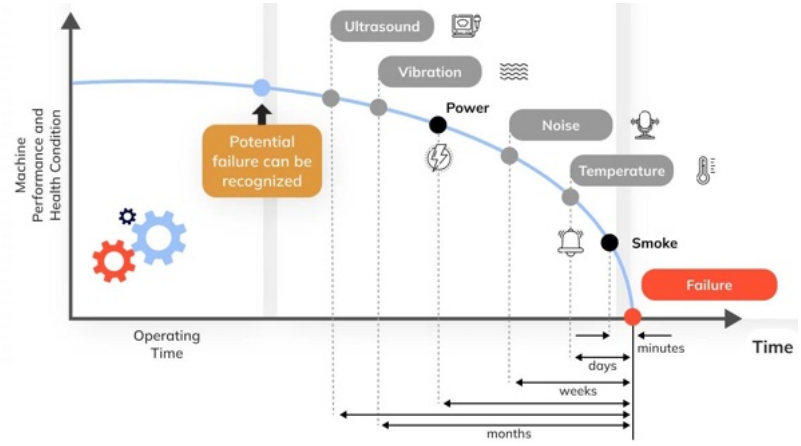
Fig. 11. Examples for clarifying our parameterizable models (a–c particular, d–g generalized): a) SPX, b) DPX, c) TMR, d) NMR, e) NMRT, f) NSD ($V_F = 0$) and NTS ($V_F = 1$), h) NMR with spares (NMRS). To increase readability, we use the following color schema for the background color of a place: ■ – the OK (fault-free) state of a system, ■ – a fault in a system, ■ – a failure of a system, ■ – a recovery activation.

sFault. Here, it waits until one of the remaining two units fails; then, it enters *sFail*. Fig. 11d generalizes TMR to model *N-modular redundancy* (NMR) that works analogically to Fig. 11c. In *sOK*, it waits until a non-faulty module (out of n modules indexed from 0) in a system fails; then, it enters *sFault*. Out of there, it moves based on the result of $nOK()$ – if the number of remaining non-faulty modules is above 1, it enters *sOk*; else *sFail*. Fig. 11e presents our *NMR* model *sensitive to transient faults* (NMRT). Alike Fig. 11d, it originates from Fig. 11c, but extends it by the possibility of the limited staying of a fault in a system. Comparing to a permanent fault (that stays in a system until it gets removed), a transient fault stays in a system for some predefined time (given by waiting for the synchronization via `fault_over`) and then, it "disappears". Thus, the following sequence may happen (# of non-faulty modules exceeds 2): the model starts in *sOk*, a module (i) gets faulty, system enters *sFault*, then *sOk*. If it gets synchronized (about i) via `fault_over` before further module gets faulty, the faulty module becomes non-faulty and the model moves to *sOk*. Fig. 11f generalizes TSD, TTS to N-modular redundancy, NSD and NTS, resp. – the models differ in the setup of V_F only. The model starts in *sOk*. Here it waits until a module gets faulty, then it enters *sChk1* – NSD, NTS moves to *sFail* if $nOK() \leq 0$ and $nOK() \leq 1$, resp.; else, it starts the reconfiguration of the faulty module and enters *sFault* – the minimal amount of non-faulty modules is 1 (SPX) and 2 (DPX) for NSD and NTS, resp. Here it waits until one of the following events happen: the reconf. completes (way to *sRec*, then *sOk*) or further non-faulty module fails (way to *sChk2*). In *sChk2*, it is decided if there is enough non-faulty modules. If so, the model moves to *sChk1*, else it enters *sFail*. Fig. 11g illustrates our model of NMR with spares defined by `sdef[]`. The model starts in *sOk*. Here it waits until a module gets faulty, then it enters *sChk1*, gets information about the availability of non-faulty spares for a faulty module and enters *sChk2*. If a spare is available, the reconfiguration (of a faulty module to the spare) starts. The model moves to *sFault*; here it waits until one of the following events happen: the reconfiguration completes (way to *sSpare*, then *sOk*) or further non-faulty module fails (way to *sChk3*). In *sChk3*, it is decided if there is enough non-faulty modules in a system. If so, the model moves to *sChk1*, else it enters *sFail*.

Models for QDA focused on M/A – LoLiPoP-IoT specific models. As mentioned at the end of Sect. 2.1, each approach to maintenance (M) control has its benefits/pros and costs/contras. For example [27], most PdM solutions are deployed either in a *High-Performance Environment* (HPE) such as a cloud, whereas data a HPE takes at it input are often produced by small sensor devices such as WSN. Those devices typically interact via a communication infrastructure – a network –, so the data has to be collected and sent over a network for further processing. This suffers from many drawbacks, e.g., (security/privacy viewpoint) data can be compromised when sent over a network, (latency viewpoint) network communication may induce a non-zero or unpredictable latency, which can be unacceptable for some apps, (availability, A, viewpoint) network

communication may get unavailable in sparsely populated etc. areas, (energy viewpoint) a battery-operated device may spend a lot of energy by sending data to/from a HPE instead of being processed on the device.

Regarding the LoLiPoP-IoT project, QDA solutions are expected to be applied in a context particular Use Cases (UCs) – in the following text, let us demonstrate the applicability in the two context: i) an engine operation and ii) battery operated (WSN, IoT etc.) devices. Firstly, let us briefly introduce the engine context. Fig. 12 illustrates that, after started, a fault-free engine operates as intended, *normal* (NRM), until it starts to produce an *ultrasound* (US) noise – this happens, e.g., after about 10 years of running in NRM.



Source: https://intellisoft.io/predictive-maintenance-iot-your-path-to-efficiency/

Fig. 12. Illustration to anomaly detection and maintenance with regard to an engine

In the order of months, the engine may, sequentially, produce *vibrations* (VIB) and need *high power* (HP) – the ability to detect such phenomena allows one to predict the failure much longer before it can occur. Then, the engine may produce an audible *noise* (NSE) and start to operate at a *high temperature* (HT) few weeks or days before the failure, resp. After it starts to *smoke* (SMK), it can fail in minutes. List. 1.4 summarizes the associated definitions.

Listing 1.4. Key definitions we used to build our model of an engine

```
const int N_ENGINE_STATES = 9;                  // # states of an engine
typedef int[0, N_ENGINE_STATES-1] t_eStates;
// engine behavior/state                        time to failure ...
const int OFF = 0;       // off
const int NRM = 1;       // normal              ... 10 years
const int US = 2;        // ultrasound          ... 10 months
const int VIB = 3;       // vibrations          ... 5 months
const int HP = 4;        // high power          ... 2 months
const int NSE = 5;       // noise               ... 4 weeks
const int HT = 6;        // high temperature    ... 7 days
const int SMK = 7;       // smoke               ... 15 minutes
const int FAIL = 8;      // failure
const int START = OFF;  // starting (initial) state of an engine
broadcast chan engAct[N_ENGINE_STATES]; // the channel used to signalize an event, a fault etc.
int eSt=START;                          // initial state of on engine
```

We expressed the behavior of an engine using an automaton from Fig. 13a. The engine starts in *sOff*. After it gets on, it moves to *sNormal*; if an US noise gets produced there, it moves to *sUltrasnd* etc. until it reaches *sFail*. If an anomalous behavior is detected in a state from *sUltrasound* to *sSmoke*, the corresponding repair process is initiated (Fig. 13b) – in general, the price/cost (in terms such as means, time, complexity and observable effects) of the process increases when approaching to *sFail*. If the repair finishes before a new fault occurs, the corresponding transition (with *rdone?*) fires and the model moves to *sOK* and then to *sNormal*. Timing of transitions between the engine states is controlled by the model from Fig. 13c – the model waits in *sWait* until there is time (random variable parameterized, e.g., by definitions from List. 1.4) for the state change.

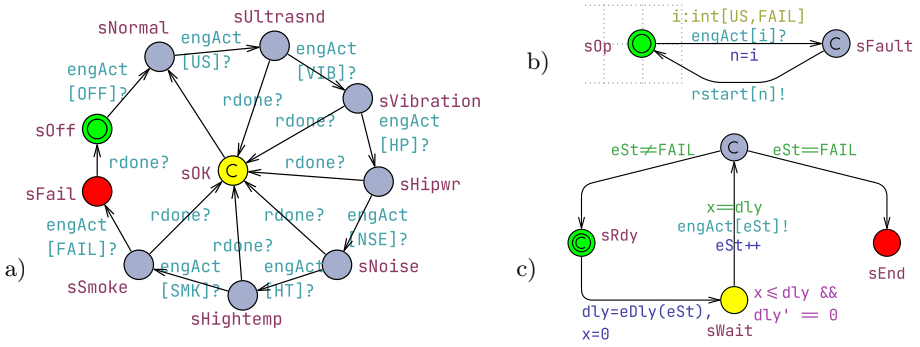


Fig. 13. Our models of a) the operation of an engine, ideal starting of its repair and c) its state transition process

To illustrate our approach to modeling a maintenance policy (see Fig. 7), we present skeletons of the models we used to implement its key processes: a) inspection and b) repair. The former one is responsible for initiating a maintenance – it waits in *sOff* until the maintenance period expires (then, it goes via *sRegIns* to start a regular maintenance) or a monitored condition gets satisfied (then, it goes via *sCallR* to start an extra-ordinary maintenance). The latter model reacts to stimuli from the former one; if signalized, it prepares for the repair process (*sPrepare*) and then, it starts the corresponding repair.

Below, let us introduce our QDA approach in the context of a battery operated devices, such as WSN/IoT nodes, typically used to communicate via a

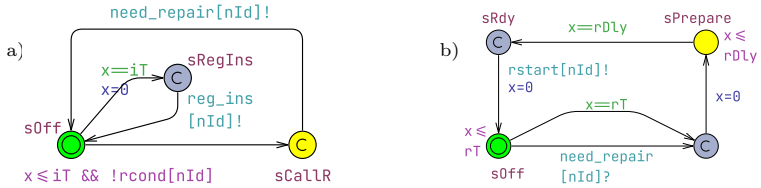


Fig. 14. Skeletons of our models of a) inspection and b) repair modules

network to provide a required service. From the viewpoint of such a device/network, one may use various power/operating modes, e.g., *Run*, *Low Power* (LP) and *Very Low Power* (VLP), to control the consumption of energy in various situations, e.g., single-node computation, multi-node communication, parameters/state of a battery. Using a proper power management needs a more sophisticated control and more time in general, but implies positive consequences such as the prolonged mission time (due to a better battery utilization), decrease of material stress (due to a lower electrical current and heat dissipation), increase of a battery life (due to a proper control of the charging process) etc. Typically, such a device uses mechanism such as *Dynamic Voltage and Frequency Scaling* (DVFS), *Clock Gating* (CG) and *State Retention* of internal logic, I/O (Inputs/Outputs) and RAM (Random Access Memory). List. 1.5 shows key definitions we used to create our models of power operated devices.

Listing 1.5. Key definitions we used to describe the power modes of a device and scaling of the frequency and exe-time

```
1  const int wRun = 10     /* probability weights (w) of power modes */
2  const int wLP = 70;
3  const int wVLP = 20;
4  int32_t mCnt[N_PWRMODES]; /* power mode transition counter */
5  double pwrtd[N_PWRMODES][N_PWRMODES] = { /* power mode transition delays */
6  // to Run        Wait    Sleep    <- target / source -v
7      {  0,        1,      10 }     // from Run
8      {  1,        0,      5 },     // from Low Power (LP)
9      { 25,        20,     0 }      // from Very Low Power (VLP)
10 };
11 const double pwrf[N_PWRMODES] = { 100, 25, 1 };          // mode freq. ratio (in %)
12 double tsc() { return 100.0/pwrf[nattr[id].pm]; }        // exe-time scaling factor of a mode
```

Particularly, it List. 1.5 defines the constants `wRun`, `wLP` and `wVLP` to define stochasticity of the power/operating mode switching process, `pwrtd[]` to define the delay of switching from a source to a target mode, `pwrf[]` to define the ratio of maximum operating frequency used in a mode and `tsc()` to compute how the execution time scales when a mode is used. Based on the definitions, we created models (Fig. 15) to express reality of our interest.

Fig. 15a outlines (a skeleton of) our model of a battery operated device. It starts in *sRdy* the goal of which is to fetch an operation (to be executed), scale it by `tsc()` corresponding to the recent mode (the initial mode is Run) and consume the corresponding amount of energy (E) at a predefined power (P) in *sWork*. The operation itself, e.g., the increment of a value, executes in

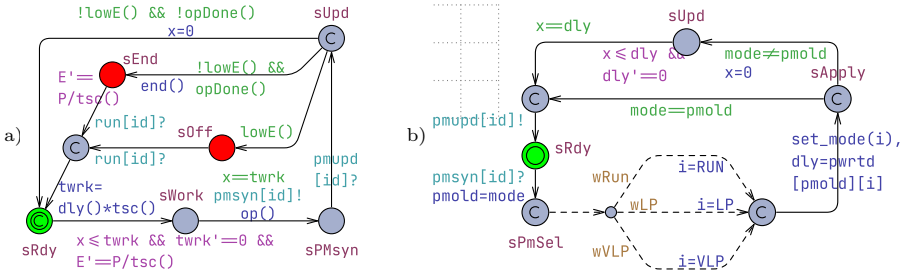


Fig. 15. Base of our models of a) a battery operated device and b) its power manager

`op()` on the transition *sWork*→*sPMsyn*, along with sending a synchronization to the power manager to allow it to switch (update) the mode. Then, it moves to *sUpd*. From there, it goes to *sRdy* if there is enough energy and the predefined goal, e.g., finishing a computation, based on the operation has not been reached yet. Otherwise, if the goal is reached, it enters *sEnd* or, if the energy is low, it moves to *sOff* to minimize the consumption of energy; then, it waits for the next command. Fig. 15b illustrates (a skeleton of) our model of the power manager of a battery operated device. It starts in *sRdy* and waits there until gets synchronized by its device. Then, it moves to *sPmSel* to select the next mode and moves to *sApply*. Out of there, it goes via *sUpd* if the target mode differs from the source mode (in such a situation, it takes some time to switch); otherwise, no switch is needed as the source mode is the target one.

4 Evaluation

This section summarizes representative results to show the kind of results achievable by our QDA approach and also, it briefly evaluates it. As mentioned before (Sect. 2.3), we used Uppaal both to create our models (Sect. 3) using TA, and to analyze them using SMC. Let us recall that at its input, the Uppaal takes a model and a query about a property of the model to be checked; at its output, it produces data such as PDF, CDF or mean w.r.t. the property, whereas the data quality is driven by parameters such as the probability uncertainty (ε), configurable by a user before the checking starts. Particularly, a query may ask the SMC engine to evaluate the probability of entering $sFail$ (see Fig. 11). Such a query is of the probability estimation type and its form is $Pr[bound](\phi)$, where *bound* defines how to bound simulation steps/runs and ϕ represents a property to be checked; e.g., for a model M to be examined within 10^5 units of time, the query would be $Pr[<= 100000](<> M.sFail)$.

Initial validation. Firstly, we validated our QDA approach by comparing its results with results of existing analytical solutions to the QDA problem – Fig. 16 shows such a comparison for SPX and an exponentially distributed X_{TTF}.

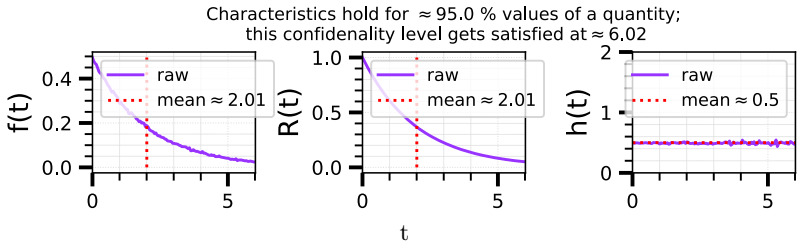


Fig. 16. An illustration to the validation of our results versus the analytical solution for SPX and X_{TTF} distributed exponentially with $\lambda = 0.5$

Until reaching cca 95% confidentiality level, simulation results were comparable to the results of analytical solutions. For higher levels, results got less accurate due to a need to simulate rare-event scenarios. However, we may conclude that our approach is able to produce sufficiently accurate results in a reasonable time.

QDA results for selected kinds of redundancy. Secondly, we evaluated the $Pr[<= 25000](<> M.sFail)$ query for models from Fig. 11. Apparently (Fig. 17), results of our approach aim to decide, in given conditions, which of the modeled redundancy-based architectures is best from the dependability viewpoint – apparently, in Fig. 17, the "winner" is NMRS with its biggest MTBF ≈ 3100.

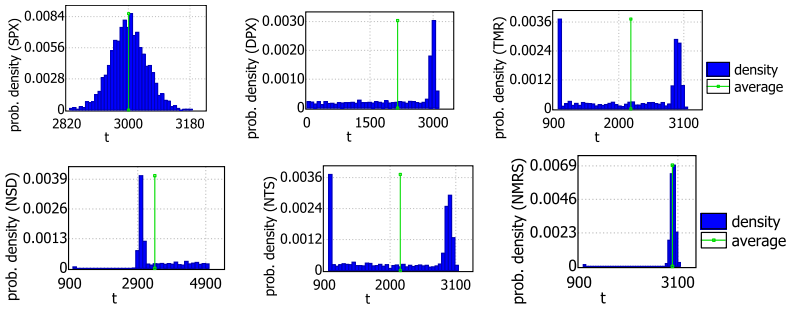


Fig. 17. PDF examples for the failure event in representative systems, in the same conditions (see Sect. 3.1); SPX is included for an interest only – it does not involve any means of dependability control.

Repair rate impact study. Thirdly, we show (Fig. 18) that, for a given architecture and conditions, our approach aims one do decide what reconfiguration process is the most suitable for achieving the desired level of maintainability, availability etc. Fig. 18 compares effects of three reconfiguration processes, out of which the best one is c), characterized by the shortest TTR and consequently, the biggest MTBF, uptime etc.

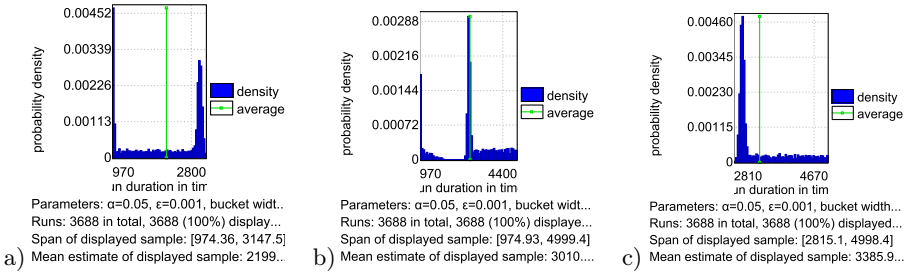


Fig. 18. Ilustration of an impact of various repair rates (RR) to dependability attributes of a NSD system for RRs characterized by the a) normal/Gaussian PDF w. attr. 15000, 2500, b) uniform PDF w. attr. 1000, c) Beta PDF w. attr. 0.1, 0.1.

Engine failures. Regarding Fig. 13, we checked the probability of entering a particular faulty state of an engine – for representative results, see Fig. 19. This kind of an experimental result may help one to identify MTTF for particular faults and consequently, to adjust properly corresponding repair mechanisms in order to set a ALARP (As Low As Reasonably Practical) value of MTTR and consequently, maximize MTBF, uptime etc.

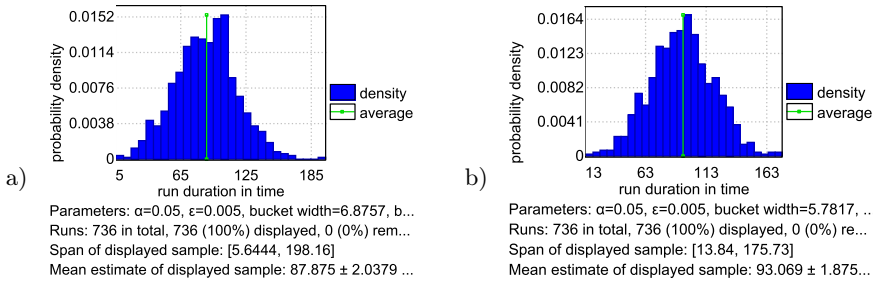


Fig. 19. Probabiblity density function (PDF) of a) producing an ultrasound, i.e., `Pr[<=2500] (<>eSt==US)`, and b) a failure, i.e., `Pr[<=2500] (<>eSt==FAIL)`; time is in 10^3 hours (kHrs), whereas a week $\approx$ 0.1 kHrs, a month $\approx$ 5 kHrs, a year $\approx$ 11 kHrs

Battery operated devices. Regarding Fig. 15, we show our approach can be used to analyze effects of using various power-management scenarios – Fig. 20 gives an example of a WSN/IoT node used to perform a task upon a request, i.e., to gradually increment its state variable from 0 (initial state) to 16 (goal/target state) by 1 each 10 units of time (ideally, this finishes at 160). Depending on the power-manag. policy used, various modes can be used (red line) until the task completes – based on Fig. 15, the three supported modes are Run (lowest level), LP (middle level), VLP (highest level); then the mode switches to VLP.

Apparently, it takes the longest/shortest time to finish the task in a)/c), resp.; the magenta line marks the goal state to be reached and the green line shows the evolution of the state variable in time. However, RBE (Remaining Battery Energy) remains the highest after finishing the task in c), probably due to the mode-switching costs, which is more apparent in a) than in b).

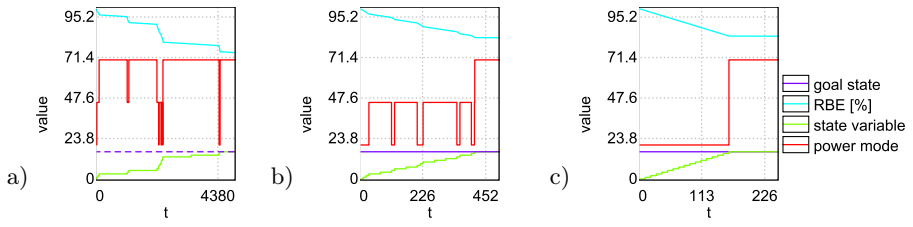


Fig. 20. Effects of using various power management scenarios: a) power-optimized: wRun = 10, wLP = 75, wVLP = 15, b) power-aware: wRun = 40, wLP = 60, wVLP = 0, c) hi-performance: wRun = 100, wLP = wVLP = 0; for details, consult Fig. 15

Scalability analysis. Finally, we evaluated (Fig. 21) the impact of the probability uncertainty (ε), e.g., to the process of checking the $Pr[<= 25000](<> M.sFail)$ query for models from Fig. 11. Fig. 21a,c, shows cca linear dependency while Fig. 21b shows (over-)linear dependency. We may conclude that our QDA approach based on TA/SMC scales well for all scenarios we studied. In general, however, it might get worse in some specific scenarios and must be solved specifically, e.g. using *importance sampling* and *splitting* [18,21] used for dealing with rare event situations.

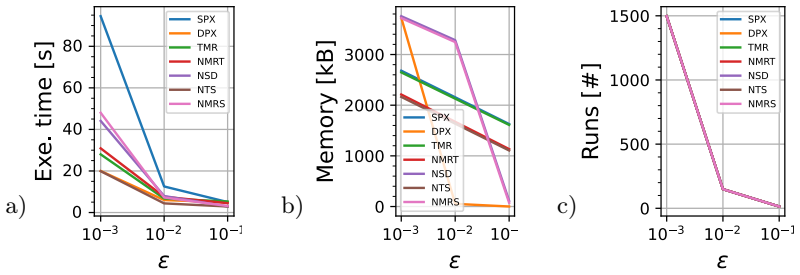


Fig. 21. Impact of ε to the following parameters of our QDA approach: a) time to execute simulation runs, b) mem. consumption, c) # of simul. runs

5 Conclusion

In the paper, we have outlined our simulation-based approach to the quantitative dependability assessment (QDA) problem in the context of the LoLiPoP-IoT project. Our approach builds on the two instruments – firstly, on Stochastic/Hybrid Timed Automata (TA) we used to model reality of our interest; secondly, on Statistical Model Checking (SMC) to evaluate properties of modeled reality in specified conditions.

Paper summary. At its beginning, the paper introduces its context, i.e., the LoLiPoP-IoT project, summarizes key preliminaries, including facts such as dependability and its assessment, with a special attention paid to its qualitative (QDA) form, related work in the areas of our interest and the means/methods we used in our approach. Then, the paper presents our approach to constructing models for reality of our interest such as faults, redundancy-based architectures (e.g., duplex, triplex), repair and maintenance mechanisms (e.g., corrective/reactive, condition-monitoring based, preventive/scheduled) w./w.o. degradation, spares etc. Next, it demonstrates the applicability of our approach in the two particular contexts – an engine operation and battery operated (WSN, IoT etc.) devices. Finally, the paper summarizes representative results we've achieved so far based on the models to show the applicability of our QDA approach. Among the others, the results help one to decide what kind of redundancy, repair and

maintenance policy are the most appropriate for predefined conditions and application in order to maximize availability of a system's service etc., whereas efficiency and accuracy of our approach are controllable by parameters such as the probability uncertainty (ε).

Research perspectives. Among the others, our research plans include problematic aspects such as the quantitative assessment under dependencies among faults, dynamics and state-dependent behavior of faults, bathtub-shaped failure rates – reflecting phenomena such as failure hump, technology or stress –, maintenance optimization and planning, optimization of power/energy management and consumption and of battery life and, finally, applications.

References

1. Abate, A., Budde, C.E., Cauchi, N., van Harmelen, A., Hoque, K.A., Stoelinga, M.: Modelling Smart Buildings Using Fault Maintenance Trees. In: Bakhshi, R., Ballarini, P., Barbot, B., Castel-Taleb, H., Remke, A. (eds.) Computer Performance Engineering. pp. 110–125. Springer International Publishing, Cham (2018)
2. Agha, G., Palmskog, K.: A Survey of Statistical Model Checking. ACM Transactions on Modeling and Computer Simulation (TOMACS) **28**(1), 6:1–6:39 (Jan 2018). https://doi.org/10.1145/3158668
3. Alur, R., Dill, D.: The Theory of Timed Automata. In: Real-Time: Theory in Practice. pp. 45–73. Springer, Berlin, Heidelberg (1992). https://doi.org/10.1016/0304-3975(94)90010-8
4. Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (2004). https://doi.org/10.1109/TDSC.2004.2
5. Baier, C., Katoen, J.P.: Principles of Model Checking. Representation and Mind, MIT Press (2008)
6. Behrmann, G., David, A., Larsen, K.: A Tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
7. Bowles, J.B.: Commentary - Caution: Constant Failure-Rate Models May be Hazardous to Your Design. IEEE Transactions on Reliability **51**(3), 375–377 (Sept 2002). https://doi.org/10.1109/TR.2002.801850
8. Calinescu, R., Ghezzi, C., Johnson, K., Pezz, M., Rafiq, Y., Tamburrelli, G.: Formal Verification With Confidence Intervals to Establish Quality of Service Properties of Software Systems. IEEE Transactions on Reliability **65**(1), 107–125 (March 2016). https://doi.org/10.1109/TR.2015.2452931
9. Calinescu, R., Ghezzi, C., Johnson, K., Pezze, M., Rafiq, Y., Tamburrelli, G.: Formal Verification With Confidence Intervals to Establish Quality of Service Properties of Software Systems. IEEE Transactions on Reliability **PP**(99), 1–19 (2015). https://doi.org/10.1109/TR.2015.2452931
10. Cinar, Z.M., Nuhu, A.A., Zeeshan, Q., Korhan, O., Asmael, M.B.A., Safaei, B.: Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. Sustainability (2020), `https://api.semanticscholar.org/CorpusID:225160331`

11. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking. Springer International Publishing, Cham, 1st edn. (2018). https://doi.org/10.1007/978-3-319-10575-8
12. van Dalen, D.: Logic and Structure. Universitext, Springer Verlag, London, 5th edn. (2013). https://doi.org/10.1007/978-1-4471-4558-5
13. David, A., Larsen, K., Legay, A., Mikučionis, M., Poulsen, D.: Uppaal SMC Tutorial. Int. Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015). https://doi.org/10.1007/s10009-014-0361-y
14. Devooght, J.: Dynamic Reliability. Advances in Nuclear Science and Technology **25**, 215–278 (1997). https://doi.org/10.1007/0-306-47812-9_7
15. Durga Rao, K., Gopika, V., Sanyasi Rao, V., Kushwaha, H., Verma, A., Srividya, A.: Dynamic Fault Tree Analysis using Monte Carlo Simulation in Probabilistic Safety Assessment. Reliability Engineering and System Safety **94**(4), 872–883 (2009). https://doi.org/10.1016/j.ress.2008.09.007
16. Geffroy, J.C., Motet, G.: Design of Dependable Computing Systems. Kluwer Academic Publishers, Hingham, MA, USA (2002)
17. Hartmanns, A., Hermanns, H.: In the Quantitative Automata ZOO. Science of Computer Programming **112**, 3–23 (2015). https://doi.org/10.1016/j.scico.2015.08.009
18. Jégourel, C., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Importance Sampling for Stochastic Timed Automata. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA. LNCS, vol. 9984, pp. 163–178 (2016). https://doi.org/10.1007/978-3-319-47677-3_11
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. SIGMETRICS Perform. Eval. Rev. **36**(4), 40–45 (Mar 2009). https://doi.org/10.1145/1530873.1530882
20. Larsen, K.G., Legay, A.: Statistical Model Checking Past, Present, and Future. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications. pp. 135–142. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45231-8_10
21. Larsen, K., Legay, A., Mikučionis, M., Poulsen, D.: Importance Splitting in Uppaal. In: Proc. 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA). pp. 433–447. LNCS, Physica-Verlag (2022). https://doi.org/10.1007/978-3-031-19759-8_26
22. Legay, A., Delahaye, B., Bensalem, S.: Statistical Model Checking: An Overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) Runtime Verification. pp. 122–135. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-16612-9_11
23. Liu, Y., Ren, Y., Liu, L., Li, Z.: A Spark-Based Parallel Simulation Approach for Repairable System. vol. 2016-April (2016). https://doi.org/10.1109/RAMS.2016.7447965
24. Lu, Y., Miller, A.A., Hoffmann, R., Johnson, C.W.: Towards the Automated Verification of Weibull Distributions for System Failure Rates, pp. 81–96. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45943-1_6
25. Lu, Y., Peng, Z., Miller, A.A., Zhao, T., Johnson, C.W.: How Reliable is Satellite Navigation for Aviation? Checking Availability Properties with Probabilistic Verification. Reliability Engineering & System Safety **144**, 95 – 116 (2015). https://doi.org/10.1016/j.ress.2015.07.020

26. Nekoukhou, V., Bidram, H.: A New Generalization of the Weibull-Geometric Distribution with Bathtub Failure Rate. Communications in Statistics - Theory and Methods **46**(9), 4296–4310 (2017). https://doi.org/10.1080/03610926.2015.1081949
27. Njor, E., Madsen, J., Fafoutis, X.: A Primer for tinyML Predictive Maintenance: Input and Model Optimisation. In: Artificial Intelligence Applications and Innovations. pp. 67–78. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-08337-2_6
28. Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., Loncarski, J.: Machine Learning approach for Predictive Maintenance in Industry 4.0. In: 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA). pp. 1–6 (2018). https://doi.org/10.1109/MESA.2018.8449150
29. Peng, Z., Lu, Y., Miller, A., Johnson, C., Zhao, T.: A Probabilistic Model Checking Approach to Analysing Reliability, Availability, and Maintainability of a Single Satellite System. In: Modelling Symposium (EMS), 2013 European. pp. 611–616 (Nov 2013). https://doi.org/10.1109/EMS.2013.102
30. Ricky W., B., Sally C., J.: Techniques for Modeling the Reliability of Fault-Tolerant Systems With the Markov State-Space Approach. Tech. rep. (1995), `http://shemesh.larc.nasa.gov/fm/papers/Butler-RP-1348-Techniques-Model_Rel-FT.pdf`
31. Ruijters, E., Guck, D., Drolenga, P., Stoelinga, M.: Fault Maintenance Trees: Reliability Centered Maintenance via Statistical Model Checking. In: 2016 Annual Reliability and Maintainability Symposium (RAMS). pp. 1–6 (2016). https://doi.org/10.1109/RAMS.2016.7447986
32. Smrz, P., et al.: LoLiPoP IoT (Long Life Power Platforms for Internet of Things), Part B for Grant Agreement (June 2023). https://doi.org/10.3030/101112286, `https://www.lolipop-iot.eu/`
33. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. NUREG, Washington, US (1981), `https://archive.org/details/nureg-0492-ml100780465/NUREG-0492_ML100780465/`, id: NUREG-0492
34. Xing, L., Amari, S.V.: Fault Tree Analysis, pp. 595–620. Springer London, London (2008). https://doi.org/10.1007/978-1-84800-131-2_38
35. Zhang, T., Dwight, R., El-Akruti, K.: On a Weibull Related Distribution Model with Decreasing, Increasing and Upside-Down Bathtub-Shaped Failure Rate. In: 2013 Proceedings Annual Reliability and Maintainability Symposium (RAMS). pp. 1–6 (Jan 2013). https://doi.org/10.1109/RAMS.2013.6517749
36. Zhu, P., Han, J., Liu, L., Lombardi, F.: Reliability Evaluation of Phased-Mission Systems Using Stochastic Computation. IEEE Transactions on Reliability **65**(3), 1612–1623 (2016). https://doi.org/10.1109/TR.2016.2570565
37. Zhu, P., Han, J., Liu, L., Zuo, M.: A Stochastic Approach for the Analysis of Fault Trees with Priority and Gates. IEEE Transactions on Reliability **63**(2), 480–494 (2014). https://doi.org/10.1109/TR.2014.2313796
38. Zhu, T., Ran, Y., Zhou, X., Wen, Y.: A Survey of Predictive Maintenance: Systems, Purposes and Approaches. arXiv e-prints arXiv:1912.07383 (Dec 2019). https://doi.org/10.48550/arXiv.1912.07383